Übungen - Bildgenierung

Übung 09.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

January 18, 2023

Table of Contents

Bézier-Flächen

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.

Bézier-Flächen

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

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

Bézier-Flächen

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

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.

```
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*/
```

Bézier-Flächen

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.

```
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*/
```

Wir brauchen...

- einen Zähler für jede Richtung (von 0 bis ... ?)
- zwei Deltas für die Kurven und Linien
- –Bezier-Basismatrix–
- Geometriematrix
- C_1, C_2, C_3.

$$x(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_x \cdot M}_{C_1} \cdot S \qquad y(s,t) = T^T \cdot \underbrace{M^T \cdot \tilde{G}_y \cdot M}_{C_2} \cdot S$$

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

Bézier-Flächen

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.

Wir brauchen...

- einen Zähler für jede Richtung
- zwei Deltas für die Kurven und Linien
- –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];
```

Bézier-Flächen

Wir brauchen...

- einen Zähler für jede Richtung
- zwei Deltas für die Kurven und Linien
- –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);
```

Bézier-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,          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];

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

Bézier-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,          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];

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

Wir könnten die Matrizen C[d] vorweg rechnnen

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

Bézier-Flächen

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

```
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] ------*/
```

Und dann, multiplizieren mit S und T.

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] ------*/

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

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;

  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++){   //jede C
        for (i = 0; i < 4; i++)    // alle 16 KontrolPunkte
          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
      }
```

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){

      for (d = 0; d < 3; d++){ // berechne vorweg die Matrizen C[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
      }
```

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 -------------*/
```

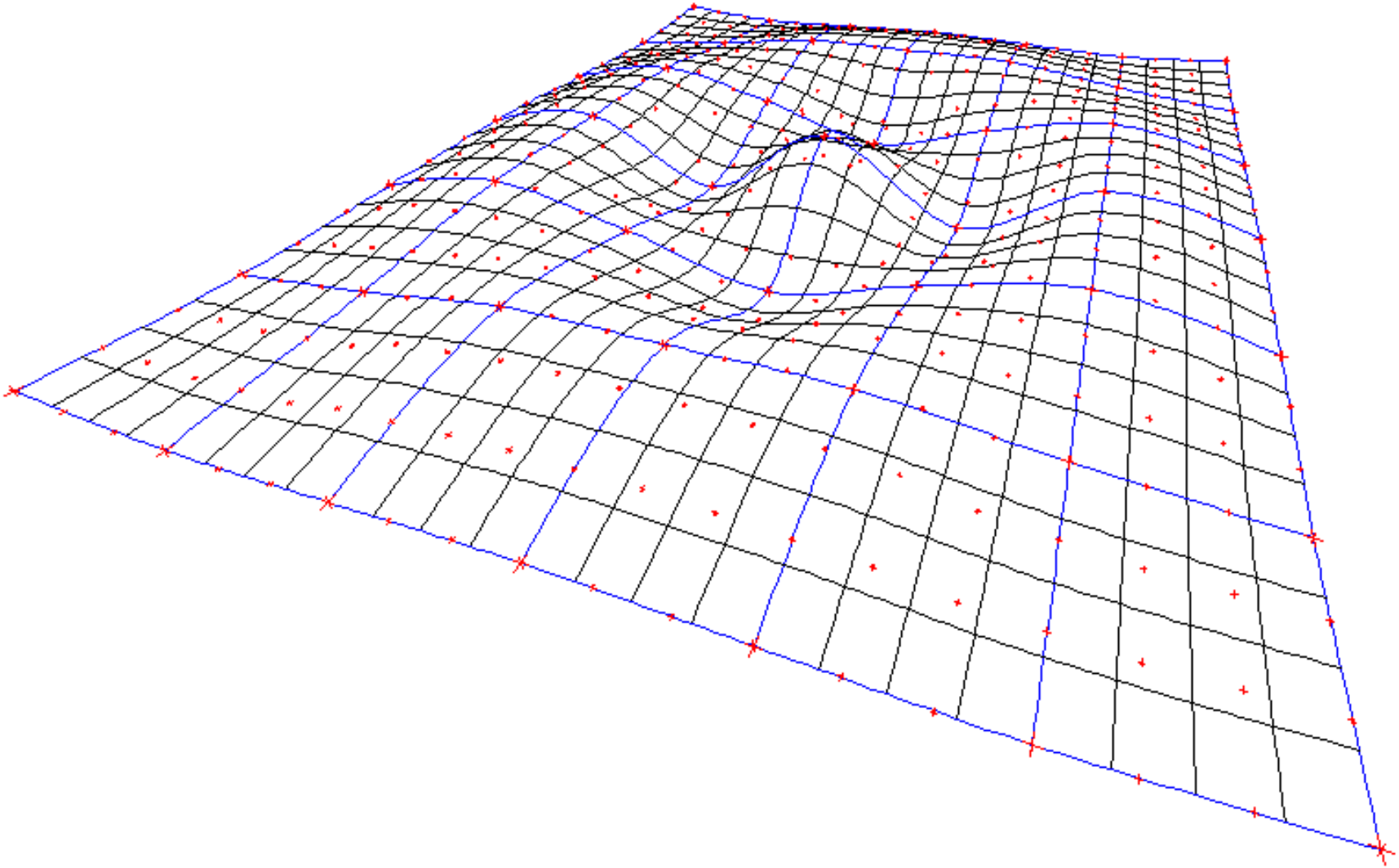
Bézier-Flächen

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

