Übungen - Bildgenierung

Übung 09.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

January 15, 2025

Table of Contents

Aufgabe: Zu jeweils 16 Punkten $p[i][j], ..., p[i+3][j+3]$ gehört ein Flächenstück bestehend aus **anzkurv** Kurvenstücken für jede der beiden Richtungen, wobei jedes Kurvenstück durch **anzlin** Linien approximiert wird.

Ihr habt in der Vorlesung gelernt:

Parametrisierte bikubische Fläche

$$Q(s,t) = T^T \cdot M^T \cdot \tilde{G} \cdot M \cdot S \qquad T = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}, \qquad S = \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix}$$

oder **koordinatenweise**:

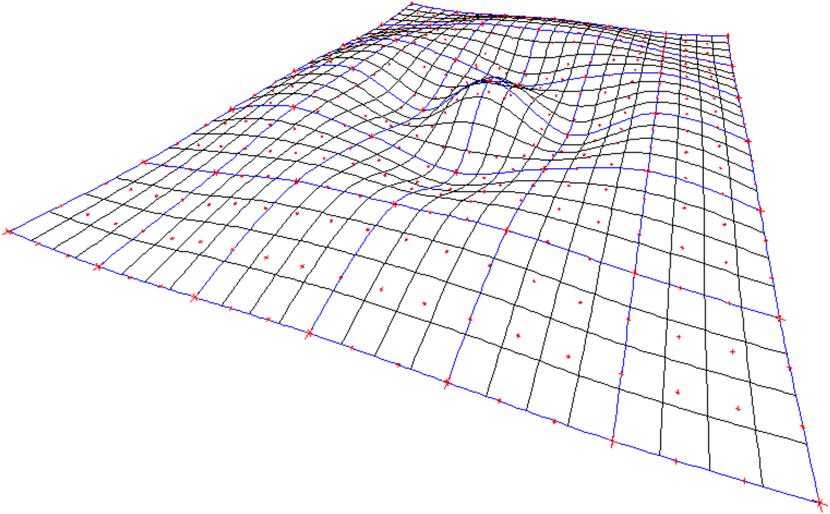
$$x(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_x \cdot M}_{C_1} \cdot S$$

$$y(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_y \cdot M}_{C_2} \cdot S$$

$$z(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_z \cdot M}_{C_3} \cdot S$$

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [?,?] \quad \textbf{und } t \in [?,?]$$



Bézier-Flächen

Zu jeweils 16 Punkten $p[i][j], \ldots, p[i+3][j+3]$ gehört ein Flächenstück bestehend aus **anzkurv** Kurvenstücken für jede der beiden Richtungen, wobei jedes Kurvenstück durch **anzlin** Linien approximiert wird.

Wir implementieren:

```
void berechneBezierFlaeche( vector<Kante>&  vk,
                            vector<vector<Vec3D> > &p,
                            int anzkurv = 5, int anzlin = 20 ){
  /* berechnet ein Netz aus Bezier-Flaechen, alle
  Kantenstuecke werden  dem Vektor vk hinzugefügt*/
}
```

Was ist **vk**?

Was ist **p?**

Wir brauchen...

- einen Zähler für jede Richtung (von $0, 1, 2...?$ Denkt über P nach)
- zwei Deltas für die Kurven und Linien

$$s = s + \Delta_{\text{kurv}}$$

$$t = t + \Delta_{\text{lin}}$$

- –Bezier-Basismatrix–
- Geometriematrix
- C_1, C_2, C_3. (Was sind die nochmal?)

Wir brauchen...

- einen Zähler für jede Richtung (von $0, 1, 2...?$ Denkt über P nach)
- zwei Deltas für die Kurven und Linien

$$s = s + \Delta_{\text{kurv}}$$

$$t = t + \Delta_{\text{lin}}$$

- –Bezier-Basismatrix–
- Geometriematrix
- C_1, C_2, C_3. (Was sind die nochmal?)

$$x(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_x \cdot M}_{C_1} \cdot S$$

$$y(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_y \cdot M}_{C_2} \cdot S$$

$$z(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_z \cdot M}_{C_3} \cdot S$$

Bézier-Flächen

Wir brauchen...

