OpenGL

Evaluators & NURBS

Sarah Wenzl

16.12.2009

Evaluators - Einführung

Was sind Evaluators?

- zugehörige Bibliothek gl (standard Graphic Library)

  #include <gl.h>

- Um sehr glatte (feine) Kurven oder Oberflächen zu erhalten, müsste man sehr viele Polygone verwenden → rechenaufwendig!

- Evaluators ermöglichen es präzise Polynome, Splines (mathematische Funktionen) oder Oberflächen zu beschreiben (mathematisch berechenbar)

- Mathematische Basis: **Bézierkurven/ Bézierflächen**

Evaluators - Einführung

- Kurven und Oberflächen können mathematisch definiert werden mit einer kleinen Anzahl an Parametern (=**Kontrollpunkte**)

- Aus Effizienzgründen max. 16 Kontrollpunkte → Weniger Speicherplatz notwendig mit 16 Kontrollpunkten

  Anmerkung: max. 16 Kontrollpunkte zu speichern erfordert bei weitem nicht so viel Speicher als Vektorinformationen zu jedem Vertex

- Kurven benutzen 1-D Evaluators, Flächen 2-D Evaluators

Evaluators - Bézierkurve-/Fläche

Bild links: Parameter- Wertebereich von u: 0.0 <=u <= 1.0

Bild rechts: Parameter- Wertebereich von u und v: 0.0 <=u <= 1.0, 0.0 <=v <= 1.0

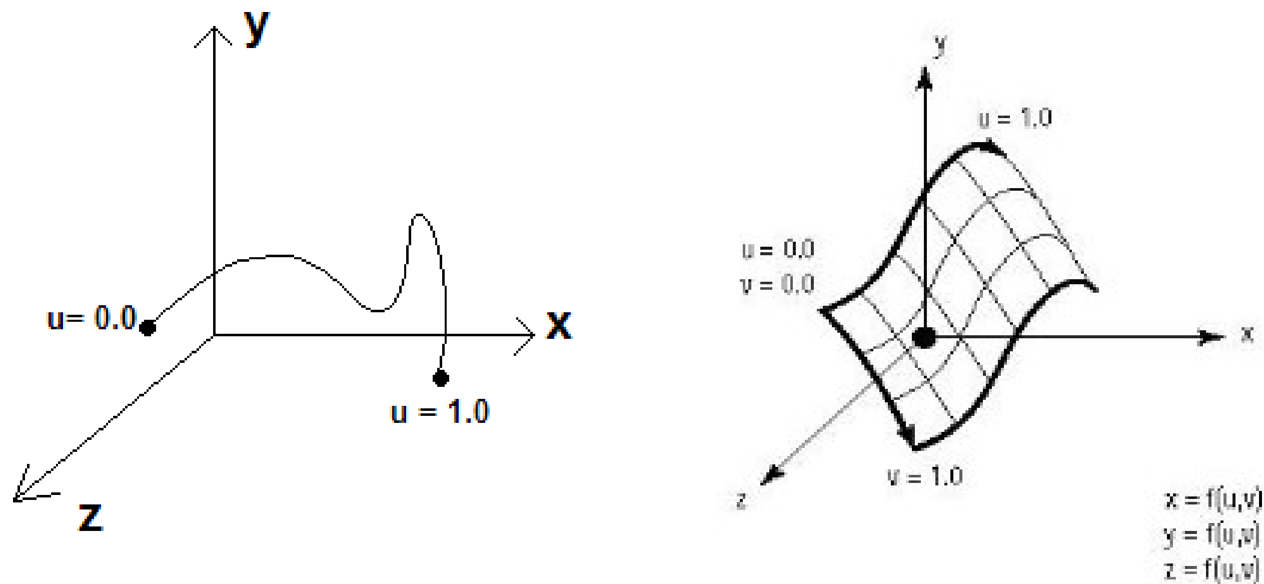


Figure 17-1 Parametric representations of curves and surfaces

Evaluators - Bézierkurve-/Fläche

Bézierkurve-/Fläche

Eine Bézierkurve ist eine Vektorfunktion mit nur einer Variablen (u).

- eine **Kurve** hat einen Startpunkt, eine Länge und einen Endpunkt

$C(u) = [X(u)\ Y(u)\ Z(u)]$

Eine **Fläche** ist eine Vektorfunktion bestehend aus zwei Variablen (u,v)

$S(u,v) = [X(u,v)\ Y(u,v)\ Z(u,v)]$

- u,v erstellen Matrix

Formeln C(u) und S(u,v) berechnen einen Kontrollpunkt auf einer Kurve oder Fläche.

Für die Definition einer Bézierkurve (eindimensional) benötigt man das

Bernstein-Polynom (Diese bilden eine Basis des Vektorraums der Polynome)

Evaluators – Control Points

Kontrollpunkte (Control Points)

- Eine Kurve wird durch eine Anzahl an Kontrollpunkten repräsentiert, welche die Form der Kurve beeinflussen
- In der Bézierkurve sind der erste und letzte Kontrollpunkt Teil der Kurve, die anderen Punkte arbeiten als “Magneten”, um die Kurve “zu sich heranzuziehen”
- wir beschäftigen uns mit quadratische (Parabel) und kubische Kurven (Polynom Dritten Grades)
- *Eine kubische Funktion hat in |R mindestens eine Nullstelle und maximal drei Nullstellen*

$$y = f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$$

Evaluators – Control Points

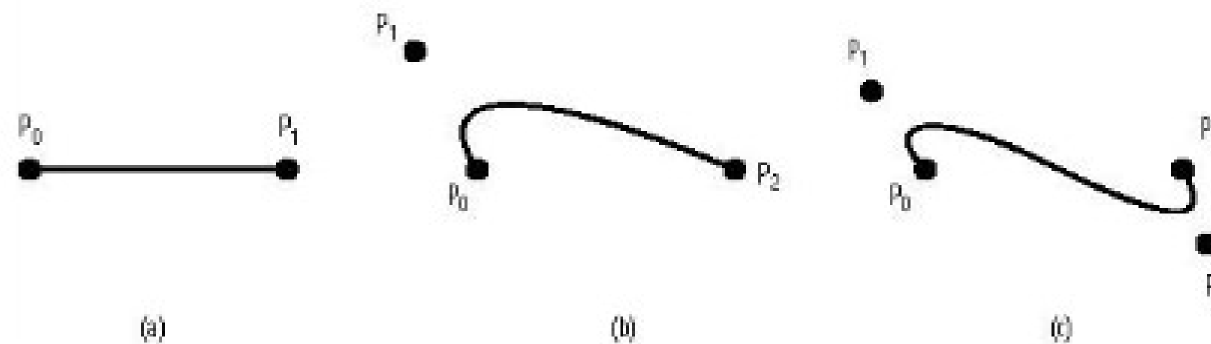
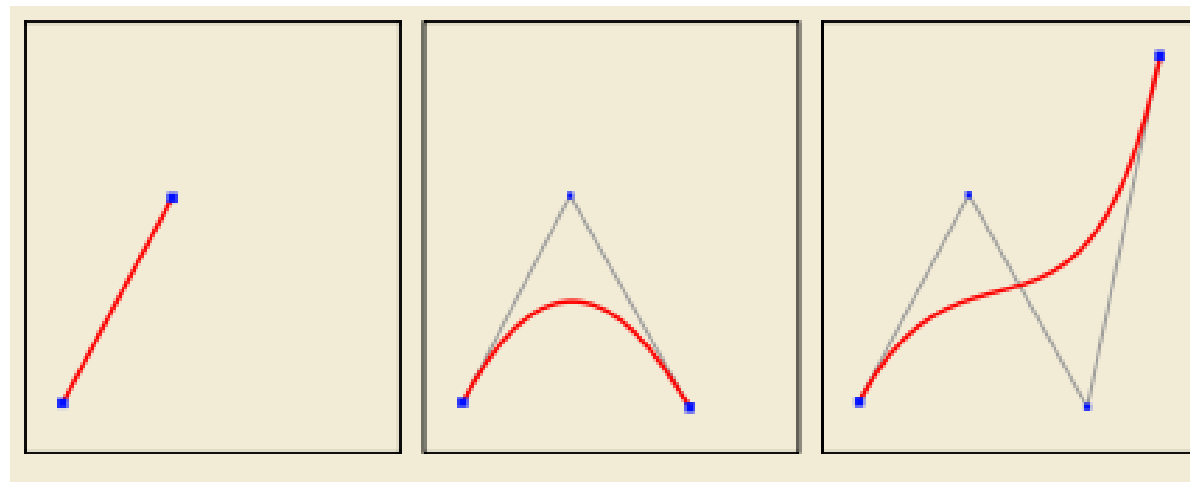