```
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

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



Bézier-Flächen

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

```
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 -------------*/
```

Wir gehen von 0 bis *anzkurv* mit schritten von *deltakruv*.

Bézier-Flächen

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

Wir gehen mit i von 0 bis *anzkurv* mit schritten von *deltakruv*.

```
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){
      /*--------------  Linien für jeweils festes s -------------*/
      for (i = 0, s = 0; i < anzkurv; i++, s += deltakurv)
      {
```

Bézier-Flächen

"Mit dem Befehl **vk.push_back(Kante(anf, end, BLACK));** können Sie dem Kanten-Vektor die einzelnen Kantenstücke hinzufügen, wobei anf und end vom Typ Vec4D sind."

Bézier-Flächen

"Mit dem Befehl **vk.push_back(Kante(anf, end, BLACK));** können Sie dem Kanten-Vektor die einzelnen Kantenstücke hinzufügen, wobei anf und end vom Typ Vec4D sind."

Wie können wir die Kanten erzeugen?

Bézier-Flächen

"Mit dem Befehl **vk.push_back(Kante(anf, end, BLACK));** können Sie dem Kanten-Vektor die einzelnen Kantenstücke hinzufügen, wobei anf und end vom Typ Vec4D sind."

Wie können wir die Kanten erzeugen?

Wir haben schleifen über die Flächen und Kurven.

Bézier-Flächen

"Mit dem Befehl **vk.push_back(Kante(anf, end, BLACK));** können Sie dem Kanten-Vektor die einzelnen Kantenstücke hinzufügen, wobei anf und end vom Typ Vec4D sind."

Wie können wir die Kanten erzeugen?

Wir haben for-schleifen über die Flächen und Kurven.

Wir brauchen noch eine über die Linien!

Bézier-Flächen

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

Wir brauchen noch eine über die Linien!

```
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){
      /*--------------  Linien für jeweils festes s -------------*/
      for (i = 0, s = 0; i < anzkurv; i++, s += deltakurv)
      {
        for (j = 1, t = deltalin; j <= anzlin; j++, t += deltalin)
        {
```

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) undt \in [0,1)$$

Wir brauchen noch eine über die Linien!

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

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

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) undt \in [0,1)$$

Wir brauchen noch eine über die Linien!

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

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

Natürlich mussen wir die Funktion mult implementieren..., aber wir gehen weiter.

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) undt \in [0,1)$$

Wir brauchen noch eine über die Linien!

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

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

Die zweite Kante für festes s ist...

```
anf = end;
end = mult(s, C, 2*deltalin);
vk.push_back(Kante(anf, end, BLACK));
```

und so weiter

Bézier-Flächen

$$z(s,t) = T^T \cdot C_3 \cdot S \qquad s \in [0,1) undt \in [0,1)$$

Wir brauchen noch eine über die Linien!

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

```
//------------------1--------------------
    anf = mult(s, C, 0);
    end = mult(s, C, deltalin);
    vk.push_back(Kante(anf, end, BLACK));
//------------------2--------------------
    anf = end;
    end = mult(s, C, 2*deltalin);
    vk.push_back(Kante(anf, end, BLACK));
//------------------3--------------------
    anf = end;
    end = mult(s, C, 3*deltalin);
    vk.push_back(Kante(anf, end, BLACK));
```

dann in unserer for-Schleife...