- einen Zähler für jede Richtung ☑️
- zwei Deltas für die Kurven und Linien ☑️

$$s = s + \Delta_{\text{kurv}}$$

$$t = t + \Delta_{\text{lin}}$$

- –Bezier-Basismatrix–
- Geometriematrix
- C_1, C_2, C_3.

```
void berechneBezierFlaeche( vector<Kante>&  vk,
                            vector<vector<Vec3D> > &p,
                            int anzkurv = 5, int anzlin = 20 ){
   /* berechnet ein Netz aus Bezier-Flaechen, alle
  Kantenstuecke werden  dem Vektor vk hinzugefügt*/
  int m = p.size() - 1;
  int n = p[0].size() - 1;
  double deltakurv = 1.0 / (anzkurv - 1);
  double deltalin = 1.0 / anzlin;
  Matrix4x4 C[3];
```

Wir brauchen...

- ...
- –Bezier-Basismatrix–
- Geometriematrix
- C_1, C_2, C_3.
- Schleifen über die Einzel-Flächen

```
void berechneBezierFlaeche( vector<Kante>&  vk,
                            vector<vector<Vec3D> > &p,
                            int anzkurv = 5, int anzlin = 20 ){
   /* berechnet ein Netz aus Bezier-Flaechen, alle
  Kantenstuecke werden  dem Vektor vk hinzugefügt*/
  int m = p.size() - 1;
  int n = p[0].size() - 1;
  double deltakurv = 1.0 / (anzkurv - 1);
  double deltalin = 1.0 / anzlin;

  double mbel[4][4] = { { -1,  3, -3, 1 },
                        {  3, -6,  3, 0 },
                        { -3,  3,  0, 0 },
                        {  1,  0,  0, 0 } };
  Matrix4x4 MB(mbel);
```

Schleife:

```
void berechneBezierFlaeche( vector<Kante>&  vk,
                            vector<vector<Vec3D> > &p,
                            int anzkurv = 5, int anzlin = 20 ){

  int m = p.size() - 1;
  int n = p[0].size() - 1;

  double deltakurv = 1.0 / (anzkurv - 1);
  double deltalin = 1.0 / anzlin;

  double mbel[4][4] = {...};   Matrix4x4 MB(mbel);  Matrix4x4 C[3];

  // Schleifen über die Einzel-Flaechen
  for (k = 3; k <= m; k += 3)
    for (l = 3; l <= n; l += 3)
      /*--------------- berechne vorweg die Matrizen C[d] ------*/

      /*---------------  Linien für jeweils festes s -------------*/
      /*---------------  Linien für jeweils festes t -------------*/
```

Fragen?

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad C_3 = M^T \cdot \tilde{G}_z \cdot M$$

Berechne vorweg die **3** Matrizen $C[d]$.

```
...

  for (k = 3; k <= m; k += 3) // Schleifen über die Einzel-Flaechen
    for (l = 3; l <= n; l += 3){

  //------------- berechne vorweg die Matrizen C[d]-------------
      for (d = 0; d < 3; d++){   //für jede Matrix C haben wir...
        for (i = 0; i < 4; i++)   // 16 KontrolPunkte im G
          for (j = 0; j < 4; j++)
            G.el[i][j] = p[k - 3 + i][l - 3 + j].el[d];
        C[d] = MB * G * MB; // MB^T = MB
      }
  ...
```

$G_{x,z,y}$ ist eine 4×4-Matrix, deren Elemente die Kontrollpunkte sind.