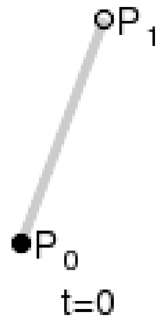
Figure 17-2 How control points affect curve shape



Evaluators - Bézierkurven

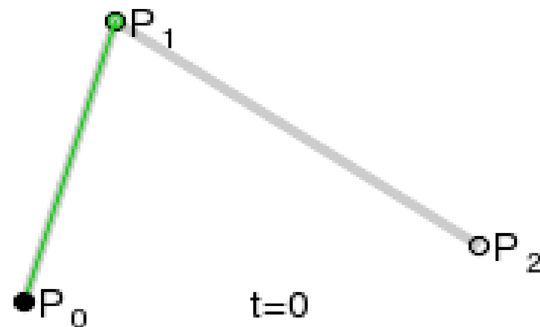
Lineare Bézierkurven (n=1)

Anmerkung: t entspricht Parameter u



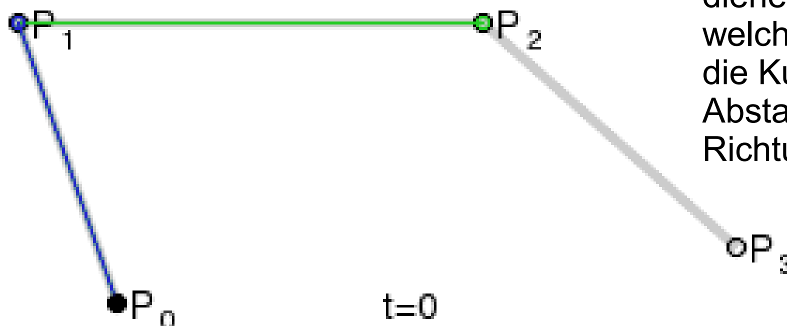
Zwei Kontrollpunkte P0 und P1 bestimmen eine lineare Bézierkurve, die einer Geraden zwischen diesen beiden Punkten entspricht.

Quadratische Bézierkurven (n=2)



Eine quadratische Bézierkurve ist der Pfad, der durch die Funktion C(t) für die Punkte P0, P1 und P2 verfolgt wird:

Kubische Bézierkurven (n=3)



Vier Punkte bestimmen eine kubische Bézierkurve. P1 und P2 dienen nur der Richtung, wobei P1 die Richtung bestimmt, in welche die Kurve in P0 geht. P2 legt die Richtung fest, aus welcher die Kurve zu P3 geht. Der Abstand zwischen P0 und P1 und der Abstand von P2 und P3 bestimmen, „wie weit" sich die Kurve in Richtung der Kontrollpunkte P1 und P2 bewegt, bevor sie in Richtung P3 läuft.

Evaluators - Functions

Evaluatorfunktionen:

- OpenGl beinhaltet verschiedene Funktionen um das Zeichnen von Bézierkurven und Flächen zu vereinfachen
- Durch Aufruf der Evaluation Funktion, generiert OpenGL die Punkte, die die Kurve oder Fläche aufstellen

Evaluators – Beispiel Bézierkurve

Beispiel einer kubischen Bézierkurve basierend auf 4 Kontrollpunkten

1. Als erstes müssen die Kontrollpunkte für die Kurve **global** definiert werden

- Jeder Eckpunkt besteht aus 3 floating- point Werten (x,y,z) → Value 3

```
GLint nNumPoints = 4;                    //Anzahl der Kontrollpunkte
GLfloat ctrlPoints[4][3] =               //Parameter u, x,y,z
                          {{ -4.0, -4.0, 0.0},         //Endpunkt
                           { -2.0, 4.0, 0.0},          //Kontrollpunkt 1
                           { 2.0, -4.0, 0.0},          //Kontrollpunkt 2
                           { 4.0, 4.0, 0.0}}           //Endpunkt
```

Evaluators – Beispiel Bézierkurve

2. Rendering Code für die Umsetzung als 3-D Darstellung

- **Aufruf in init()**
- glMap1f für die Abbildung der Kurve
- Beziérkurve- Erzeugungsmodus, wird über eine GL-Konstante definiert

- Zwei GL-MAP Konstanten prägnant:

  GL_MAP1 = 1D, Erzeugung Bézierkurven

  GL_MAP2 = 2D, z.B. für Oberflächenerzeugung einer NURBS

void ***glMap1****{fd}(GLenum* ***target****,* ***TYPEu1****,* ***TYPE****u2,*

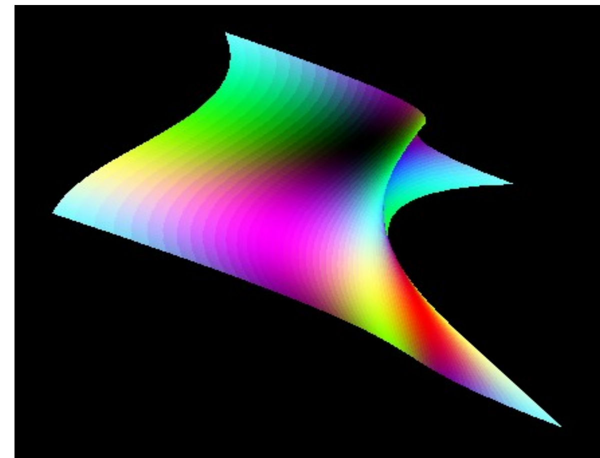
GLint ***stride****, GLint* ***order****,* ***const TYPE*points****);*

Evaluators – Beispiel Bézierkurve

```
glClear(GL_COLOR_BUFFER_BIT);  //zuerst Bildschirm löschen

glMap1f(                   //Für Abbildung des Objekts (Kurve)

GL_MAP1_VERTEX_3,          //über den Erzeugungsmodus wird angezeigt, dass eine Kurve
                           mithilfe von 3D-Eckpunkten(Vertices) gebildet werden soll,
                           → Bézier-Eckpunktefeld wird berechnet (approximiert)

0.0,                       // Anfangswert von Parameter u, Startwert der Kurve

1.0,         // Endwert von Parameter u. Bestimmt Feinheit der Kurve, Anzahl der
             Punkte die er beim Approximieren erzeugt - Anzahl kann selbst bestimmt
             werden für Feinheit der Kurve


3                          // Distanz zwischen den Kontrollpunkten

nNumPoints,                // Anzahl der Kontrollpunkte



&ctrlpoints[0][0]);        // Array der Kontrollpunkte, Zeiger auf Puffer welcher
                               Kontrollpunkte die die Kurve definieren beinhaltet
```