Bézier-Flächen

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

```
/*--------------  Linien für jeweils festes s -------------*/
 for (i = 0, s = 0; i < anzkurv; i++, s += deltakurv)
{
  end = mult(s, C, 0);
  for (j = 1, t = deltalin; j <= anzlin; j++, t += deltalin)
  {
    anf = end;
    end = mult(s, C, t);
    vk.push_back(Kante(anf, end, BLACK));
  }
}
```

Bézier-Flächen

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

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

  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 (i = 0, s = 0; i < anzkurv; i++, s += deltakurv)
      {
        end = mult(s, C, 0);
        for (j = 1, t = deltalin; j <= anzlin; j++, t += deltalin)
        {
          anf = end;
          end = mult(s, C, t);
          vk.push_back(Kante(anf, end, BLACK));
        }
      }
```

Bézier-Flächen

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

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

  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 -------------*/
                                /*DONE*/

      /*--------------  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 );
}
```

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 );
}
```

Schauen wir uns den Code an.

Rotationskörper

Beim letzten mal haben wir gelernt:

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

Hier:

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

und M sind die Basismatrizen.

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

```
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;
    }
```

Und es ist gleich für y, dann...

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.

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

```
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];
    }
```

Frage: Was war m?

Rotationskörper

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

Frage: Was war m?

```
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) {
    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];
    }
```

Rotationskörper

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

Now what?

```
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*/
   }
```

Rotationskörper

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

Now what?

```
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  --------------------*/

   }
```

ok, Wie?

Rotationskörper

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

Unsere erste Punkt ist

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    /*-------------------- male Kurvenstücke ----------------*/
    anf = //Q(0) = C * T(0)

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

  }
```

weil...

Rotationskörper

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

Unsere zweite Punkt ist

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){


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


  }
```

weil...

Rotationskörper

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

Unsere zweite Punkt ist

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){


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


   }
```

Beim letzten Mal haben wir erfahren, dass das Produkt wie folgt hergestellt wird:

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

dann...

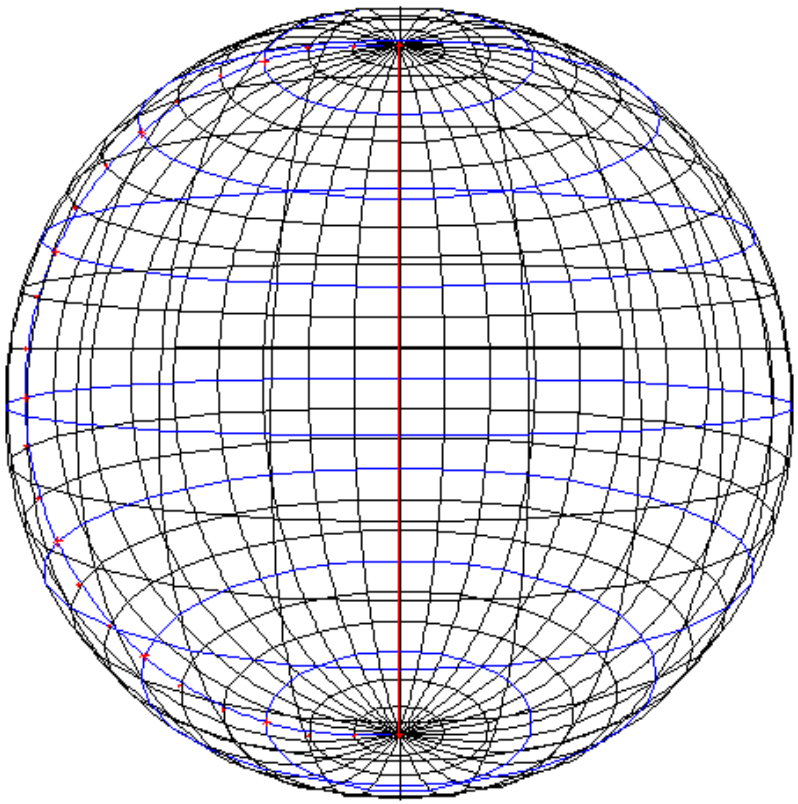
Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    /*-------------------- 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));


  }
```

Ok, jetzt müssen wir von 0 auf 2π gehen und die anderen Kanten erstellen.

Rotationskörper



Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){


    double deltakurv = 2.0 * M_PI / anzkurv;

    /*-------------------- 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));


  }
```

Ok, jetzt müssen wir von 0 auf 2π gehen und die anderen Kanten erstellen.