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad C_3 = M^T \cdot \tilde{G}_z \cdot M$$

```
void berechneBezierFlaeche( vector<Kante>&  vk,
                            vector<vector<Vec3D> > &p,
                            int anzkurv = 5, int anzlin = 20 ){

  int m = p.size() - 1,         n = p[0].size() - 1;

  double deltakurv = 1.0 / (anzkurv - 1),   deltalin = 1.0 / anzlin;

  double mbel[4][4] = {...};   Matrix4x4 MB(mbel);  Matrix4x4 C[3];

  for (k = 3; k <= m; k += 3) // Schleifen über die Einzel-Flaechen
    for (l = 3; l <= n; l += 3){
      /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                  /*DONE*/


      /*--------------  Linien für jeweils festes s -------------*/
      /*--------------  Linien für jeweils festes t -------------*/
```

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) \quad \textbf{und } t \in [0,1).$$

s geht von 0 bis 1 in Schritten von *deltakruv*.
Für jede feste Kurve s haben wir eine Schleife über die Linien:

```
...
  for (k = 3; k <= m; k += 3) // Schleifen über die Einzel-Flaechen
    for (l = 3; l <= n; l += 3){
      /*--------------  Linien für jeweils festes s -------------*/
      for (s = 0; s < 1; s += deltakurv)
      {
        for (t = deltalin; t<=1; t += deltalin)
        {
...
```

Die erste Kante für festes s

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) \quad \textbf{und} \ t \in [0,1).$$

s geht von 0 bis 1 in Schritten von *deltakruv*.
Für jede feste Kurve *s* haben wir eine Schleife über die Linien:

```
...
  for (k = 3; k <= m; k += 3) // Schleifen über die Einzel-Flaechen
    for (l = 3; l <= n; l += 3){
      /*--------------  Linien für jeweils festes s -------------*/
      for (s = 0; s < 1; s += deltakurv)
      {
        for (t = deltalin; t<=1; t += deltalin)
        {
...
```

Die erste Kante für festes *s* ist etwas wie...

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) \quad \textbf{und } t \in [0,1).$$

s geht von 0 bis 1 in Schritten von *deltakruv*.
Für jede feste Kurve *s* haben wir eine Schleife über die Linien:

```
...
  for (k = 3; k <= m; k += 3) // Schleifen über die Einzel-Flaechen
    for (l = 3; l <= n; l += 3){
      /*--------------  Linien für jeweils festes s -------------*/
      for (s = 0; s < 1; s += deltakurv)
      {
        for (t = deltalin; t<=1; t += deltalin)
        {
...
```

Die erste Kante für festes *s* ist etwas wie...

```
anf = mult(s, C, 0);
   end = mult(s, C, deltalin);  //deltalin = t
   vk.push_back(Kante(anf, end, BLACK));
```

Dann: $z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) \quad$ **und** $t \in [0,1)$.

```
...
  for (k = 3; k <= m; k += 3) // Schleifen über die Einzel-Flaechen
    for (l = 3; l <= n; l += 3){
     /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                   /*DONE*/
      /*--------------  Linien für jeweils festes s -------------*/
       for (s = 0; s < 1; s += deltakurv)
      {
        end = mult(s, C, 0);
        for (t = deltalin; t<=1; t += deltalin)
        {
          anf = end;
          end = mult(s, C, t);
          vk.push_back(Kante(anf, end, BLACK));
        }
      }

      /*--------------  Linien für jeweils festes t -------------*/
                               /*Analog zu s*/
```

Und... die Function mult?