Evaluators - Types of Control Points for glMap1*()

Parameter	Meaning
GL_MAP1_VERTEX_3	x, y, z vertex coordinates
GL_MAP1_VERTEX_4	x, y, z, w vertex coordinates
GL_MAP1_INDEX	color index
GL_MAP1_COLOR_4	R, G, B, A
GL_MAP1_NORMAL	normal coordinates
GL_MAP1_TEXTURE_COORD_1	s texture coordinates - 1D
GL_MAP1_TEXTURE_COORD_2	s, t texture coordinates - 2D
GL_MAP1_TEXTURE_COORD_3	s, t, r texture coordinates - 3D
GL_MAP1_TEXTURE_COORD_4	s, t, r, q texture coordinates - 4D



* oder glMap2()

Evaluators – Beispiel Bézierkurve

3. Ausführen des Evaluators um Punkte entlang der Kurve zu produzieren

- **Aufruf in init()**

```
glEnable(GL_MAP1_VERTEX_3);
```

GlEnable() ermöglicht Ausführung des eindimensionalen Evaluators für dreidimensionale Eckpunkte

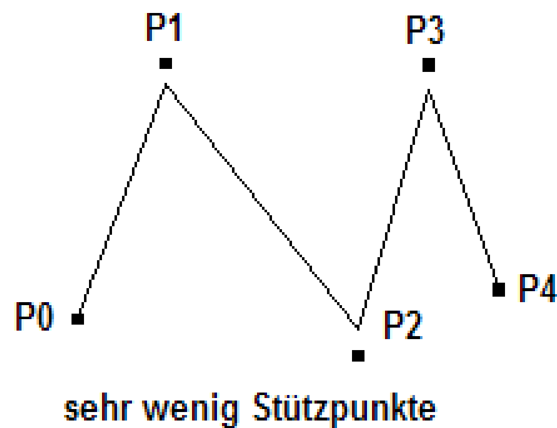
4. Kurve Zeichnen in der Routine display() zwischen glBegin() und glEnd()

- **Aufruf in display()**

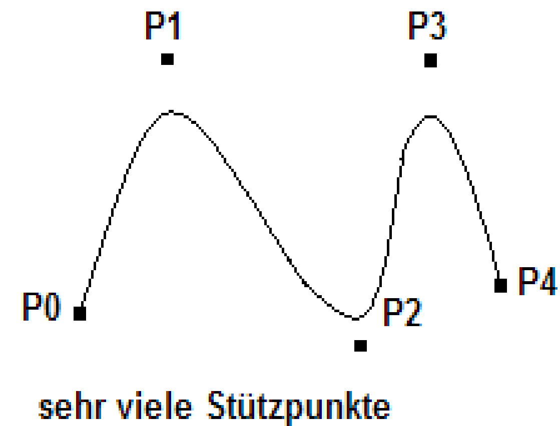
```
   glBegin(GL_LINE_STRIP);     // oder GL_POINTS
      for (i = 0; i <= 30; i++) //Feinheit Kurve
//Funktion erhält einen parametrierten Wert entlang der Kurve, d.h.
Punkte der Kurve können über einen einzigen Parameter abgelaufen werden
 →  glVertex()command wird aufgerufen
         glEvalCoord1f((GLfloat) i/30.0);  //hier wird Feinheit definiert
   glEnd();
```

Evaluators – Beispiel Bézierkurve

Feinheit einer Kurve = Anzahl der Stützpunkte die zwischen Start- und Endpunkt definiert werden



sehr wenig Stützpunkte



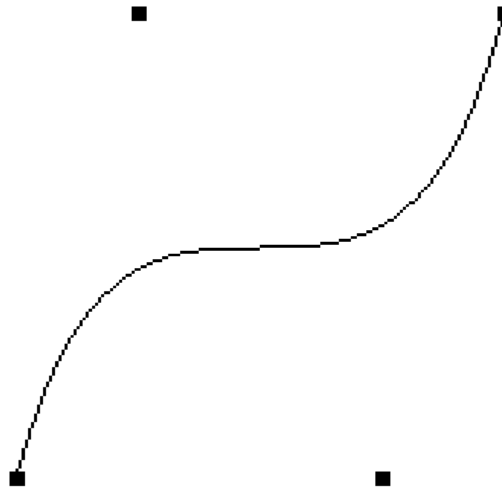
sehr viele Stützpunkte

Evaluators – Beispiel Bézierkurve

5. Zeichnen der Kontrollpunkte

```
glPointSize(5.0);
  glColor3f(1.0, 1.0, 0.0);
  glBegin(GL_POINTS);
     for (i = 0; i < 4; i++)
        glVertex3fv(&ctrlpoints[i][0]);
  glEnd();
glFlush();
```

Ergebnis:



Evaluators – Funktion: glMapGrid()

Evaluators - Funktion: glMapGrid

- Vereinfachung der Evaluation einer Kurve/Fläche → kompakter, effizienter
- Erstellen eines gleichmäßigen Rasters von Punkten in regelmäßigen Abständen über der Domain u (= Bereich u)

Ablauf:

1. Erstellen eines Rasters mit der Funktion glMapGrid1d

- **In init()**

```
glMapGrid1d(100, 0.0, 100.0) //100 Punkte, Parameter u1 - u2
```

```
void glMapGrid1{fd}(GLint n, TYPEu1, TYPEu2);
```

glMapGrid definiert ein Raster von Parameter u1 – u2 in n Schritten

Evaluators – Funktion: glMapGrid()

2. Evaluiere das Raster durch Verwendung von Linien

- **In display()**

```
glEvalMesh1(GL_LINE,0,100);
```

glEvalMesh1() wendet das aktuelle glMapGrid auf alle Evaluators an

*void **glEvalMesh1**(GLenum **mode**, GLint **p1**, GLint **p2**);*

glMapGrid und glEvalMesh ersetzen vorher benötigte Schleife

zum Zeichnen der Kurve!

```
/*glBegin(GL_LINE_STRIP);
	for (i = 0; i <= 30; i++)
		glEvalCoord1f((GLfloat) i);
glEnd();*/
```

Evaluators – Beispiel Bézierfläche

Beispiel einer Bézierfläche

- Aufbau ähnlich einer Bézierkurve

1. Erweiterung der Kontrollpunkte um eine zusätzliches Arraydimension (für Parameter v)

- **Global definieren**

```
GLint nNumPoints = 4;
GLfloat ctrlpoints[4][4][3] = {          // u, v , (x,y,z)
   {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
    {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
   {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
    {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
   {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},
    {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
   {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},
    {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};
```

Evaluators – Beispiel Bézierfläche

2. Rendering Code erweitern: Funktion glMap2f - Spezifiziert Kontrollpunkte entlang zweier Domains (u und v) anstelle einer (u)

- **In init()**

glMap2() und glEvalCoord2() um einen zweidimensionalen Evaluator zu definieren und auszuführen

```
void glMap2{fd}( GLenum target,
        TYPEu1, TYPEu2, GLint ustride, GLint uorder,
        TYPEv1, TYPEv2, GLint vstride, GLint vorder,
        TYPE points );
```