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    double deltakurv = 2.0 * M_PI / anzkurv;
    /*-------------------- male Kurvenstücke ----------------*/
    anf = //Q(0) = C * T(0)
    end = //Q(Δt) = CΔT
    vk.push_back(Kante(anf, end, BLUE));

    for (j = 1, phi = deltakurv; j < anzkurv; j++, phi += deltakurv){
        anf2 = ?
        end2 = ?
      }
  }
```

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    double deltakurv = 2.0 * M_PI / anzkurv;
    /*-------------------- male Kurvenstücke ----------------*/
    anf = //Q(0) = C * T(0)
    end = //Q(Δt) = CΔT
    vk.push_back(Kante(anf, end, BLUE));

    for (j = 1, phi = deltakurv; j < anzkurv; j++, 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));
      }
  }
```

Rotationskörper

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

Was wir bisher gemacht haben:

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    double deltakurv = 2.0 * M_PI / anzkurv;
    /*-------------------- male Kurvenstücke ----------------*/
    //-------------- Ein Kante erzeug----------------------
    anf = //Q(0) = C * T(0)
    end = //Q(Δt) = CΔT
    vk.push_back(Kante(anf, end, BLUE));

    //----------------- von 0 bis 2π--------------
    for (j = 1, phi = deltakurv; j < anzkurv; j++, phi += deltakurv){
        anf2 = anf
        end2 = end
        vk.push_back(Kante(anf2, end2, BLACK));
      }
  }
```

Rotationskörper

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

Was wir bisher gemacht haben:

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

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

    //--------------for(0 ... 2π) {...}
```

Wir müssen dasselbe für die anderen **anzlinku** geradenstücke tun.

Rotationskörper

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

Was wir bisher gemacht haben:

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    double deltalinku = 1.0 / anzlinku;

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

    //--------------for(0 ... 2π) {...}
```

Wir müssen dasselbe für die anderen **anzlinku** geradenstücke tun. Mit einer for Schleife

Rotationskörper

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

Was wir bisher gemacht haben:

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){
    double deltalinku = 1.0 / anzlinku;
    /*------------------- 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) {...}
```

Rotationskörper

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

Was wir bisher gemacht haben:

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

 double deltalinku = 1.0 / anzlinku;
/*-------------------- male Kurvenstücke ----------------*/
//-------------- Alle Kante erzeugen-----------------------
end = Vec4D(c[3][0], c[3][1], 0, 1);
for (i = 1, t = deltalinku; i <= anzlinku; i++, t+= deltalinku){
  anf = end;
  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));
//-------------for(0 ... 2π) {... SCHWARZ ...}
```

Rotationskörper

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

Was wir bisher gemacht haben:

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

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


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

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

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


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

Für die Kreise, ist das Konzept dasselbe. d.h.

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

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


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

Für die Kreise, ist das Konzept dasselbe. d.h.

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    double deltakreis = 1.0 / (anzkreis - 1);
   /*--------------------- male Kreise  ----------------------

      -------------- 1. Alle Kante erzeugen -----------------*/
   end = C*T(0);
   for (i = 0, t = 0; i < anzkreis; i++, t+= deltakreis)
   {
     anf = C*T(t)

    /* ------------- 2. for(0 ... 2π) -------------------*/
```

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){

    double deltakreis = 1.0 / (anzkreis - 1);
    /*-------------------- male Kreise  ---------------------

       -------------- 1. Alle Kante erzeugen ------------------*/
     end = Vec4D(c[3][0], c[3][1], 0, 1);
    for (i = 0, t = 0; i < anzkreis; i++, t+= deltakreis)
    {
      anf = 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);


    /*-------------- 2. for(0 ... 2π) --------------------*/
```

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){
   double deltakreis = 1.0 / (anzkreis - 1);
    /*---------------------- male Kreise  ----------------------

       -------------- 1. Alle Kante erzeugen -----------------*/
    end = C*T(0);
    for (i = 0, t = 0; i < anzkreis; i++, t+= deltakreis)
    {
      anf = C*T(t)

     /* ------------- 2. for(0 ... 2π) -------------------*/
```

