Übungen - Bildgenierung

Übung 09.

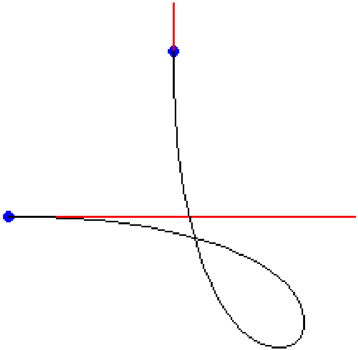
Jose Jimenez

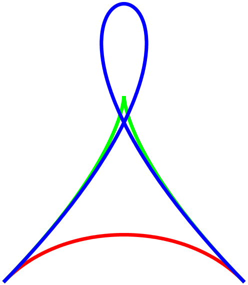
Angewandte Informatik
Bergische Universität Wuppertal

December 18, 2024

Table of Contents

Hermite-Kurven Schleife





Was wir gerade gelernt haben ...
Und Rahmen Program.

Hermite-Kurven

In der Vorlesung habt ihr die Hermite-Basis-Polynome kennengelernt

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

Hermite-Kurven

In der Vorlesung habt ihr die Hermite-Basis-Polynome kennengelernt

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

Der Parameter ist t. Schreiben wir $Q(t)$ als **Polynom**.

$$Q(t) = t^3(2P_1 - 2P_4 + R_1 + R_4) + t^2(-3P_1 + 3P_4 - 2R_1 - R_4) + t(R_1) + P_1$$

Hermite-Kurven

In der Vorlesung habt ihr die Hermite-Basis-Polynome kennengelernt

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

Der Parameter ist t. Schreiben wir $Q(t)$ als **Polynom**.

$$Q(t) = t^3(2P_1 - 2P_4 + R_1 + R_4) + t^2(-3P_1 + 3P_4 - 2R_1 - R_4) + t(R_1) + P_1$$

Der Parameter ist t, dannn:

$$Q(t) = t^3 \underbrace{(2P_1 - 2P_4 + R_1 + R_4)}_{c_0} + t^2 \underbrace{(-3P_1 + 3P_4 - 2R_1 - R_4)}_{c_1} + t \underbrace{(R_1)}_{c_2} + \underbrace{P_1}_{c_3}$$

Bis hier: Hermite-Basis-Polynome.

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

$$Q(t) = t^3 \underbrace{(2P_1 - 2P_4 + R_1 + R_4)}_{c_0} + t^2 \underbrace{(-3P_1 + 3P_4 - 2R_1 - R_4)}_{c_1} + t \underbrace{(R_1)}_{c_2} + \underbrace{P_1}_{c_3}$$

$$Q(t) = c_0 t^3 + c_1 t^2 + c_2 t + c_3$$

Gut, aber wir haben: **Neun Multiplikationen** 😟...

Bis hier: Hermite-Basis-Polynome.

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

$$Q(t) = t^3 \underbrace{(2P_1 - 2P_4 + R_1 + R_4)}_{c_0} + t^2 \underbrace{(-3P_1 + 3P_4 - 2R_1 - R_4)}_{c_1} + t \underbrace{(R_1)}_{c_2} + \underbrace{P_1}_{c_3}$$

$$Q(t) = c_0 t^3 + c_1 t^2 + c_2 t + c_3$$

Gut, aber wir haben: **Neun Multiplikationen** 😟...

$$Q(t) = (c_0 t^2 + c_1 t + c_2)t + c_3$$

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Drei Multiplikationen!

Hermite-Kurven

Wir haben also $Q(t)$ als Polynom mit den Koeffiziente c_i.

$$Q(t) = t^3 \underbrace{(2P_1 - 2P_4 + R_1 + R_4)}_{c_0} + t^2 \underbrace{(-3P_1 + 3P_4 - 2R_1 - R_4)}_{c_1} + t \underbrace{(R_1)}_{c_2} + \underbrace{P_1}_{c_3}$$

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

c_i sind Vektoren (Punkte), d.h., $c_i(x, y)$.