Evaluators – Beispiel Bézierfläche

```
glClearColor (0.0, 0.0, 0.0, 0.0);   //Bildschirm löschen

...

glMap2f(GL_MAP2_VERTEX_3, //Erstellt Evaluator für Koordinaten x,y z

0.0f,        // siehe Bézierkurve

10.0f,       // “”

3,           // “”

4,           // “”

0.0f,        // Unterer Startwert für v Bereich

10.0f,       // Oberer Wert für v Bereich, bestimmt Feinheit

12,          // Abstand zwischen den Punkten (3 x 4 = 12)

4,           // Dimension in v- Richtung, 4 Kontrollpunkte in v-
                Richtung

&ctrlpoints[0][0][0]);   // Arraydimensionen von Kontrollpunkten
```

Evaluators – Beispiel Bézierfläche

3. Ausführen des Evaluators

- **In init()**

```
glEnable(GL_MAP2_VERTEX_3);
```

4. Grid Funktion

- **GlMapGrid2 in init(), glEvalMesh2 in display()**

```
void glMapGrid2{fd}(
        GLint nu, TYPEu1, TYPEu2,
        GLint nv, TYPEv1, TYPEv2 );
```

```
glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0); // n = 20, Parameter (u,v)
```

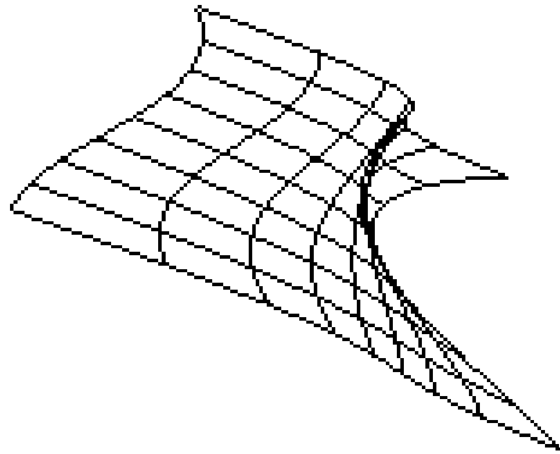
Evaluiere das Raster durch Verwendung von Linien

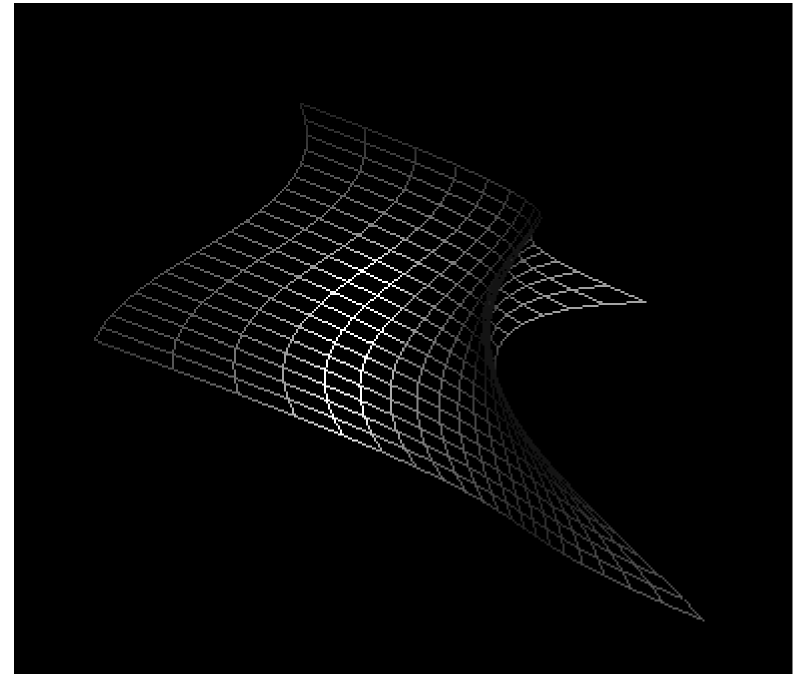
```
void glEvalMesh2( GLenum mode,
        GLint p1, GLint p2,
        GLint q2, GLint q2 );
```

```
glEvalMesh2(GL_LINE,0,20,0,20);
```

Evaluators – Beispiel Bézierfläche

Resultat:





Evaluators - Features

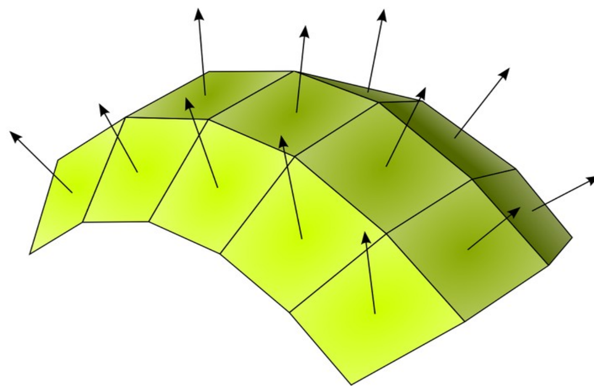
Evaluator Features

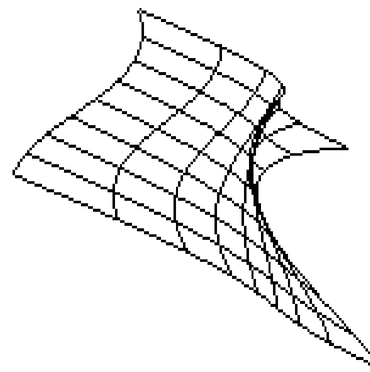
- Bsp. Fläche füllen

```
glEvalMesh2(GL_FILL, 0, 20, 0, 20);  //anstelle GL_LINE

glEnable(GL_AUTO_NORMAL) // Autonormalvektor wird generiert
                                → Auswirkung nur auf Beleuchtung!
```

GL_AUTO_NORMAL

Wenn aktiviert, werden automatisch Normalen (Normalvektoren) generiert für eine Oberfläche, wenn GL_MAP2_VERTEX_3 oder GL_MAP2_VERTEX_4 zur Erstellung von Vertices genutzt werden.





vorher: GL_LINE

nachher: GL_FILL

NURBS - Einführung

Warum NURBS?

- Für komplexere Kurven oder Oberflächen reichen Bézierkurven nicht aus
- Je mehr Punkte in einer Bézierkurve-/fläche vorhanden sind (ab 5 Kontrollpunkten) desto ungenauer ist die Feinheit des Objekts, da hinzukommende Kontrollpunkte an der Kurve ziehen und reißen.
- stückweises Zusammensetzen aus Bézierkurven oder Oberflächen an Schnittstellen ist nicht einfach
- Es muss gewährleistet werden dass der Endpunkt der einen Kurve gleich dem Anfangspunkt der Zweiten Kurve ist (gleichmäßiger Übergang)

NURBS - Einführung

- Tangentiale Stetigkeit muss gewahrt bleiben (zwei Kurvenstücke im gemeinsamen Punkt müssen die gleiche Tangente haben)

- Ist die Tangentiale Stetigkeit nicht gegeben sieht man z.B. Eine unerwünschte Kante.