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S$$

Und... die Function mult?

```
Vec4D mult(double s, Matrix4x4 C[3], double t)
{
  // berechnet den Punkt fuer die Parameter s und t

  Vec4D ss(s * s * s, s * s, s , 1);
  Vec4D tt(t * t * t, t * t, t , 1);

  return Vec4D( skalarprod(ss, (C[0] * tt)),
                skalarprod(ss, (C[1] * tt)),
                skalarprod(ss, (C[2] * tt)),
                1 );
}
```

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) \quad \textbf{und } t \in [0,1).$$

```
// Schleifen ueber die Einzel-Flaechen
  for (k = 3; k <= m; k += 3)
    for (l = 3; l <= n; l += 3){
      // berechne vorweg die Matrizen C[i]
      for (d = 0; d < 3; d++){
        for (i = 0; i < 4; i++)
          for (j = 0; j < 4; j++)
            G.el[i][j] = p[k - 3 + i][l - 3 + j].el[d];
        C[d] = MB * G * MB; // MB^T = MB
      }
      // Linien fuer jeweils festes s  (Link nach Rechts)
      for (s = 0; s < 1; s += deltakurv){
        end = mult(s, C, 0);
        for (t = deltalin; t<=1; t += deltalin){
          anf = end;
          end = mult(s, C, t);

          vk.push_back(Kante(anf, end, BLACK));
        }
      }
      // Linien fuer jeweils festes t    -ANALOG- zu s
      end = mult(0, C, t);
      ...
```

Rotationskörper

Beim letzten mal und im Vorlesung haben wir gelernt:

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

Hier:

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

und M sind die Basismatrizen.

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

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Idee:

```
void berechneRotationsKoerper( vector<Kante>& vk, const
↪  vector<Vec3D> &p, int anzkurv = 5, int anzlinku = 20,
int anzkreis = 5, int anzlinkr = 20 ){
  int m = p.size() - 1;
  for (k = 3; k <= m; k += 3) {
   /*-------------- berechne vorweg die Matrizen C[d] ------*/

    /*-------------------- male Kurvenstücke --------------------
        -------------- 1. Alle Kante erzeugen -------------------
        -------------- 2. for(0 ... 2π) --------------------*/


     /*--------------------- male Kreise  ---------------------
       -------------- 1. Alle Kante erzeugen -------------------
       -------------- 2. for(0 ... 2π) --------------------*/
   }
```

Wie?: letzten mal haben wir es gemacht

Rotationskörper

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

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.

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

```
int m = p.size() - 1;
for (k = 3; k <= m; k += 3) {
    for (d = 0; d < 2; d++){ //für x und y
      c[0][d] = -p[k - 3].el[d] + 3 * p[k - 2].el[d] - 3 * p[k - 1].el[d]
        + p[k].el[d];
      c[1][d] = 3 * p[k - 3].el[d] - 6 * p[k - 2].el[d]
        + 3 * p[k - 1].el[d];
      c[2][d] = -3 * p[k - 3].el[d] + 3 * p[k - 2].el[d];
      c[3][d] = p[k - 3].el[d];
    }
```

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Idee:

```
void berechneRotationsKoerper( vector<Kante>& vk, const
↪  vector<Vec3D> &p, int anzkurv = 5, int anzlinku = 20,
int anzkreis = 5, int anzlinkr = 20 ){
  int m = p.size() - 1;
  for (k = 3; k <= m; k += 3) {
    /*-------------- berechne vorweg die Matrizen C[d] ------*/
                            /*DONE*/


    /*-------------------- male Kurvenstücke ----------------*/


    /*--------------------- male Kreise  -------------------*/


   }
```

Kurvenstücke malen

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Die erste stück der Kurve ist:

```
...

 for (k = 3; k <= m; k += 3){
/*-------------- berechne vorweg die Matrizen C[d] ------*/
                              /*DONE*/

   /*-------------------- male Kurvenstücke ----------------*/
   anf = //Q(0) = C * T(0)
   end = //Q(Δt) = CΔT
   /*--------------------- male Kreise  --------------------*/

  }
```

Kurvenstücke malen

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Die erste stück der Kurve ist:

```
...

 for (k = 3; k <= m; k += 3){
/*-------------- berechne vorweg die Matrizen C[d] ------*/
                              /*DONE*/

   /*------------------- male Kurvenstücke ----------------*/
   anf  = //Q(0) = C * T(0)
   end  = //Q(Δt) = CΔT
   /*-------------------- male Kreise  --------------------*/

  }
```

Wir wissen schon, dass das Produkt wie folgt hergestellt wird:

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

dann...