Rotationskörper

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

```
void berechneRotationsKoerper( vector<Kante>& vk, const vector<Vec3D> &p,
                               int anzkurv = 5, int anzlinku = 20,
                               int anzkreis = 5, int anzlinkr = 20 ){
  double deltakreis = 1.0 / (anzkreis - 1);
  /*-------------------- male Kreise  ---------------------
     -------------- 1. Alle Kante erzeugen -----------------*/
  end = C*T(0);
  for (i = 0, t = 0; i < anzkreis; i++, t+= deltakreis){
    anf = C*T(t)
    /* -------------- 2. for(0 ... 2π) -------------------*/
    end2 = anf;
    for (j = 1, phi = deltalinkr; j <= anzlinkr; j++, 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));
    }
  }
```

Rotationskörper

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

Am Ende sieht unser Code so aus::

```
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[i]------------*/

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

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

Rotationskörper

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

Am Ende sieht unser Code so aus:

```
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[i]------------*/

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

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

Schauen wir uns den Code an.

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.

Bézier-Flächen mit OpenGL

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
```

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**.

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**.

```
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

```
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**.

Bézier-Flächen mit OpenGL

"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."*/
```

Bézier-Flächen mit OpenGL

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

- Paremeter für glEvalMesh2

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

Bézier-Flächen mit OpenGL

Bisher:

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 );
}
```

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 für des **aktuellen Flachenstucks:**

```
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?. **recap**

Bézier-Flächen mit OpenGL

Die Vorbereitung für Aufgabe 32 war wie folgt:

```
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;

  // 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 );
    }
```

Was denn mit die Punkte?

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 ){
        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 );
    }
```

Bézier-Flächen mit OpenGL

Wir mussen Bezierkurve vorbereiten, indem **alle 16 Kontrollpunkte** des aktuellen Surface-Gebiets an OpenGL uebergeben werden

```
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 ){

    //--------------> OpenGL calls <----------
    }
```

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 ){

      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
        }
    //--------------> OpenGL calls <----------
    }
```

Bézier-Flächen mit OpenGL

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

  // 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 ){

      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
        }
    //--------------> OpenGL calls <----------
    }
```

Schauen wir uns den Code an.

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten )
{
    }
```

Schauen wir uns den Rahmenprogram an.

Rotationskörper mit OpenGL

Wie können wir anfangen? Wir haben p nochmal

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten )
{
    }
```

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten ){
  int m= p.size() - 1;

}
```

Wie viele Kontrollpunkte brauchen wir ? ?

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten ){
  int m= p.size() - 1;


   // Array der Kontrollpunkte für OpenGL vorbereiten
  GLfloat *apPoints = new GLfloat[12];
}
```

Dann, mussen wir irgendwie die **anzkurv** Kurven malen, d.h....

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten ){
   int m= p.size() - 1;

  // Array der Kontrollpunkte für OpenGL vorbereiten
  GLfloat *apPoints = new GLfloat[12];
  // Male die Kurven
  for ( int c = 0; c < daten.anzkurv; c++ )
  {...}
}
```

Wir sollen Bezierkurve vorbereiten, indem alle 4 Kontrollpunkte des aktuellen Kurvenabschnitts an OpenGL übergeben werden.

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten ){
  int m= p.size() - 1;


   // Array der Kontrollpunkte für OpenGL vorbereiten
  GLfloat *apPoints = new GLfloat[12];
  // Male die Kurven
  for ( int c = 0; c < daten.anzkurv; c++ ){
    for ( int k = 3; k <= m; k += 3 )
    {}
}
```

Wir sollen Bezierkurve vorbereiten, indem alle 4 Kontrollpunkte des aktuellen Kurvenabschnitts an OpenGL übergeben werden.

Was sind die Werte?

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten ){
  int m= p.size() - 1;

   // Array der Kontrollpunkte für OpenGL vorbereiten
  GLfloat *apPoints = new GLfloat[12];

  for ( int c = 0; c < daten.anzkurv; c++ ){
    double phi = 2.0 * M_PI * ((double)c) / daten.anzkurv;
    for ( int k = 3; k <= m; k += 3 ){
      for( int i = 0; i < 4; i++ ){
        apPoints[3*i+0] = p[k-3+i].el[0];
        apPoints[3*i+1] = cos(phi) * p[k-3+i].el[1];   //0 ≤ φ ≤ pi
        apPoints[3*i+2] = sin(phi) * p[k-3+i].el[1];
      }
    }
}
```

Mit openGL, haben wir **glMap2f** für die flächen benutz...