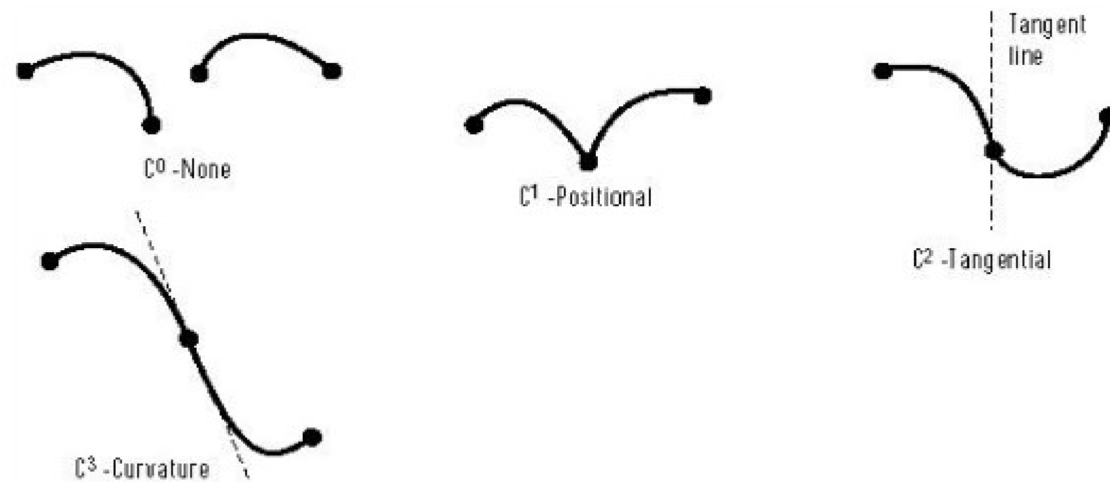


Figure 17-3 Continuity of piecewise curves

NURBS - Einführung

Was sind NURBS?

- NURBS supported in einer Higher Level Utility Library (glu)

  #include <glu.h>

  GLU stellt dem Anwender eine erweiterte OpenGL-Funktionalität zur Verfügung

- **NURBS** (**N**on-**U**niform **R**ational **B-S**pline) surfaces

- Evaluator stellen Grundlage für NURBS da

- Kurven- oder Schnittlinien, Oberflächen und Volumina exakt berechenbar

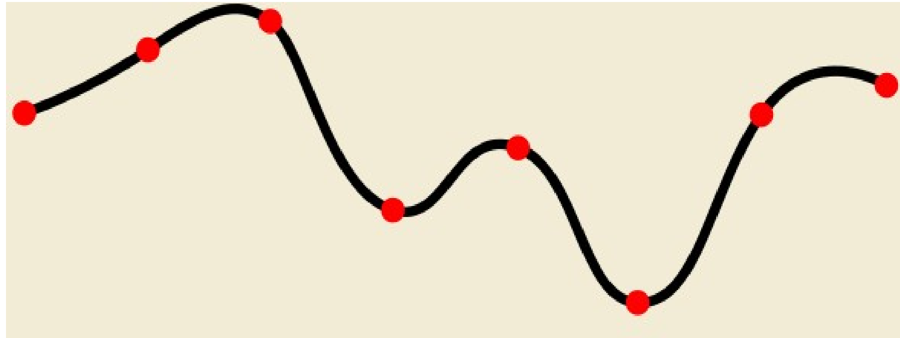
- Nurbs-Flächen stellen das Nonplusultra innerhalb der Freiformflächenerzeugung dar

NURBS – B-Splines

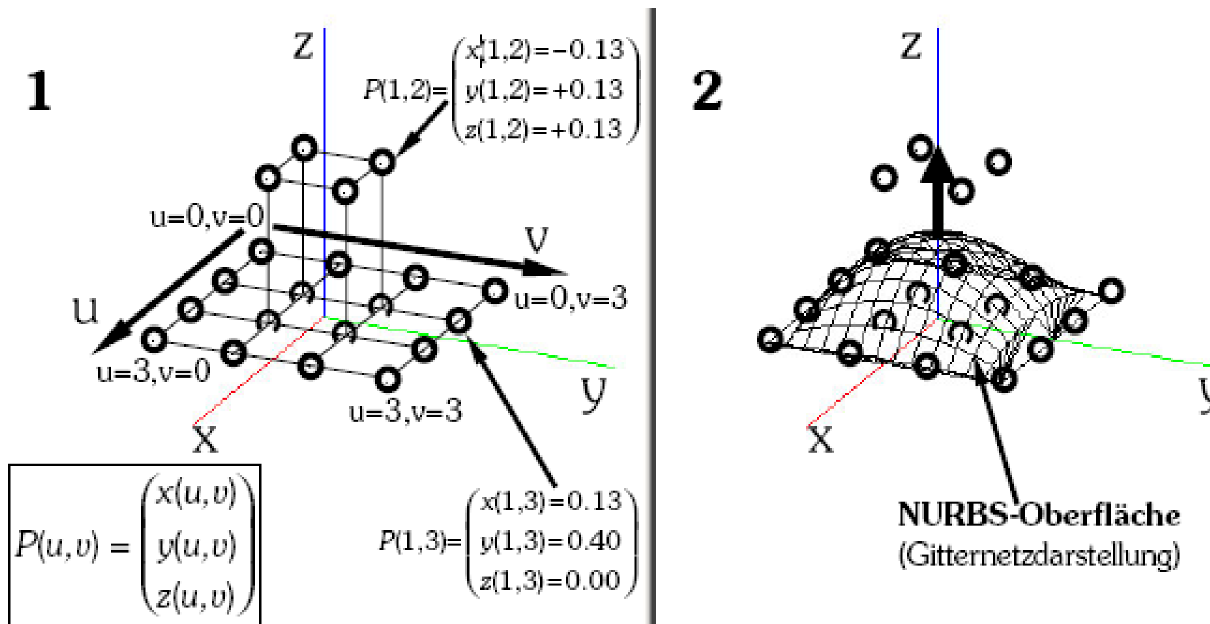
<u>Lösung</u>: Beziér → **B**(i-cubic) – **Splines:**

- *Ein **Spline n-ten Grades** ist eine Funktion, die stückweise aus Polynomen mit maximalem Grad n zusammengesetzt ist*
- B-Splines (Bi-cubic splines) arbeiten ähnlich Bézierkurven, jedoch wird die Kurve in mehrere Segmente unterteilt welche von den vier nächsten Kontrollpunkte beeinflusst wird
- <u>Segment</u>: mathematische Funktion (Polynom) zwischen zwei Punkten
- Ein Segment hat die Charakteristika einer Bézierkurve der Ordnung 4
- Eine komplexe Kurve mit vielen Kontrollpunkten wird somit von Haus aus feiner, der Kurvenverlauf kontinuierlicher

NURBS – B-Splines



- Sement entspricht einer Kurve zwischen 2 Kontrollpunkten
- Eine Wölbung des Segments entsteht durch 4 Kontrollpunkte die das Segement “anziehen”



NURBS - Knots

Knoten

- Mit Nurbs können die vier Kontrollpunkte des Segments je nach Genauigkeit optimiert werden (Wie “fein” soll die Kurve sein?!)