Kurvenstücke malen

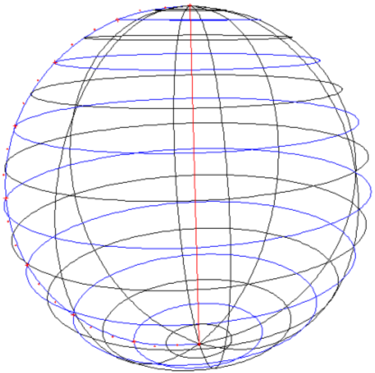
- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

```
...
  for (k = 3; k <= m; k += 3){
 /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                /*DONE*/


    /*-------------------- male Kurvenstücke ----------------*/
  anf = Vec4D(c[3][0], c[3][1], 0, 1);

  end = Vec4D(((c[0][0] * t + c[1][0]) * t + c[2][0]) * t + c[3][0],
↪  ((c[0][1] * t + c[1][1]) * t + c[2][1]) * t + c[3][1], 0, 1);

  vk.push_back(Kante(anf, end, BLUE));


   }
```

Kurvenstücke malen

OK, eine Kante... wir drehen unsere Kante von 0 auf 2π.

$$\phi = 0, \Delta_{\mathsf{kurv}}, ..., 2\Pi. \qquad \Delta_{\mathsf{kurv}} = \frac{2\Pi}{\mathsf{anzkurv}},$$



Kurvenstücke malen

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

$$\phi = 0, \Delta_{\mathsf{kurv}}, ..., 2\Pi. \qquad \Delta_{\mathsf{kurv}} = \frac{2\Pi}{\mathsf{anzkurv}},$$

```
...
  double deltakurv = 2.0 * M_PI / anzkurv;
  for (k = 3; k <= m; k += 3){
 /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                /*DONE*/

    /*------------------- male Kurvenstücke ---------------*/
    anf  = //Q(0) = C * T(0)
    end  = //Q(Δt) = CΔT
    vk.push_back(Kante(anf, end, BLUE));

    for (phi = deltakurv; phi <= 2*M_PI, phi += deltakurv){
        anf2 = ?
        end2 = ?
      }
   }
```

Kurvenstücke malen

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

```
...
  double deltakurv = 2.0 * M_PI / anzkurv;
  for (k = 3; k <= m; k += 3){
 /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                /*DONE*/
    /*-------------------- male Kurvenstücke ----------------*/
    anf = //Q(0) = C * T(0)
    end = //Q(Δt) = CΔT
    vk.push_back(Kante(anf, end, BLUE));

    for (phi = deltakurv; phi <= 2*M_PI, phi += deltakurv){
        anf2 = Vec4D(anf.el[0], cos(phi) * anf.el[1],
                     sin(phi) * anf.el[1], 1);
        end2 = Vec4D(end.el[0], cos(phi) * end.el[1],
                     sin(phi) * end.el[1], 1);
        vk.push_back(Kante(anf2, end2, BLACK));
      }
   }
```

Das ist für die erste geradenstück, wir müssen dasselbe für die anderen **anzlinku** geradenstücke tun.

Rotationskörper

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Für die andere geradenstücke:

```
...
  for (k = 3; k <= m; k += 3){
 /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                   /*DONE*/
    /*-------------------- male Kurvenstücke ----------------*/
    //-------------- Alle Kante erzeugen-----------------------
    end = C*T(0);
    for (i = 1, t = deltalinku; i <= anzlinku; i++, t+= deltalinku){
      anf = end;
      end = C*T(t);
      vk.push_back(Kante(anf, end, BLUE));
    //--------------for(0 ... 2π) {Kante drehen}
```

male Kurvenstücke: DONE!