Hermite-Kurven

Wir haben also $Q(t)$ als Polynom mit den Koeffiziente c_i.

$$Q(t) = t^3 \underbrace{(2P_1 - 2P_4 + R_1 + R_4)}_{c_0} + t^2 \underbrace{(-3P_1 + 3P_4 - 2R_1 - R_4)}_{c_1} + t \underbrace{(R_1)}_{c_2} + \underbrace{P_1}_{c_3}$$

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

c_i sind Vektoren (Punkte), d.h., $c_i(x, y)$. In C++

```
double cx[4], cy[4];
// c_x
  cx[0] = 2 * p1.x - 2 * p4.x + r1.x + r4.x;
  cx[1] = -3 * p1.x + 3 * p4.x - 2 * r1.x - r4.x;
  cx[2] = r1.x;
  cx[3] = p1.x;
//c_y
  cy[0] = 2 * p1.y - 2 * p4.y + r1.y + r4.y;
  cy[1] = -3 * p1.y + 3 * p4.y - 2 * r1.y - r4.y;
  cy[2] = r1.y;
  cy[3] = p1.y;
```

Hermite-Kurven

"Approximieren Sie die Kurve durch eine Folge von n Linien, indem Sie $n-1$ Zwischenpunkte bestimmen."

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Hermite-Kurven

"Approximieren Sie die Kurve durch eine Folge von n Linien, indem Sie $n - 1$ *Zwischenpunkte bestimmen."*

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Also, wir zeichnen Linien. Wie viele Linien haben wir? und wie lauten die Werte von t für jede Linie?

```
double cx[4], cy[4];
  DPoint2D anf, end;
  double t;
  double ninv = 1.0 / n;
```

"Approximieren Sie die Kurve durch eine Folge von n Linien, indem Sie $n - 1$ *Zwischenpunkte bestimmen."*

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Also, wir zeichnen Linien. Wie viele Linien haben wir? und wie lauten die Werte von t für jede Linie?

```
double cx[4], cy[4];
  DPoint2D anf, end;
  double t;
  double ninv = 1.0 / n;
```

Erzeugen und mahlen wir Linien wie immer:

Hermite-Kurven

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

```
DPoint2D anf, end;
  double t;
  double ninv = 1.0 / n;
  end.x = cx[3];
  end.y = cy[3];
  for (i = 1; i <= n; i++)
  {
    t = ninv * i;
    anf = end;
    end.x = ((cx[0] * t + cx[1]) * t + cx[2]) * t + cx[3];
    end.y = ((cy[0] * t + cy[1]) * t + cy[2]) * t + cy[3];
    pic.drawLine(round(anf), round(end), 0, slow);
  }
  // FERTIG!
```

Für die **Hermite-Kurven**, wir haben es so gemacht:

$$Q(t) = t^3 \underbrace{(2P_1 - 2P_4 + R_1 + R_4)}_{c_0} + t^2 \underbrace{(-3P_1 + 3P_4 - 2R_1 - R_4)}_{c_1} + t \underbrace{(R_1)}_{c_2} +$$

$$Q(t) = c_0 t^3 + c_1 t^2 + c_2 t + c_3 t$$

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

c_i sind Vektoren, d.h., $c_i(x, y)$.

Für die **Hermite-Kurven**:

$$Q(t) = c_0 t^3 + c_1 t^2 + c_2 t + c_3 t$$
$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

c_i sind Vektoren, d.h., $c_i(x, y)$.

Bézier-Kurven und B-Splines sind auch Polynome, d.h. man kann sie so beschreiben

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Bézier-Kurven und B-Splines

dann sieht der Code genauso aus wie vorher.

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Bézier-Kurven und B-Splines

dann sieht der Code genauso aus wie vorher.