Rotationskörper mit OpenGL

```
void zeichneRotationskoerper( const vector<Vec3D>& p,
                              RotkDaten daten ){
  int m= p.size() - 1;

   // Array der Kontrollpunkte für OpenGL vorbereiten
  GLfloat *apPoints = new GLfloat[12];

  for ( int k = 3; k <= m; k += 3 ){
    for( int i = 0; i < 4; i++ ){
        apPoints[3*i+0] = p[k-3+i].el[0];
        apPoints[3*i+1] = cos(phi) * p[k-3+i].el[1];
        apPoints[3*i+2] = sin(phi) * p[k-3+i].el[1];
      }
    }
}
```

Wir haben **glMap2f** für die flächen benutz...

Dieses mal, brauchen wir **glMap1f**.

Rotationskörper mit OpenGL

Wir haben **glMap2f** für die flächen benutz...

- "The glMap2f function defines a two-dimensional evaluator."

Dieses mal, brauchen wir **glMap1f**...

- "The glMap1f function defines a one-dimensional evaluator."

Rotationskörper mit OpenGL

So geht es:

1. Legen wir mittels **glMap1f** und des Target-Parameters **GL_MAP1_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Kurvenstücks fest.

Rotationskörper mit OpenGL

So geht es:

1. Legen wir mittels **glMap1f** und des Target-Parameters **GL_MAP1_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Kurvenstücks fest.
2. Aktivieren wir die Kontrollpunkte mittels **glEnable**.
3. Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl **glBegin**.

Rotationskörper mit OpenGL

So geht es:

1. Legen wir mittels **glMap1f** und des Target-Parameters **GL_MAP1_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Kurvenstücks fest.
2. Aktivieren wir die Kontrollpunkte mittels **glEnable**.
3. Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl **glBegin**.
4. Werten Wir die Kurve mittels **glEvalCoord1f** an Zwischenpunkten aus, so dass pro Kurvenstück **anzlinku** Linien entstehen.

Rotationskörper mit OpenGL

So geht es:

1. Legen wir mittels **glMap1f** und des Target-Parameters **GL_MAP1_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Kurvenstücks fest.
2. Aktivieren wir die Kontrollpunkte mittels **glEnable**.
3. Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl **glBegin**.
4. Werten Wir die Kurve mittels **glEvalCoord1f** an Zwischenpunkten aus, so dass pro Kurvenstück **anzlinku** Linien entstehen.
5. Beenden wir das Zeichnen mit **glEnd**.

Rotationskörper mit OpenGL

Legen wir mittels **glMap1f** und des Target-Parameters **GL_MAP1_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Kurvenstücks fest.Aktivieren wir die Kontrollpunkte mittels **glEnable**.

Dann, es ist einfach

```
glMap1f( GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, apPoints );
glEnable( GL_MAP1_VERTEX_3 );
```

Rotationskörper mit OpenGL

1. Legen wir mittels **glMap1f** und des Target-Parameters **GL_MAP1_VERTEX_3**, welcher dabei anzugeben ist, die Kontrollpunkte des aktuellen Kurvenstücks fest.
2. Aktivieren wir die Kontrollpunkte mittels **glEnable**.
3. Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl **glBegin**.
4. Werten Wir die Kurve mittels **glEvalCoord1f** an Zwischenpunkten aus, so dass pro Kurvenstück **anzlinku** Linien entstehen.
5. Beenden wir das Zeichnen mit **glEnd**.

Rotationskörper mit OpenGL

1. **Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl glBegin.**
2. Werten Wir die Kurve mittels **glEvalCoord1f** an Zwischenpunkten aus, so dass pro Kurvenstück **anzlinku** Linien entstehen.
3. Beenden wir das Zeichnen mit **glEnd**.

```
glBegin( GL_LINE_STRIP );
```

Rotationskörper mit OpenGL

1. Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl **glBegin**.
2. **Werten Wir die Kurve mittels glEvalCoord1f an Zwischenpunkten aus, so dass pro Kurvenstück anzlinku Linien entstehen.**
3. Beenden wir das Zeichnen mit **glEnd**.

```
glBegin( GL_LINE_STRIP );
for ( int i = 0; i <= daten.anzlinku; i++ )
```