- Knots stellt dieses Verfahren zur Verfügung

- Knots ist eine Sequenz aus Werten, zwei Knotenwerte sind für jeden Kontrollpunkt definiert

  Beispiel: Bei 3 Kontrollpunkten → Sequenz aus 3 x 2 Werten = 6 Werte

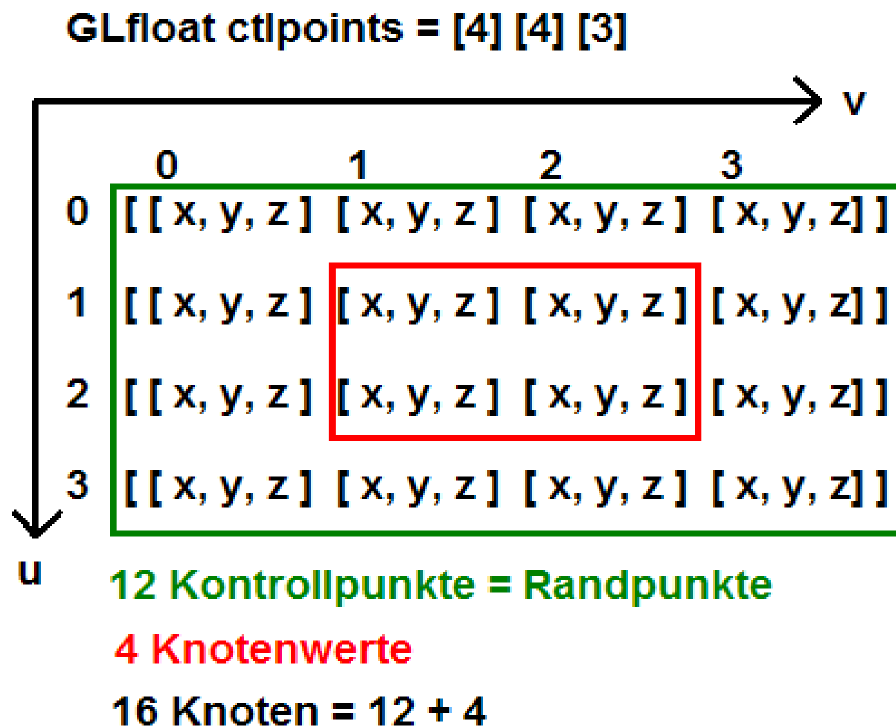
- Knotenwerte sind ausschlaggebend für den Einfluss der Kontrollpunkte im definierten Bereich (u,v)

NURBS - Knots

- Mehr als u x v = 8 x 8 = 64 Kontrollpunkte können zur Definition einer NURBS

Fläche zurzeit nicht verwendet werden. Höhere Indexwerte als 8 für u und v

führen zu einem GL_INVALID_VALUE- OpenGL- Fehler

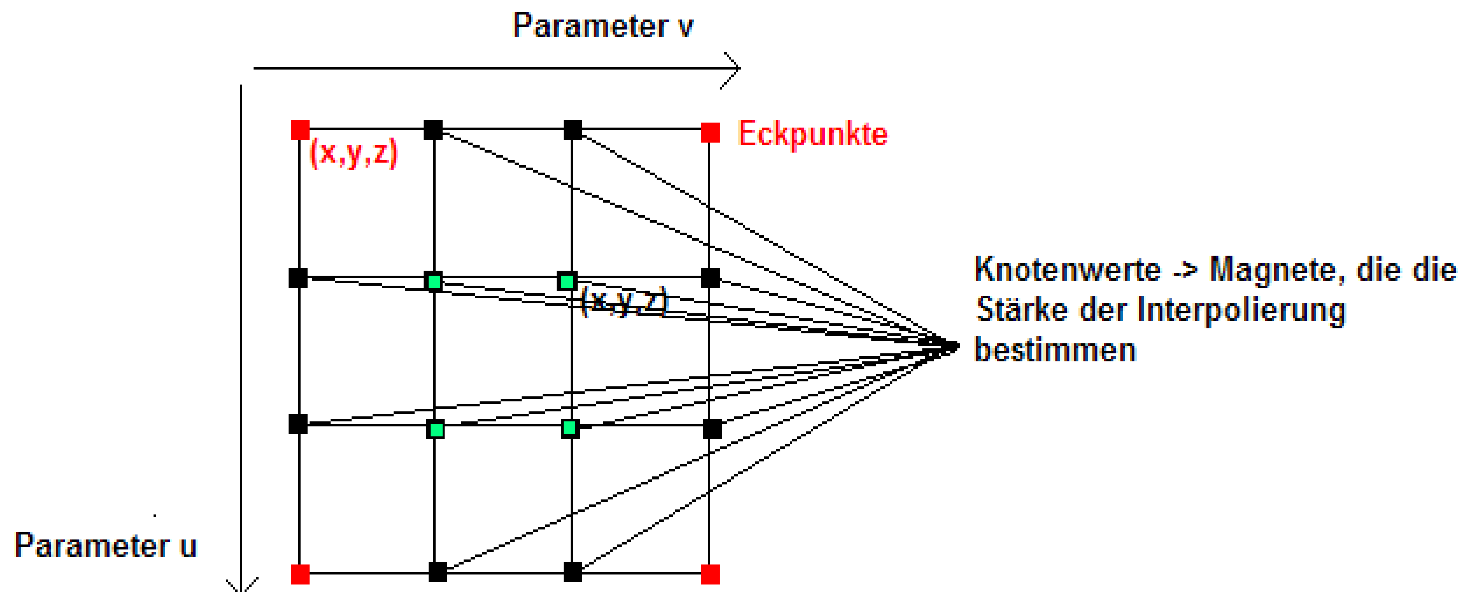


Dreidimensionales Array. Die Punkte in der Mitte fungieren als starke Magnete →

ziehen Objekt zu sich heran. Randpunkte haben keine solche “Anziehungskraft”

NURBS - Knots

- Beispiel: Gegeben ist eine Grundfläche die mittels Parameter u und v definiert ist
- Jeder der Knoten besteht aus einem Eckpunkt (x,y,z)
- Um eine Wölbung zu erreichen werden alle Punkte, die in der Matrix zwischen den Eckpunkten liegen, mit Hilfe des Eckpunkts interpoliert



NURBS - Oberfläche

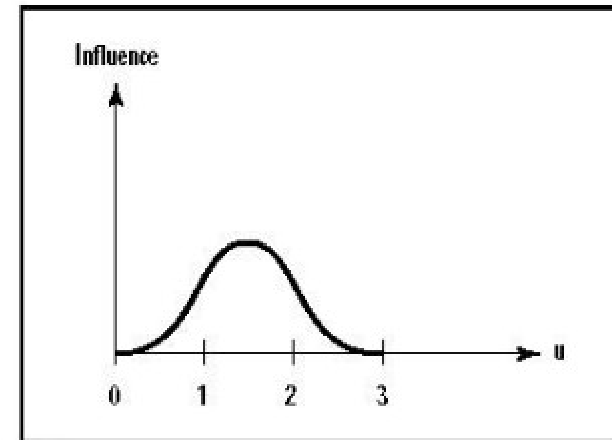
- Knoten- Sequenz definiert die Stärke des Einflusses von Punkten (Knot Multiplicity)

Einfluss von Kontrollpunkten über eine Kurve

mit 4 Einheiten im Parameterbereich (u)

Punkte in der Mitte haben einen größeren

Einfluss auf die Kurve (ziehen diese stärker an)



NURBS Oberfläche

- GLU NURBS Funktionen stellen ein nützliches High-level Interface zur Verfügung

um Oberflächen zu rendern

- Es müssen nicht explizit Evaluators, Maps oder Grids aufgerufen werden

NURBS – Oberfläche Beispiel

1. referenziertes NURBS Objekt erzeugen

- Globales referenziertes NURBS Objekt wird erzeugt
- bei jedem Aufruf einer NURBS Funktion wird darauf zugegriffen
- Modifiziert Oberfläche eines Objekts

GLUnurbsObj **gluNewNurbsRenderer** (void);*

*void **gluDeleteNurbsRenderer** (GLUnurbsObj *nobj);*

- **NURBS- Objekt global definieren!**

```
GLUnurbsObj *theNurb = NULL;      // NURBS Objekt-Zeiger
```

- **Erzeugtes Objekt in init()**

```
pNurb = gluNewNurbsRenderer();   // NURBS Objekt erzeugen
.. //Code
gluDeleteNurbsRenderer(pNurb);   //Löschen des Nurbs Objekts
```