Bisher:

```
...
  double deltakurv = 2.0 * M_PI / anzkurv;
  for (k = 3; k <= m; k += 3){
 /*-------------- berechne vorweg die Matrizen C[d] ------*/
                              /*DONE*/
   // male Kurvenstuecke
    end = Vec4D(c[3][0], c[3][1], 0, 1);
    for (i = 1, t = deltalinku; i <= anzlinku; i++, t+= deltalinku){
      anf = end;
      end = //C*T(t);
      vk.push_back(Kante(anf, end, BLUE));
      for (j = 1, phi = deltakurv; j < anzkurv; j++, phi += deltakurv)
↪   //drehen
      {
        anf2 = Vec4D(anf.el[0], cos(phi) * anf.el[1],
                     sin(phi) * anf.el[1], 1);
        end2 = Vec4D(end.el[0], cos(phi) * end.el[1],
                     sin(phi) * end.el[1], 1);
        vk.push_back(Kante(anf2, end2, BLACK));
      }
    }
   }
```

- **Bézier-Kurven** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Bisher

```
...
  for (k = 3; k <= m; k += 3){
 /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                  /*DONE*/

    /*-------------------- male Kurvenstücke -------------------
        -------------- 1. Alle Kante erzeugen ------------------
        -------------- 2. for(0 ... 2π) --------------------*/
                               /*DONE*/

    /*--------------------- male Kreise  ----------------------
       -------------- 1. Alle Kante erzeugen ------------------
       -------------- 2. for(0 ... 2π) --------------------*/
```

Kreise: Analog

$$Q(t) = G_B M_B T = C_{Be} T \qquad Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$$

```
...
  double deltakreis = 1.0 / (anzkreis - 1);
  for (k = 3; k <= m; k += 3){
   ...
       /*--------------------- male Kreise  ----------------------
      -------------- 1. Alle Kante erzeugen ------------------*/
    end = C*T(0);   //Q(t) = ((c0 t + c1)t + c2)t + c3
    for (i = 0, t = 0; i < anzkreis; i++, t+= deltakreis)
    {
      anf = C*T(t)
     /* -------------- 2. for(0 ... 2π) --------------------*/
       end2 = anf;
       for (phi = deltalinkr; phi <= 2*M_PI; phi += deltalinkr){
         anf2 = end2;
         end2 = Vec4D(anf.el[0], cos(phi) * anf.el[1],
                      sin(phi) * anf.el[1], 1);
         vk.push_back(Kante(anf2, end2, BLACK));
        }
    }
```

- $Q(t) = G_B M_B T = C_{Be} T \qquad Q(t) = ((c_0 t + c_1)t + c_2)t + c_3$

Am Ende sieht unser Code so aus::

```
...
  int m = p.size() - 1;
  for (k = 3; k <= m; k += 3){
    /*-------------berechne vorweg die Matrizen C[i]------------*/

    /*-------------------- male Kurvenstücke -------------------
        -------------- 1. Alle Kante erzeugen ------------------
        -------------- 2. for(0 ... 2π) + pushBack --------*/

    /*-------------------- male Kreise  ----------------------
       -------------- 1. Alle Kante erzeugen ------------------
      -------------- 2. for(0 ... 2π) + pushBack  --------*/
}
```

Bézier-Flächen mit OpenGL

Heute, one-by-one folgen

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.
2. Aktivieren Sie die Kontrollpunkte mittels **glEnable**.
3. Erzeugen Sie unter Verwendung des Befehls **glMapGrid2f** ein Mesh, das aus nMeshSize Partitionen in jeder Richtung besteht.
4. Zeichnen Sie die Bézier-Fläche mit **glEvalMesh2**.

Bézier-Flächen mit OpenGL

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.

Bézier-Flächen mit OpenGL

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.

”The **glMap2f** function defines a two-dimensional evaluator. It generates some values depending of the **target**.”

Bézier-Flächen mit OpenGL

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.

"The **glMap2f** function defines a two-dimensional evaluator. It generates some values depending of the **target**."

Das Target das wir brauchen ist:

- **GL_MAP2_VERTEX_3**: Each control point is three floating-point values representing x, y, and z.

Paremeter für glMAP2f

```
glMap2f( target,   // GL_MAP2_VERTEX_3
            t1, t2,    // 0, 1
            tstride,   //floats/doubles
            torder,    // 4
            s1, s2,    // 0, 1
            sstride,   // 3*4
            sorder,    // 4
            points )   // unsere Punkte
```

1. **tstride** The number of floats or doubles between the beginning of control point $R_{i,j}$ and the beginning of control point $R_{i+1,j}$.
2. **torder:** The dimension of the control point array in the t-axis. Must be positive.

sstride The number of floats or doubles between the beginning of control point $R_{i,j}$ and the beginning of control point $R_{i,j+1}$.

Bézier-Flächen mit OpenGL

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.
2. Aktivieren Sie die Kontrollpunkte mittels **glEnable**.

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.
2. Aktivieren Sie die Kontrollpunkte mittels **glEnable**.

```
glMap2f( target,   // GL_MAP2_VERTEX_3
            t1, t2,   // 0, 1
            tstride,  // floats/doubles = 3
            torder,   // 4
            s1, s2,   // 0, 1
            sstride,  // 3*4
            sorder,   // 4
            points )  // unsere Punkte

   glEnable( GL_MAP2_VERTEX_3 );
```

Bézier-Flächen mit OpenGL

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.
2. Aktivieren Sie die Kontrollpunkte mittels **glEnable**.
3. Erzeugen Sie unter Verwendung des Befehls **glMapGrid2f** ein Mesh, das aus nMeshSize Partitionen in jeder Richtung besteht.

Bézier-Flächen mit OpenGL

"Defines a one-dimensional mesh."

- Paremeter für glMapGrid2f

```
void zeichneBezierFlaeche( const vector<vector<Vec3D> >& p,
                           int nMeshSize = 10 )
{
...
    glMapGrid2f( nMeshSize,
                 t1, t2,      // 0, 1
                 nMeshSize,
                 s1, s2 );    // 0, 1
}
```

Bézier-Flächen mit OpenGL

1. Legen Sie mittels **glMap2f** und des Target-Parameters **GL_MAP2_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Flächenstücks fest.
2. Aktivieren Sie die Kontrollpunkte mittels **glEnable**.
3. Erzeugen Sie unter Verwendung des Befehls **glMapGrid2f** ein Mesh, das aus nMeshSize Partitionen in jeder Richtung besteht.
4. Zeichnen Sie die Bézier-Fläche mit **glEvalMesh2**.

"Computes a two-dimensional grid of points or lines."

- Paremeter für glEvalMesh2

```
glEvalMesh2( mode,      // GL_POINT oder GL_LINE
            i1, i2,   /* "The first integer value for grid domain
                                 variable i."*/
            j1, j2;)  /* "The first integer value for grid domain
                                 variable j."*/
```

d.h:

```
glEvalMesh2( GL_LINE
                 0, nMeshSize
                 0, nMeshSize);
```

Bézier-Flächen mit OpenGL

Bisher:

1. Legen Sie die Kontrollpunkte des aktuellen **Flächenstücks** fest.
2. Aktivieren Sie die Kontrollpunkte mittels **glEnable**.
3. Erzeugen Sie unter Verwendung des Befehls **glMapGrid2f** ein Mesh, das aus nMeshSize Partitionen in jeder Richtung besteht.
4. Zeichnen Sie die Bézier-Fläche mit **glEvalMesh2**.

Bézier-Flächen mit OpenGL

Bisher:

```
void zeichneBezierFlaeche( const vector<vector<Vec3D> >& p,
                           int nMeshSize = 10 ){

/*glMap2f( target, t1, t2, tstride, torder,
            s1, s2, sstride, sorder, points )*/
  glMap2f( GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4,
            0.0, 1.0, 3*4, 4, /*points*/ );

  glEnable( GL_MAP2_VERTEX_3 );

//glMapGrid2f( tn, t1, t2, sn, s1, s2);
  glMapGrid2f( nMeshSize, 0.0, 1.0, nMeshSize, 0.0, 1.0 );

//glEvalMesh2( mode, i1, i2, j1, j2 )
  glEvalMesh2( GL_LINE, 0, nMeshSize, 0, nMeshSize );
}
```

Ok, wie initialisieren wir die Punkte?. **Wie Fruher**

Bézier-Flächen mit OpenGL

Die Vorbereitung war wie folgt:

```
void berechneBezierFlaeche( vector<Kante>&  vk, vector<vector<Vec3D>
↪  > &p, int anzkurv = 5, int anzlin = 20 ){

  int m = p.size() - 1,
  int n = p[0].size() - 1;

  // Schleifen über die Einzel-Flaechen
  for (k = 3; k <= m; k += 3)
    for (l = 3; l <= n; l += 3)
```

Nehmen wir denn alles! ctrl+ C

Bézier-Flächen mit OpenGL

```
void zeichneBezierFlaeche( const vector<vector<Vec3D> >& p,
                           int nMeshSize = 10 ){

  int m = p.size() - 1;        int n = p[0].size() - 1;

  for ( int k = 3; k <= m; k += 3 )
    for ( int l = 3; l <= n; l += 3 ){

      glMap2f( GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4,
            0.0, 1.0, 3*4, 4, /*points*/ );

      glEnable( GL_MAP2_VERTEX_3 );

      glMapGrid2f( nMeshSize, 0.0, 1.0, nMeshSize, 0.0, 1.0 );

      glEvalMesh2( GL_LINE, 0, nMeshSize, 0, nMeshSize );
    }
```

Punkte an OpenGL übergeben.

Bézier-Flächen mit OpenGL

```
void zeichneBezierFlaeche( const vector<vector<Vec3D> >& p,
                           int nMeshSize = 10 ){

  int m = p.size() - 1;      int n = p[0].size() - 1;

  // Array der Kontrollpunkte fuer OpenGL vorbereiten
  GLfloat *apPoints = new GLfloat[3*4*4];

  for ( int k = 3; k <= m; k += 3 )
    for ( int l = 3; l <= n; l += 3 ){

      //Übergeben 16 Kontrollpunkte der aktuellen Fläche an OpenGL.
        for( int i = 0; i < 4; i++ )
            for( int j = 0; j < 4; j++ ){
                apPoints[3*(4*i+j)+0] = p[k-3+i][l-3+j].el[0];  //x
                apPoints[3*(4*i+j)+1] = p[k-3+i][l-3+j].el[1];  //y
                apPoints[3*(4*i+j)+2] = p[k-3+i][l-3+j].el[2];  //z
            }
    }
```

```
void zeichneBezierFlaeche( const vector<vector<Vec3D> >& p,
                            int nMeshSize = 10 ){

  GLfloat *apPoints = new GLfloat[3*4*4];

  for ( int k = 3; k <= m; k += 3 )
    for ( int l = 3; l <= n; l += 3 ){

      //Übergeben 16 Kontrollpunkte der aktuellen Fläche an OpenGL.

      ...

      glMap2f( GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4, 0.0, 1.0, 3*4, 4, apPoints
↪  );
      glEnable( GL_MAP2_VERTEX_3 );
      glMapGrid2f( nMeshSize, 0.0, 1.0, nMeshSize, 0.0, 1.0 );
      glEvalMesh2( GL_LINE, 0, nMeshSize, 0, nMeshSize );
    }
  delete[] apPoints;
}
```

Rotationskörper

Rotationskörper Versucht es zu Hause.