Rotationskörper mit OpenGL

1. Teilen Wir OpenGL mit, dass durch Linien verbundene Punkte gezeichnet werden sollen. Dies erfolgt mit dem Befehl **glBegin**.
2. Werten Wir die Kurve mittels **glEvalCoord1f** an Zwischenpunkten aus, so dass pro Kurvenstück **anzlinku** Linien entstehen.
3. **Beenden wir das Zeichnen mit glEnd.**

```
// auswerten an Zwischenpunkten und zeichnen
glBegin( GL_LINE_STRIP );
for ( int i = 0; i <= daten.anzlinku; i++ )
glEnd();
```

Rotationskörper mit OpenGL

```
for ( int c = 0; c < daten.anzkurv; c++ )

    double phi = 2.0 * M_PI * ((double)c) / daten.anzkurv;

    // Male die Kurvenabschnitte
    for ( int k = 3; k <= m; k += 3 ){
      // Bezierkurve vorbereiten, indem alle 4 Kontrollpunkte des
      // aktuellen Kurvenabschnitts an OpenGL uebergeben werden
      for( int i = 0; i < 4; i++ )
      { apPoints[ ... ] = ... }

      glMap1f( GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, apPoints );
      glEnable( GL_MAP1_VERTEX_3 );

      // auswerten an Zwischenpunkten und zeichnen
      glBegin( GL_LINE_STRIP );
        for ( int i = 0; i <= daten.anzlinku; i++ )
          glEvalCoord1f( ((GLfloat)i)/daten.anzlinku );
      glEnd();
    }
```

Rotationskörper mit OpenGL

```
// Male die Kurvenabschnitte
for ( int c = 0; c < daten.anzkurv; c++ ){}

// Male die Kreise
```

Rotationskörper mit OpenGL

```
// Male die Kurvenabschnitte
for ( int c = 0; c < daten.anzkurv; c++ ){}

// Male die Kreise
```

Wie haben wir es gemacht ohne Opengl?

Rotationskörper mit OpenGL

```
// Male die Kurvenabschnitte
for ( int c = 0; c < daten.anzkurv; c++ ){}

// Male die Kreise
```

Wie haben wir es gemacht ohne Opengl?

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

```
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*/
   }
```

Also, nur zu erinnerung:

Rotationskörper mit OpenGL

```
// Male die Kreise
for (k = 3; k <= m; k += 3) {
    /*-------------- berechne vorweg die Matrizen C[d] ------*/
    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** $Q(t) = G_B M_B T = C_{Be} T \qquad T = (t^3, t^2, t, 1)^T$

Rotationskörper mit OpenGL

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

Wie viele Kreisen mahlen wir?

Rotationskörper mit OpenGL

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

```
// Male die Kreise
for (k = 3; k <= m; k += 3) {
    /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                    /*DONE*/
  for ( int i = 0; i < daten.anzkreis; i++ ){
    double t = i * 1.0 / ((double)daten.anzkreis-1);
    Vec4D anf  = C*T(t)
}
```

Jeder Kreis wird durch anzlink Geradenstücke angenähert.

Rotationskörper mit OpenGL

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

```
// Male die Kreise
for (k = 3; k <= m; k += 3) {
    /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                    /*DONE*/
  for ( int i = 0; i < daten.anzkreis; i++ ){
    double t = i * 1.0 / ((double)daten.anzkreis-1);
    Vec4D anf = C*T(t)
}
```

Jeder Kreis wird durch anzlink Geradenstücke angenähert. Nochmal verwenden wir

- **glBegin**(GL_LINE_STRIP)
- **glVertex3f**(x,y,z)
- **glEnd**

Rotationskörper mit OpenGL

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

```
// Male die Kreise
for (k = 3; k <= m; k += 3) {
    /*-------------- berechne vorweg die Matrizen C[d] ------*/
                                    /*DONE*/
  for ( int i = 0; i < daten.anzkreis; i++ ){
    double t = i * 1.0 / ((double)daten.anzkreis-1);
    Vec4D anf = C*T(t)

    glBegin( GL_LINE_STRIP );
    for ( int j = 0; j <= daten.anzlinkr; j++ ){
      double phi = j * 2.0 * M_PI / ((double)daten.anzlinkr);
      glVertex3f( anf.el[0], cos(phi) * anf.el[1], sin(phi) * anf.el[1] );
    }
    glEnd();
}
```

Jeder Kreis wird durch anzlink Geradenstücke angenähert.

Rotationskörper mit OpenGL

run!