NURBS – Oberfläche Beispiel

2. High-level Eigenschaften für ein NURB Objekt erzeugen

- **In init()**

```
void gluNurbsProperty(GLUnurbsObj *nobj, GLenum property, GLfloat value);
```

- *Wenn Nurbs gerendert wird können verschiedene Eigenschaften eingestellt werden*
- *z.B. Maximale Anzahl an Linien, Polygonen, Wie ist Oberfläche gerastert (Filled, wireframe), Präzision der Tesselation*

Beispiel: GLU_DISPLAY_MODE für grundsätzliche Darstellungsart der NURBS- Fläche

GLU_FILL, GLU_OUTLINE_POLYGON (Umrissdarstellung der Flächenstücke des Nurbs)

```
gluNurbsProperty(pNurb, GLU_DISPLAY_MODE,(GLfloat)GLU_FILL);
```

NURBS – Oberfläche Beispiel

*void **gluLoadSamplingMatrices** (GLUnurbsObj *nobj, const GLfloat modelMatrix[16], const GLfloat projMatrix[16], const GLint viewport[4]);*

Beispiel:

GLU_AUTO_LOAD_MATRIX = automatische Matrizenbehandlung (falls

GL_TRUE) → lädt Projektionsmatrix, Modelviewmatrix, Viewport

*void **gluGetNurbsProperty** (GLUnurbsObj *nobj, GLenum property, GLfloat*value);*

- *Um aktuell angewendete Eigenschaft für NURBS abzufragen*

*void **gluNurbsCallback** (GLUnurbsObj *nobj, GLenum which, void (*fn)(GLenum errorCode));*

- *Informiert über Error -> Rechenleistung aufwendiger*

NURBS – Oberfläche Beispiel

3. Definieren der Oberfläche **in Routine display()**

void ***gluBeginSurface*** *(GLUnurbsObj *nobj);*

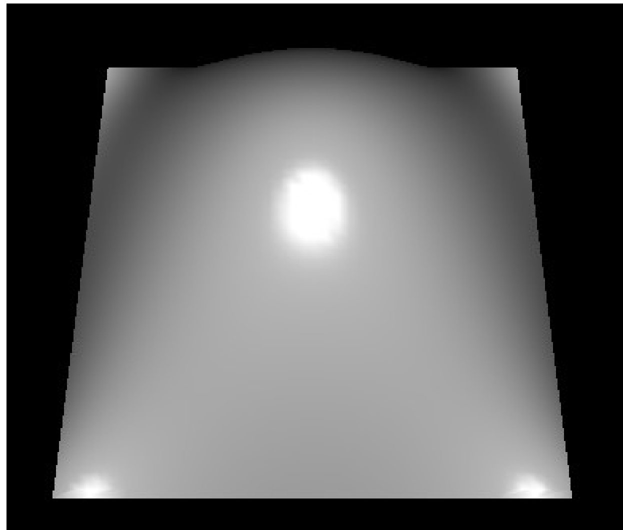
void ***gluEndSurface*** *(GLUnurbsObj *nobj);*

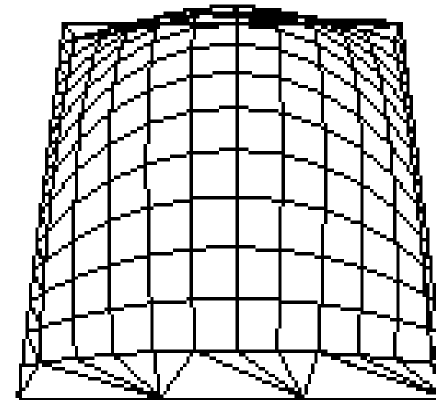
void ***gluNurbsSurface*** *(GLUnurbsObj *nobj, GLint uknot_count,*
*GLfloat *uknot, GLint vknot_count, GLfloat *vknot,*
*GLint u_stride, GLint v_stride, GLfloat *ctlarray,*
GLint uorder, GLint vorder, GLenum type);

```
// NURB Rendern
gluBeginSurface(pNurb);          // Beginn der NURB Definition,Oberfläche definieren
    gluNurbsSurface(
    pNurb,                       //Zeiger auf NURBS renderer
    8, Knots,                    //Anzahl der Knoten(4 * je  2 Knotenwerte)  und Knoten-Array in u- Richtung
    8, Knots,                    //Anzahl der Knoten(4 Kontrollpunkte * 2 ) und Knoten-Array in v- Richtung
    4 * 3,                       //Abstand zwischen den Kontrollpunkten in u - Richtung
    3,                           //Abstand zwischen Kontrollpunkten in v Richtung
    &ctrlPoints[0][0][0],        //Kontrollpunkte
    4, 4,                        //Ordung von u,v der Oberfläche
    GL_MAP2_VERTEX_3);           //Oberflächen-Typ
gluEndSurface(pNurb);            //Oberfläche fertig!
```

NURBS – Oberfläche Beispiel

Dieses Bild zeigt eine gerenderte NURBS surface in Form eines symmetrischen Hügels mit control points im Bereich von -3.0 to 3.0. Basis der Funktion ist eine B-kubische Spline





NURBS - Trimming

Trimming:

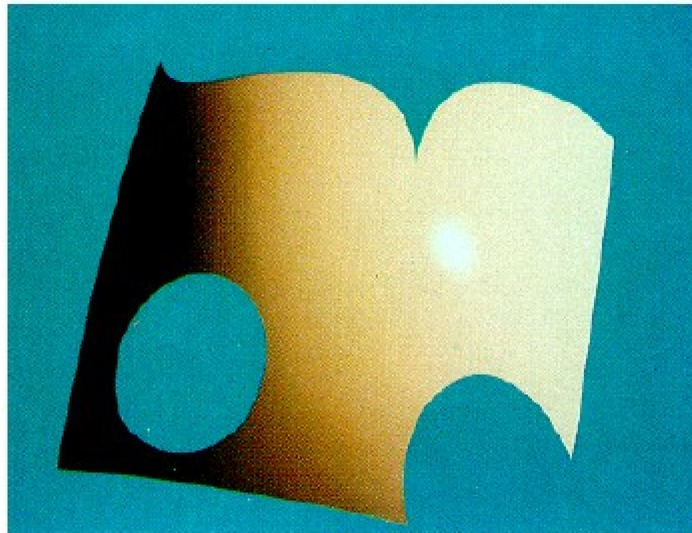
- Ausschnitte aus einer NURB Oberfläche (Löcher)
- Innerhalb der **gluBeginSurface** und der **gluEndSurface** wird eine Funktion **gluBeginTrim** aufgerufen, welche eine getrimmte Kurve mit **gluPwlCurve** spezifiziert und mit **gluEndTrim** beendet
- Trimming Kurven müssen geschlossen sein und dürfen sich nicht überschneiden
- **gluPwlCurve()** = erzeugt eine stückweise lineare Kurve
- **gluNurbsCurve()** = erzeugt eine NURBS Kurve

Ablauf

1. Äußere und Innere Trimming Punkte für Oberfläche festlegen

- Äußere und Innere Trimming Punkte dienen als Werkzeug um ein Objekt richtig auszurichten
- Äußere Trimming Punkte welche die gesamte Oberfläche includieren verlaufen gegen den Uhrzeigersinn
- Innere Trimming Punkte welche den eigentlichen Ausschnitt darstellen verlaufen im Uhrzeigersinn