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

```
for (i = 1; i <= n; ++i){
  t = ninv * i;
  anf = end;
  end.x = ((cx[0] * t + cx[1]) * t + cx[2]) * t + cx[3];
  end.y = ((cy[0] * t + cy[1]) * t + cy[2]) * t + cy[3];
  pic.drawLine(round(anf), round(end));
  }
```

Die Frage ist denn natürlich: was sind die Koeffizienten c_i?

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Bézier-Kurven und B-Splines

Die Frage ist denn natürlich: was sind die Koeffizienten c_i?

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Die antwort ist: Hermite-Form

- **Bézier-Kurven** (9-6)

$$Q(t) = G_B M_B T = C_{Be} T$$

- **B-Splines** (9-16)

$$Q_i(t) = G_{BS_i} M_{BS} T_i = C_B T_i$$

Hier:

$$G = (P_{i-3}, P_{i-2}, P_{i-1}, P_i) \qquad \text{und} \qquad T = (t^3, t^2, t, 1)^T$$

und M sind die Basismatrizen.

Bézier-Kurven und B-Splines

Die Frage ist denn natürlich: was sind die Koeffizienten c_i?

$$Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

Die antwort ist: Hermite-Form

- **Bézier-Kurven** (9-6)

$$Q(t) = G_B M_B T = C_{Be} T$$

- **B-Splines** (9-16)

$$Q_i(t) = G_{BS_i} M_{BS} T_i = C_B T_i$$

Hier:

$$G = (P_{i-3}, P_{i-2}, P_{i-1}, P_i) \qquad \text{und} \qquad T = (t^3, t^2, t, 1)^T$$

und M sind die Basismatrizen.

Bézier-Kurven und B-Splines

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T$
- **B-Splines** $Q_i(t) = G_{BS_i} M_{BS} T_i = C_B T_i$

Hier:

$$G = (P_{i-3}, P_{i-2}, P_{i-1}, P_i) \qquad \text{und} \qquad T = (t^3, t^2, t, 1)^T \tag{2}$$

und M sind die Basismatrizen.

$$M_B = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 4 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad M_{BS} = \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T$
- **B-Splines** $Q_i(t) = G_{BS_i} M_{BS} T_i = C_B T_i$

Hier:

$$G = (P_{i-3}, P_{i-2}, P_{i-1}, P_i) \qquad \text{und} \qquad T = (t^3, t^2, t, 1)^T \tag{2}$$

und M sind die Basismatrizen.

$$M_B = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 4 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad M_{BS} = \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Ok, klar, jetzt nur noch den Code implementieren. (Werte von C rechnen)

Bézier-Kurven und B-Splines

Von Übungsblatt:

```
for (k = 3; k <= m; k += 3) {
    // Die Kurve besteht dann aus m/3 einzelnen Kurvenstücken.
}
```

Bézier-Kurven $Q(t) = G_B M_B T = C_{Be} T$

$$G = (P_{i-3}, P_{i-2}, P_{i-1}, P_i) \quad \text{und} \quad T = (t^3, t^2, t, 1)^T \tag{3}$$

und M sind die Basismatrizen.

$$M_B = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 4 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

```
for (k = 3; k <= m; k += 3) { //Bézier
    cx[0] = -p[k - 3].x + 3 * p[k - 2].x - 3 * p[k - 1].x + p[k].x;
    cx[1] = 3 * p[k - 3].x - 6 * p[k - 2].x + 3 * p[k - 1].x;
    cx[2] = -3 * p[k - 3].x + 3 * p[k - 2].x;
    cx[3] = p[k - 3].x;
}

end.x = cx[3];  end.y = cx[3];

for (i = 1; i <= n; ++i){
  t = ninv * i;      anf = end;
  end.x = ((cx[0] * t + cx[1]) * t + cx[2]) * t + cx[3];
  end.y = ((cy[0] * t + cy[1]) * t + cy[2]) * t + cy[3];
  pic.drawLine(round(anf), round(end));
}
```

Und gleich für $cy[i]$. Fertig! 🙂