NURBS - Trimming



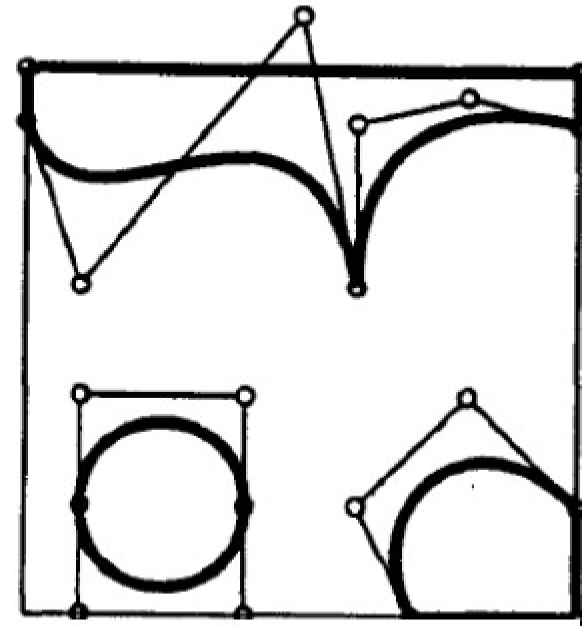
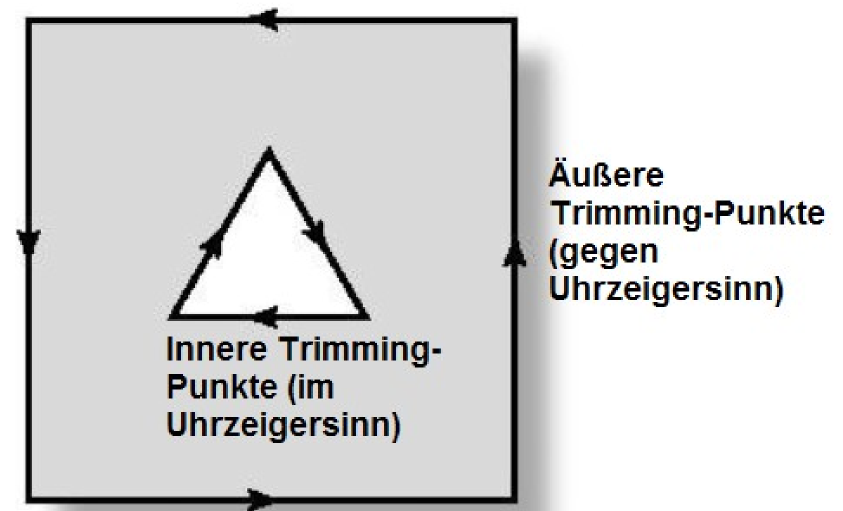
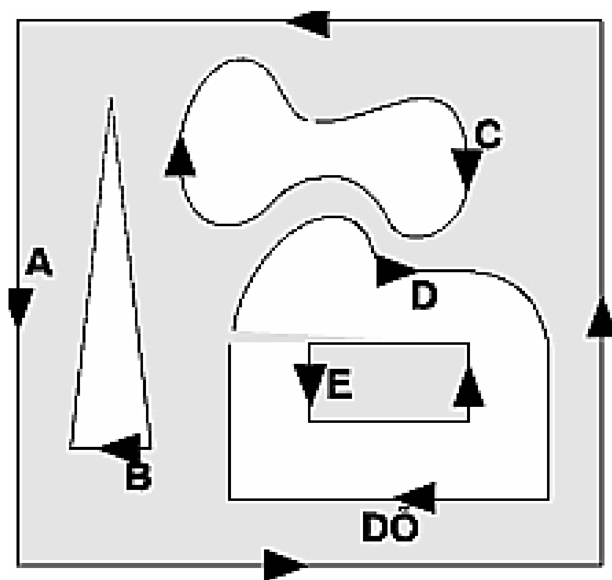


Abbildung: Definition von (NURBS-) Kurven in Parameterebene

NURBS - Trimming

- Korrekte Trimmingkurven






```
gluBeginSurface();
gluNurbsSurface(...);
gluBeginTrim();
gluPwlCurve(...); /* A */
gluEndTrim();
gluBeginTrim();
gluPwlCurve(...); /* B */
gluEndTrim();
gluBeginTrim();
gluNurbsCurve(...);/* C */
gluEndTrim();
gluBeginTrim();
gluNurbsCurve(...);/* D */
gluPwlCurve(...); /* DŌ */
gluEndTrim();
gluBeginTrim();
gluPwlCurve(...); /* E */
gluEndTrim();
gluEndSurface();
```

NURBS – Trimming - Beispiel

- **In display()**

```
//Äußere Trimming-Points um die gesamte Oberfläche zu
includieren - Gegen den Uhrzeigersinn
GLfloat outsidePts[5][2]= /*counterclockwise*/
{{0.0f, 0.0f},{1.0f, 0.0f}, {1.0f, 1.0f}, {0.0f, 1.0f}, {0.0f, 0.0f}};

//Innere Trimming-Points zum Erzeugen eines Dreiecks in der
Oberfläche - im Uhrzeigersinn
GLfloat insidePts[4][2]= /*clockwise*/
{{0.25f, 0.25f},{0.5f, 0.5f}, {0.75f, 0.25f}, {0.25f, 0.25f}};
```

NURBS – Trimming Beispiel

2. Äußeren Bereich für gesamte Kurve und inneren Bereich für Dreieck definieren

- **In display()**

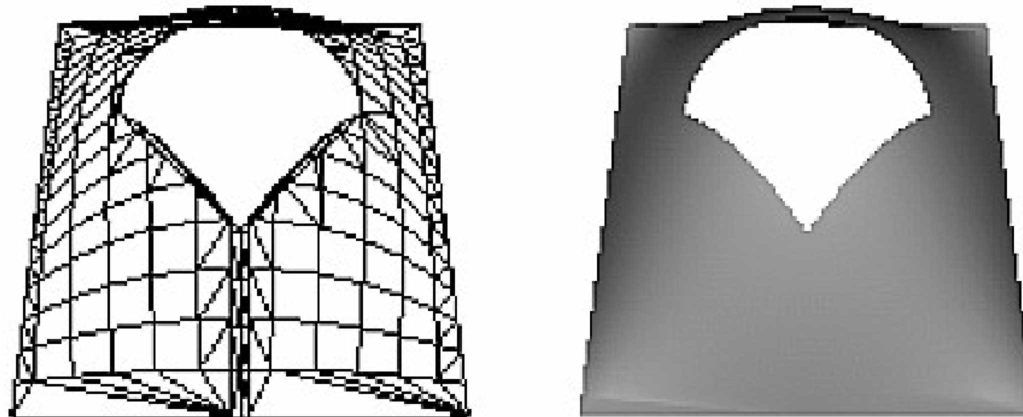
```
void gluBeginTrim (GLUnurbsObj *nobj);

void gluEndTrim (GLUnurbsObj *nobj);

void gluPwlCurve (GLUnurbsObj *nobj, GLint count, GLfloat *array,
    GLint stride, GLenum type);
```

```
//Äußerer Bereich
gluBeginTrim(pNurb);
    gluPwlCurve(pNurb,
    5,                   //Äußere Punkte
    &outsidePts[0][0], 2, GLU_MAP1_TRIM_2);
gluEndTrim(pNurb);
```

NURBS – Trimming Beispiel

```
// Innerer Bereich
gluBeginTrim(pNurb);
    gluPwlCurve(pNurb, 4, &insidePts[0][0], 2, GLU_MAP1_TRIM_2);
    //gluPwlCurve definiert stückweise eine lineare Kurve- eine
      Menge von Punkten verbunden von Endpunkt zu Endpunkt.
gluEndTrim(pNurb);
```

Resulat:



Fragen?

- Quellen: