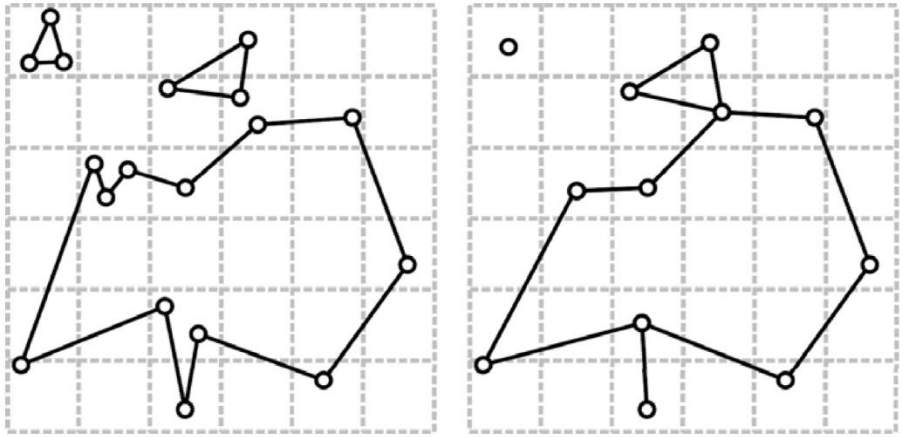
Mesh-Dezimierung

- Oft resultiert eine große Polygon-Anzahl aus dem Meshing
- Mesh-Dezimierung versucht die Polygon-Anzahl zu verringern, die Geometrie jedoch so weit wie möglich zu erhalten

Vertex Clustering

- Abbildung des Meshs in ein gleichmäßiges Gitter
- Für Knoten die innerhalb einer Gitterzelle liegen: repräsentativen Knoten bestimmen (z.B. Mittelwert)
- Verbindung der repräsentativen Knoten über Gitterzellengrenzen hinweg zu neuen Kanten (sofern es dort zuvor eine Kante gab)

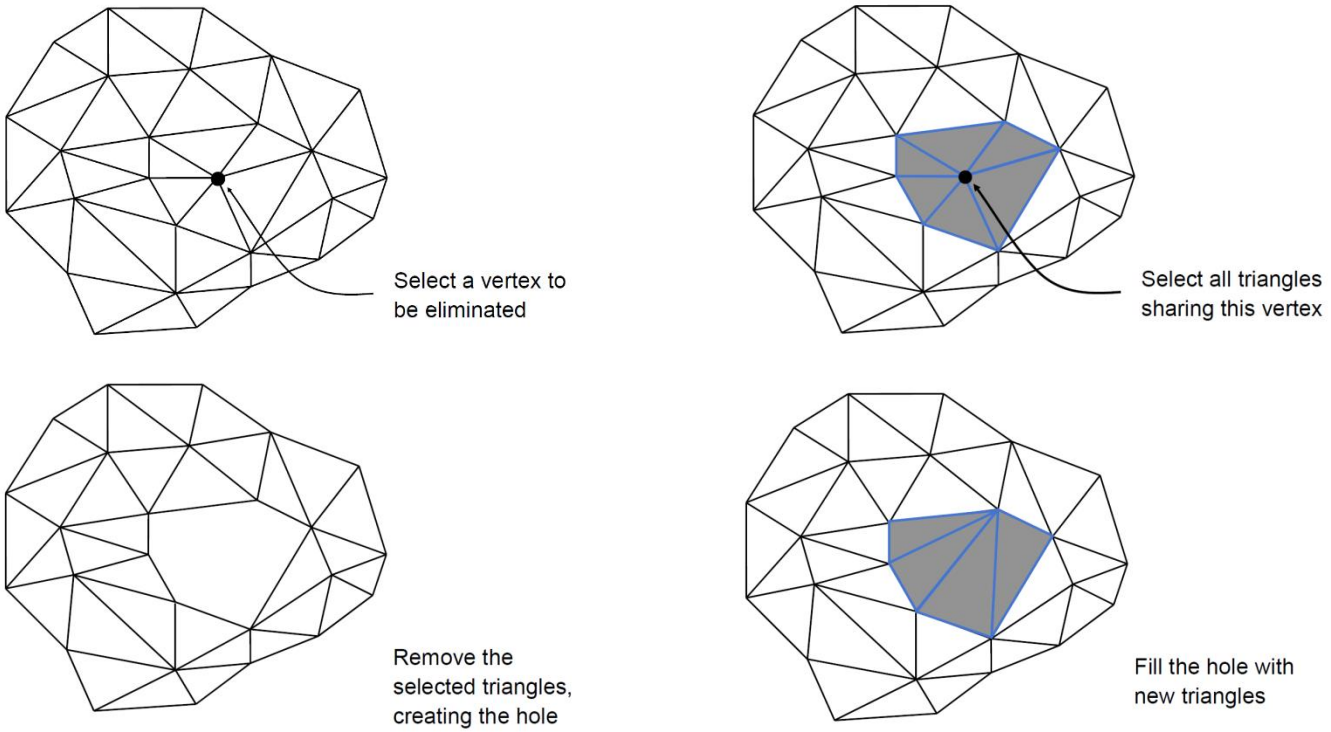


http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf

Mesh-Dezimierung

Incremental Decimation:

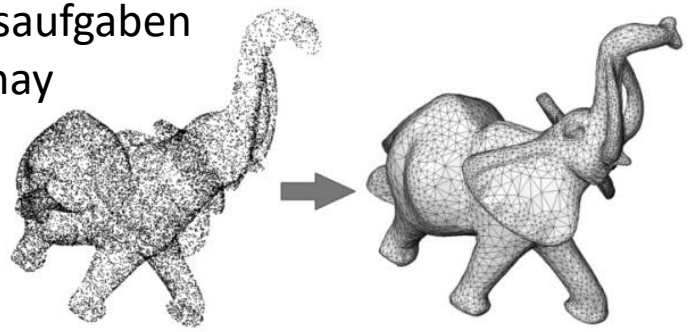
- Grundidee: Auswahl einer Mesh-Region, z.B. Umbrella-Region. Darin Anwendung eines Dezimierungs-Operators
- Beispiel-Operator „Vertex removal“:



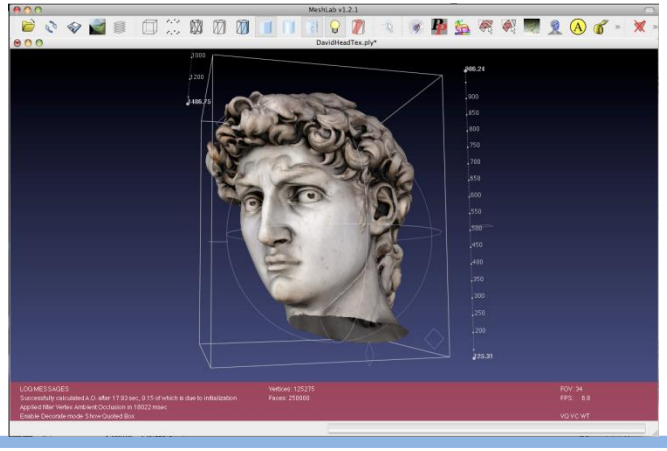
http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf

Softwarepakete für Meshing (Beispiele)

- CGAL: The Computational Geometry Algorithms Library
  - Open Source Bibliothek (C++) für geometrische Berechnungen
  - Enthält diverse Algorithmen für Polygonisierungsaufgaben (u.a. 2D / 3D Surface Mesh basierend auf Delaunay Triangulation)
  - Sehr gutes Manual: http://doc.cgal.org/latest/Manual/index.html

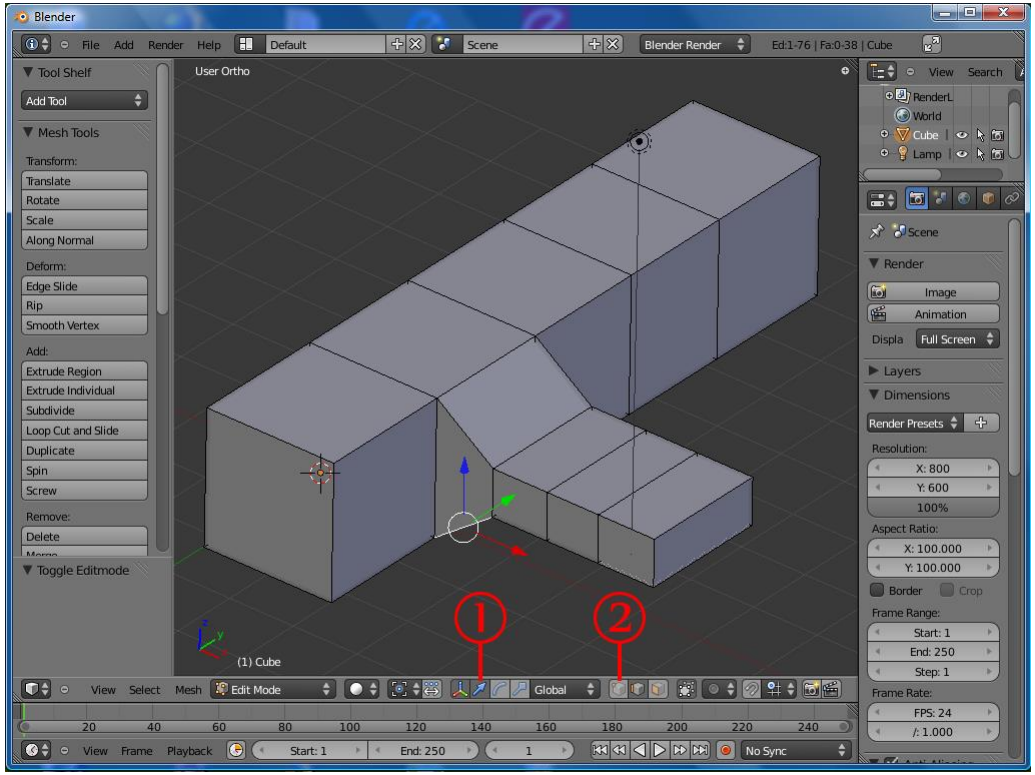


- MeshLab
  - Open Source zur Erstellung und Bearbeitung von 3D-Dreieck-Polygonnetzen
  - http://meshlab.sourceforge.net/
  - Grafische Benutzeroberfläche



Meshgenerierung durch CAD

- Erzeugung komplexer Körper in OpenGL ohne Datengrundlage schwierig
- Meist Modellierung in entsprechender CAD-Anwendung oder 3D Grafikwerkzeugen, z.B. Blender

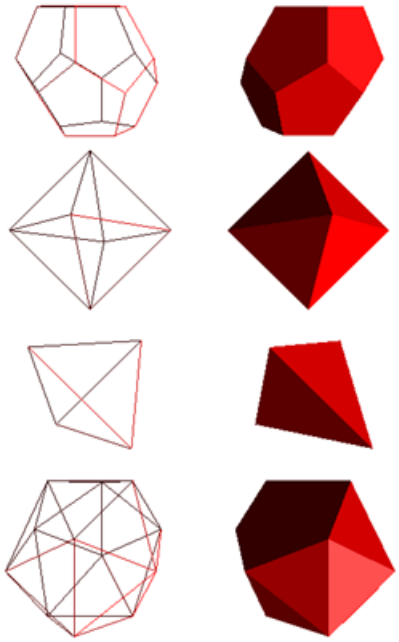


Geometrische Körper in GLUT

- Würfel:
  - **`glutWireCube(size):`** Würfel-Drahtmodell mit Seitenlänge *size*
  - **`glutSolidCube(size):`** Gefüllter Würfel, Seitenlänge size
- Kugel:
  - **`glutWireSphere(radius, slices, stacks):`** Kugel mit Radius *radius*, Mittelpunkt im Ursprung, Anzahl der Längengrade (*slices*) und Breitengrade (*stacks*).
  - **`glutSolidSphere(radius, slices, stacks):`** analog
- Kegel:
  - **`glutWireCone(base, height, slices, stacks):`** Kegel mit Grundradius *radius*, Grundfläche liegt in der x/y-Ebene mit Mittelpunkt im Ursprung, Höhe des Kegels in z-Richtung (*height*). Anzahl der Längengrade (*slices*) und Breitengrade (*stacks*).
  - **`glutSolidCone(base, height, slices, stacks):`** analog
- Torus:
  - **`glutWireTorus(innerRadius, outerRadius, nsides, rings):`** Torus mit Mittelpunkt im Ursprung und z-Achse als Achse. Zahl der Seiten in jedem Radialschnitt *nsides* und Zahl der Radialschnitte (*rings*).
  - **`glutSolidTorus(innerRadius, outerRadius, nsides, rings):`** analog

Geometrische Körper in GLUT

- Platonische Polyeder, (jeweils Mittelpunkt im Ursprung)
  - Dodekaeder:
    - **`glutWireDodecahedron(void)`:** Radius $\sqrt{3}$
    - **`glutSolidDodecahedron(void)`**
  - Oktaeder:
    - **`glutWireOctahedron(void)`:** Radius 1
    - **`glutSolidOctahedron(void)`**
  - Tetraeder:
    - **`glutWireTetrahedron(void)`:** Radius $\sqrt{3}$
    - **`glutSolidTetrahedron(void)`**
  - Ikosaeder:
    - **`glutWireIcosahedron(void)`:** Radius 1
    - **`glutSolidIcosahedron(void)`**



- Teekanne
  - **`glutWireTeapot(scale)`**: klassische Teekanne, um Faktor *scale* gestreckt
  - **`glutSolidTeapot(scale)`**: analog

http://www.programming-techniques.com/2012/05/rendering-dodecahedron-octahedron.html

3.3. FREIFORMFLÄCHEN

Freiformkurven- und Flächen

- Polygonnetze aus explizit definierten planaren Polygonen approximieren gekrümmte Flächen i.d.R. durch eine Vielzahl von Polygonen
- Andere Möglichkeit für eine speichersparende Repräsentation: parametrische Beschreibung gekrümmter Flächen ➔ „Freiformflächen“
- Vorteile:
  - Ermöglicht Arbeiten mit unterschiedlichen Auflösungen → Tessellierung je nach Bedarf.
  - Normalenvektoren für Lichtberechnung können direkt aus gekrümmter Fläche abgeleitet werden.

- Gewünschte Eigenschaften:
  - **Kontrollierbarkeit**: Einfluss der Parameter für Benutzer intuitiv verständlich
  - **Lokalitätsprinzip**: Möglichkeit die Kurve/Fläche lokal zu verändern.
  - **Glattheit**: Kurve/Fläche soll gewisse Glattheitseigenschaften besitzen
    - Stetig (=keine Lücken/Sprünge) (C0-Kontinuität)
    - mindestens einmal stetig differenzierbar (= keine Knicke) (C1-Kontinuität)
    - besser zweimal stetig differenzierbar (= keine abrupte Änderung der Krümmung) (C2-Kontinuität)

Freiformkurven- und Flächen

- Beschreibung erfolgt mittels **parametrischer Funktionen**
  - Punkte (x, y, z) auf der Kurve werden als Funktion einer oder mehrerer Variablen durchlaufen:

$$\begin{aligned} x &= x(t) \\ y &= y(t) \\ z &= z(t) \end{aligned} \qquad t_1 < t < t_2$$

  - Für jeden Wert t aus $[t_1, t_n]$ erhält man eine kartesische Koordinate (x, y, z) eines Punktes
  - Menge der Punkte legt die Funktionskurve der durch den Parameter t dargestellten Funktion fest.
  - Beschreibung einer Kurve im Raum: 1 Parameter *(t)*
  - Beschreibung einer Fläche im Raum: 2 Parameter *(t,s)*

- Beispiel: parametrische Beschreibung eines Kreises

$$r(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix} \qquad f\ddot{u}r\ 0 \leq t \leq 2\pi$$

Freiformkurven- und Flächen

- Stützpunkte definieren Kurven- bzw. Flächenverlauf
  - *Interpolation*: Kurve läuft exakt durch Stützpunkte

    für $(n+1)$ Stützpunkte existiert immer eine Zerlegung in ein Polynom n-ten Grades

    Nachteile: - oft hoher Polynomgrad → hoher Rechenaufwand
    - Lokalitätsprinzip verletzt
    - Schwingen

  - *Approximation*: Kurve nähert Stützpunkte gut an

    → gewünschte Eigenschaften können erhalten werden

- Bekannte Ansätze für Freiformkurven/-flächen u.a.:
  - Bézierkurven/-flächen
  - B-Splines
  - Non-uniform rational B-Splines (NURBS)

Bézierkurven

- Grundbaustein: Bernstein-Polynome
- Besondere Familie reeller Polynome mit ganzzahligen Koeffizienten

Das i-te Bernstein-Polynom n-ten Grades (mit $i \in \{0, \dots, n\}$) ist durch die Gleichung

$$B_i^{(n)}(t) = \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i$$

gegeben, wobei $t \in [0,1]$.

Erinnerung aus der Kombinatorik (Binomialkoeffizient):

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!} = \frac{\prod_{k=1}^{n} k}{\prod_{k=1}^{i} k \cdot \prod_{k=1}^{(n-i)} k}$$

- Eigenschaften von Bernstein-Polynomen:
  - Im Definitionsbereich $[0, 1]$ nehmen Bernstein-Polynome nur Werte zwischen 0 und 1 an.
  - An jeder Stelle des Definitionsbereichs $[0, 1]$ addieren sich die Bernstein-Polynome zu 1 auf.

Bézierkurven

- **Beispiel für $n = 3$:** einsetzen von $i = 0 \ldots 3$ und $n = 3$ liefert die 4 Bernsteinpolynome

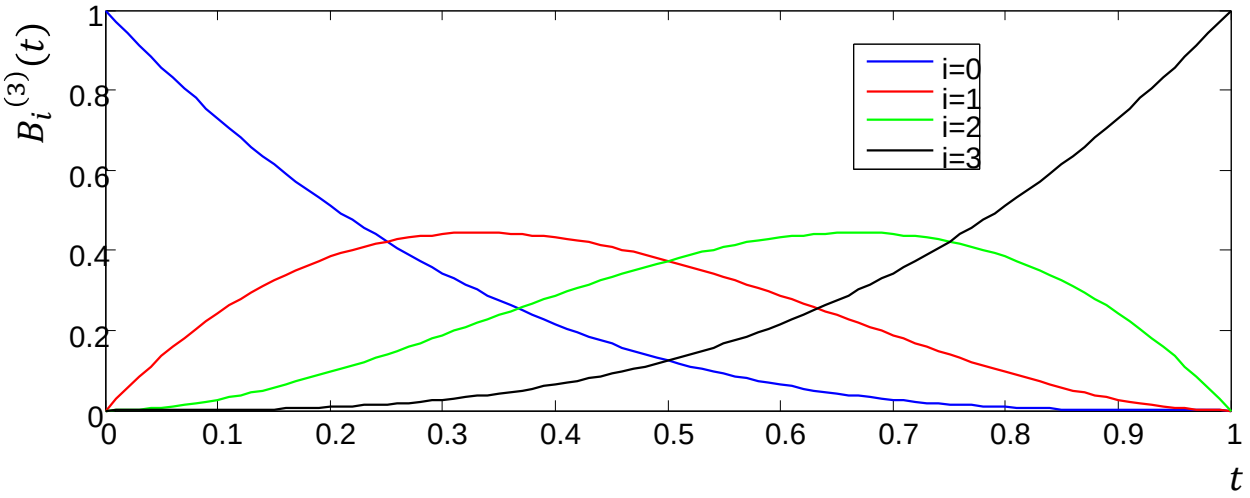
$$B_0^{(3)}(t) = (1-t)^3$$

$$B_1^{(3)}(t) = 3(1-t)^2 t$$

$$B_2^{(3)}(t) = 3(1-t)t^2$$

$$B_3^{(3)}(t) = t^3$$

$$B_i^{(n)}(t) = \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i$$



Bézierkurven

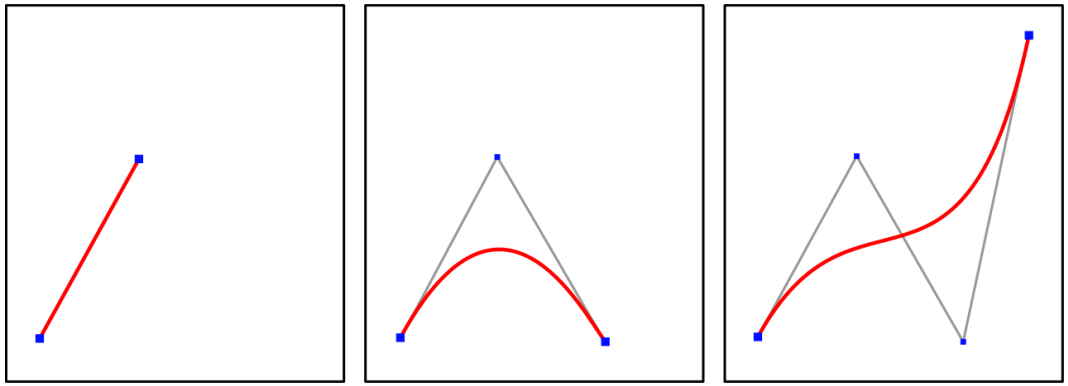
- Bézierkurven verwenden Bernstein-Polynome n-ten Grades zur Approximation von $(n+1)$ Stützpunkten $b_0, \dots, b_n \in \mathbb{R}^p$.
- Die durch diese Stützpunkte definierte Kurve

$$p(t) = \sum_{i=0}^{n} b_i \cdot B_i^{(n)}(t)$$

An jeder Stelle $t \in [0,1]$ der Bézierkurve: Bernsteinpolynom i an der Stelle t gibt ~Gewichtungsfaktor für den Stützpunkt i an

heißt Bézierkurve vom Grad n.

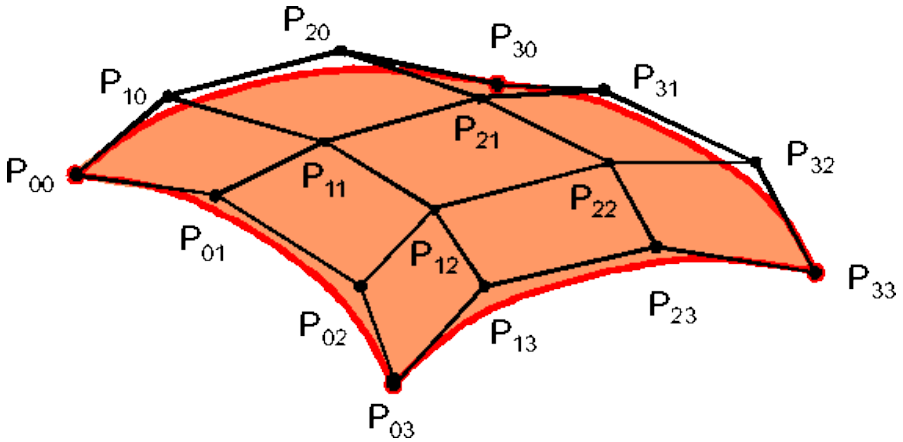
- Bézierkurven interpolieren den Anfangs- und Endpunkt, die anderen Stützpunkte liegen i.d.R. nicht auf der Kurve.



https://de.wikipedia.org/wiki/B%C3%A9zierkurve

Bézierflächen

- Hinzufügen einer Parameterdimension ermöglicht Bézierflächen im Raum
- Statt der Bedingung $t \in [0,1]$ gilt nun $s, t \in [0,1]$.

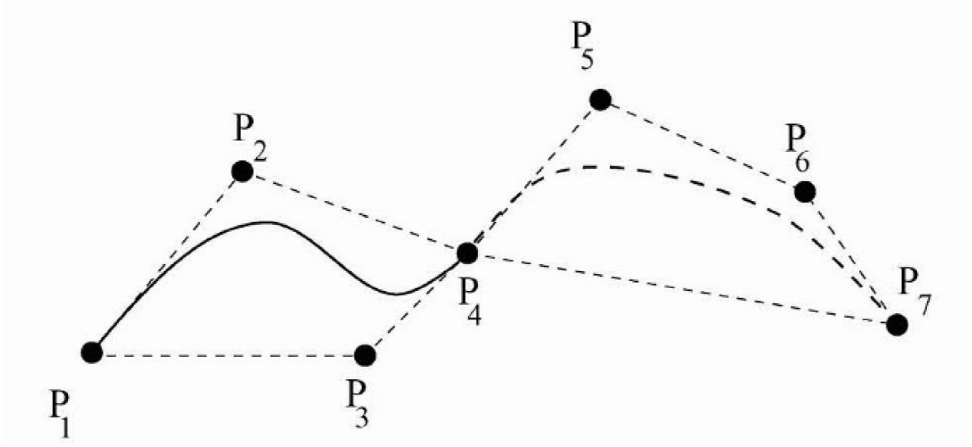


- Eigenschaften:
  - Invariant gegenüber affinen Transformationen (= Rotation, Verschiebung, Skalierung)
  - Symmetrisch gegenüber Stützpunkten, d.h. $b_0, \dots, b_n$ ergibt die selbe Kurve wie $b_n, \dots, b_0$.
- Nachteil Bézierkurven: Hoher Polynomgrad bei vielen Stützpunkten

http://www-lehre.inf.uos.de/~cg/2008/skript/node132.html

Stückweise Bézierkurven

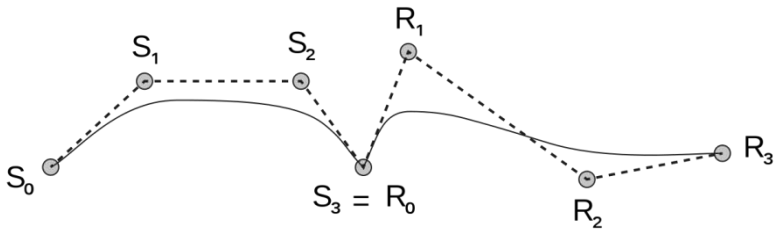
- Setzen sich aus mehreren Bézierkurven niedrigen Grades n zusammen
  - Üblicherweise dritter oder vierter Grad
- Berechnung einer Bézierkurve für jeweils $n + 1$ aufeinander folgende Stützpunkte
- Letzter Stützpunkt der ersten Bézierkurve wird erster Stützpunkt der nächsten usw.
  - Stützpunkte an Nahtstellen werden daher interpoliert



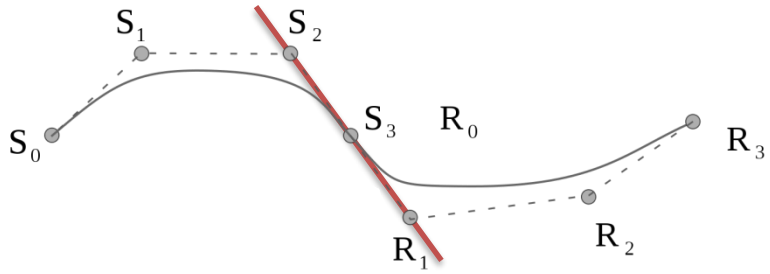
F. Klawonn: „Grundkurs Computergrafik mit Java“, Vieweg und Teubner, 3. Auflage, 2010

Stückweise Bézierkurven

- Vermeidung von Knicken an Nahtstellen durch Kollinearität des vorhergehenden und nachfolgenden Punktes (= C1-Kontinuität)

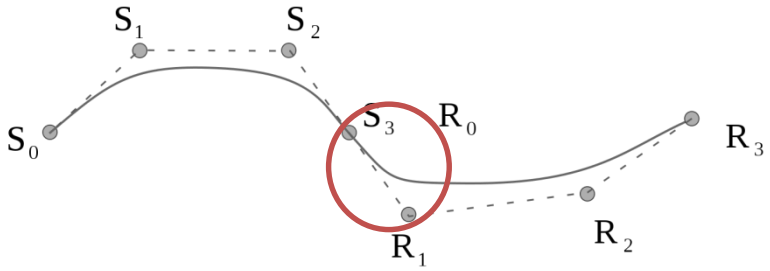


C0-Kontinuität



C1-Kontinuität

- C2-Kontinuität bei stückw. Bézierkurven nicht zwingend gegeben



- Gewünschte Eigenschaften *Interpolation*, *lokale Kontrolle* und *C2-Kontinuität* lassen sich für kubische Kurven nicht garantiert vereinbaren .

https://en.wikipedia.org/wiki/Smoothness

B-Spline-Kurven

- Durch Verzicht auf die Interpolation lässt sich lokale Kontrolle und C2-Kontinuität erreichen.
- Möglich durch Austausch der Basisfunktion zur B-Spline-Basisfunktion:

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t)$$

B-Spline Basisfunktion:

$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \leq t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{i,k}(t) = \frac{t - x_i}{x_{i+k} - x_i} N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(t)$$

Cox- de Boor-Rekursion

- Ordnung der Basisfunktion $k + 1$
- Polynom hat den Grad k.
- Knotenvektor aus aufsteigend sortierten Werten (bestimmt durch Anzahl der Stützpunkte n und Polynomgrad k).

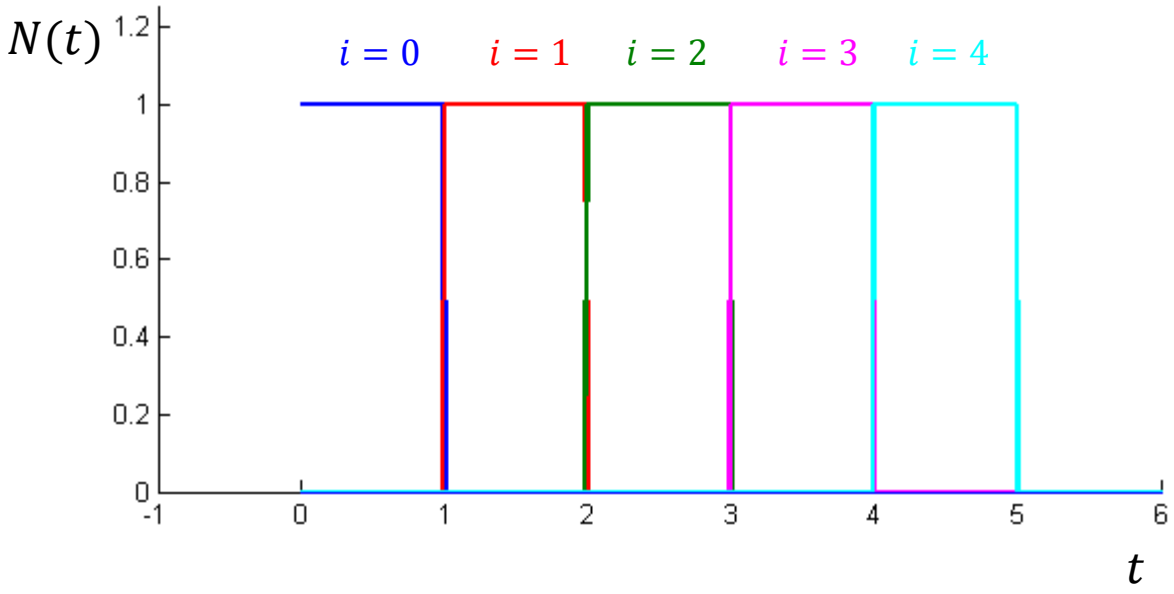
$$x_i : x_1 < \cdots < x_n < x_{n+1} < x_{n+k+1}$$

- Definitionsbereich von t frei wählbar
- Knotenvektor im Definitionsbereich frei wählbar (Kontrolle über Kurvenverlauf)

B-Spline-Kurven

- Beispiel: $k = 0, n = 5$ Stützpunkte → $n + k + 1 = 6$ Knoten, $i = [0 \dots 4]$

Plot des Kurvenabschnitts für Knotenvektor $[0\ 1\ 2\ 3\ 4\ 5]$ durch Einsetzen in die Cox-de-Boor-Rekursion, für $t = [0 \dots 5]$



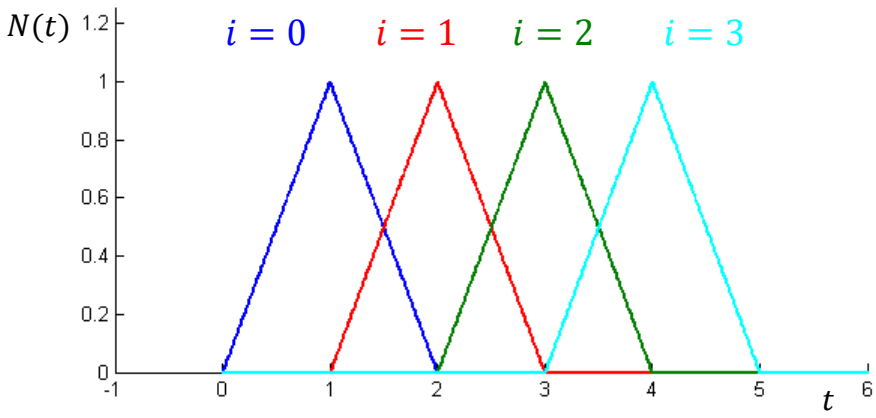
$\boldsymbol{k = 0}$

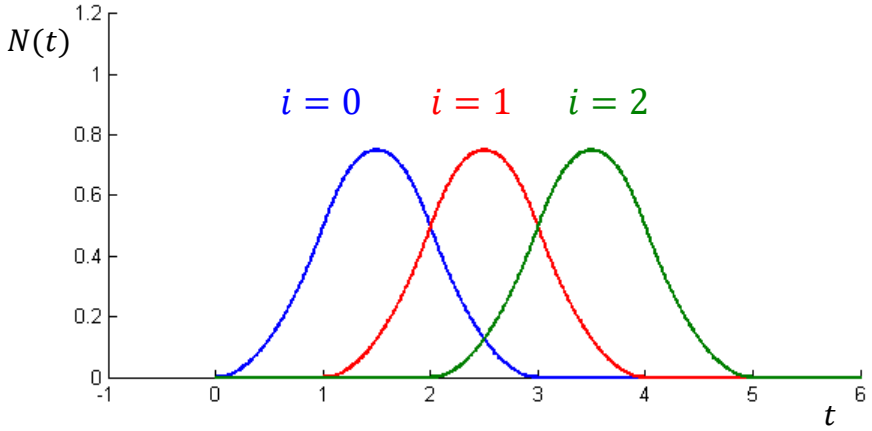
$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \le t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

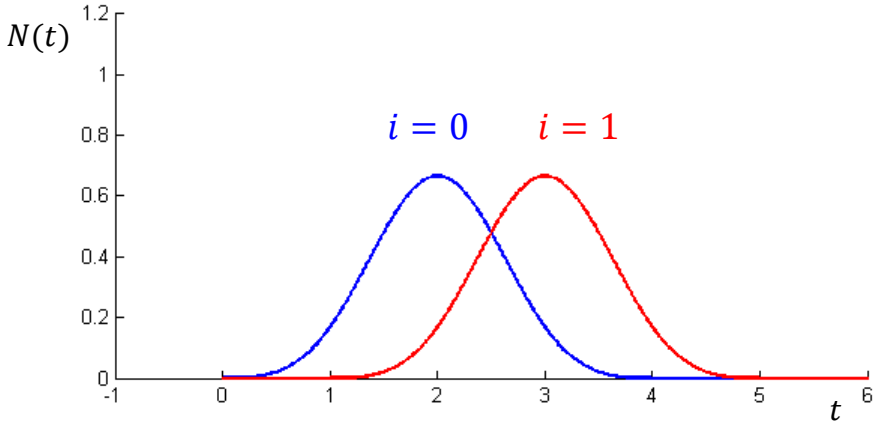
$i = 0$: $N_{0,0}(t) = 1$ für $0 \le t \le 1$

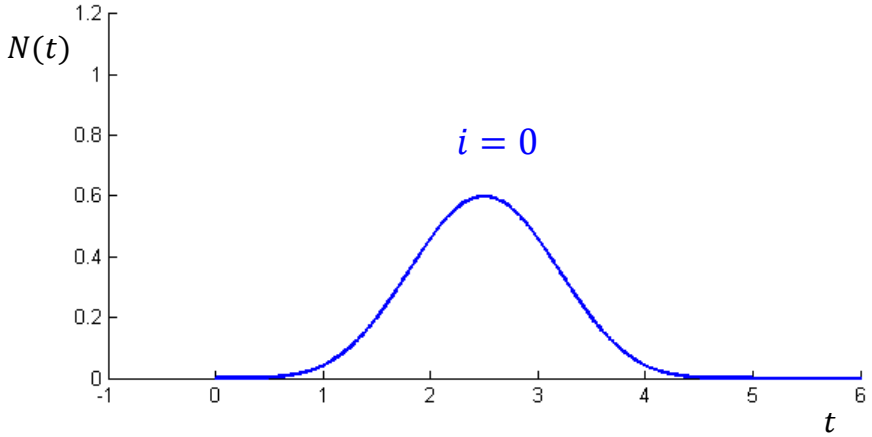
B-Spline-Kurven

- Beispiele für $k = 1, 2, 3, 4$ mit Knotenvektor [0 1 2 3 4 5]





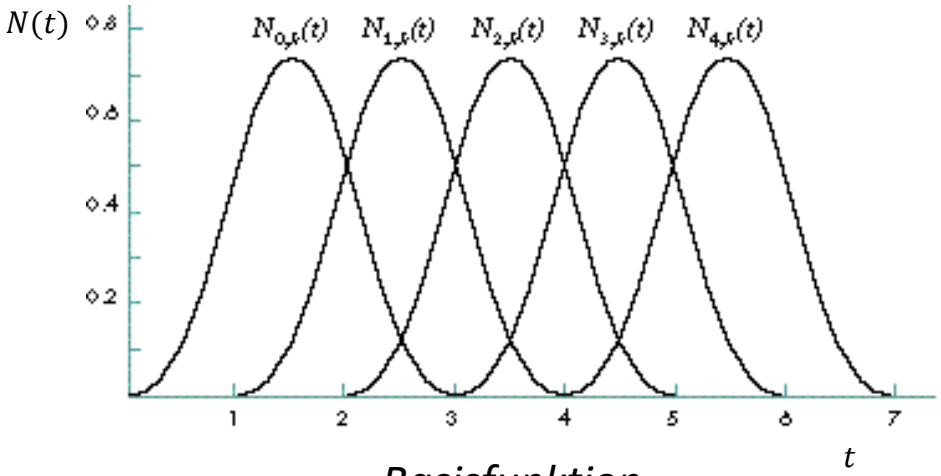




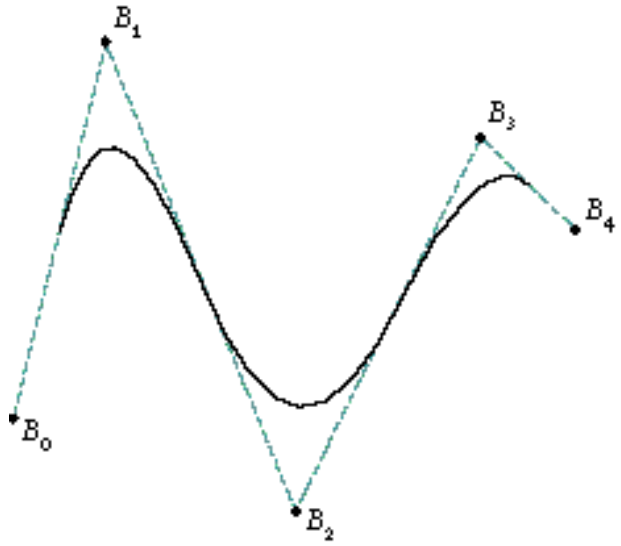
B-Spline-Kurven

- Beispiel:
  - 5 Stützpunkte b_0 … b_4, Grad $k = 2$ → Länge Knotenvektor $= 8$
  - mit $t = 0 \ldots 7$: <u>uniformer</u> Knotenvektor $[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7]$

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t)$$



Basisfunktion



2D B-Spline

http://www.mactech.com/articles/develop/issue_25/schneider.html

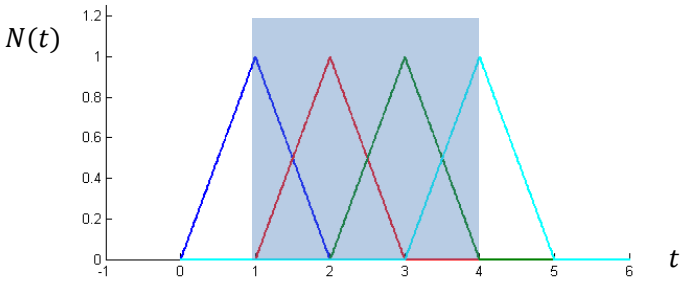
B-Spline-Kurven

$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \leq t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{i,k}(t) = \frac{t - x_i}{x_{i+k} - x_i} N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(t)$$

- Eigenschaften der Basisfunktion:
  - Eine Kurve der Ordnung $k + 1$ ist nur definiert wenn $k + 1$ Basisfunktionen ungleich 0 sind.

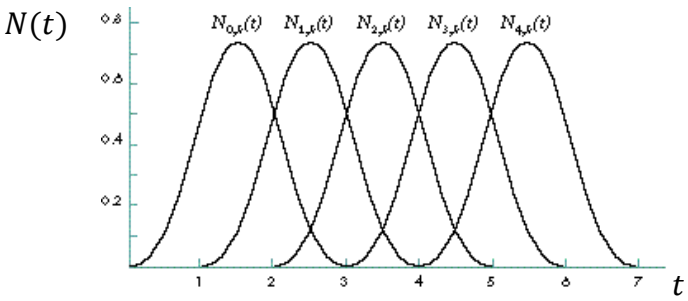
    Beispiel für $k = 1$:



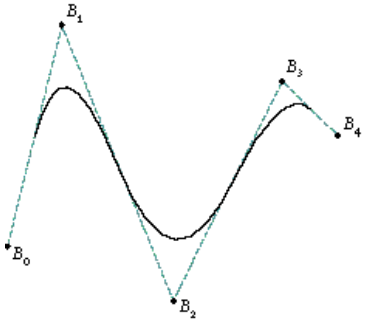
  - Für jedes t im Definitionsbereich addieren sich die Werte der Basisfunktionen zu 1.
  - Für jedes t im Definitionsbereich haben nie mehr als $k + 1$ Basisfunktionen einen Einfluss auf den Kurvenverlauf

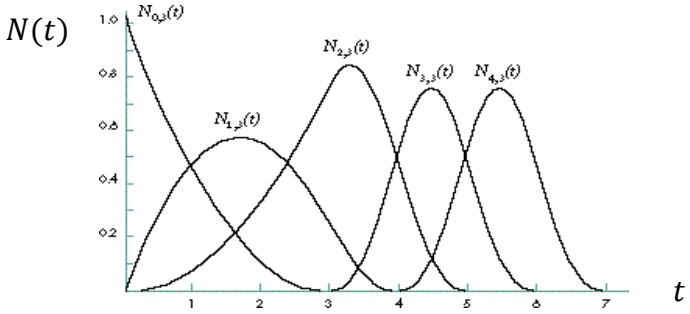
B-Spline-Kurven

- Bisher: periodische, uniforme Knotenvektoren
- Auswirkung offener uniformer Knotenvektoren:
  - k Wiederholungen des ersten Elements, Beispiel: $k = 2$

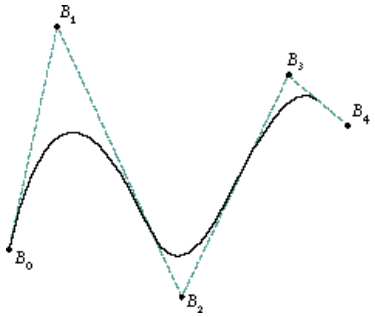


Knotenvektor = [0 1 2 3 4 5 6 7]





Knotenvektor = [0 0 0 3 4 5 6 7]

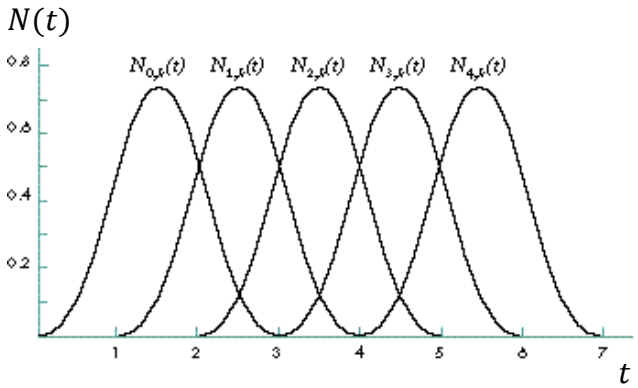


Geometrisch: Erster Knotenpunkt fällt mit erstem Stützpunkt zusammen (= Interpolation)

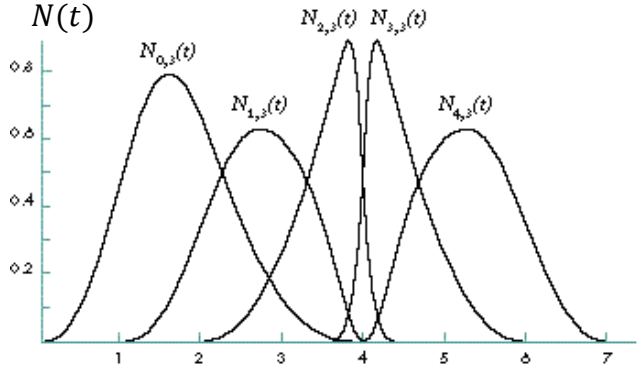
http://www.mactech.com/articles/develop/issue_25/schneider.html

B-Spline-Kurven

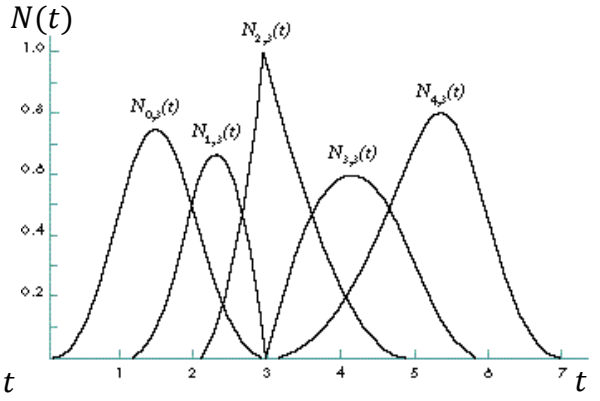
- Auswirkung nicht-uniformer Knotenvektoren:
  - Beispiele für $k = 2$



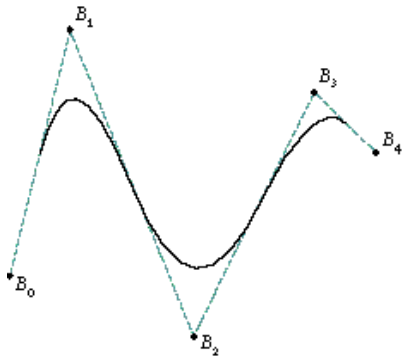
Knotenvektor = [0 1 2 3 4 5 6 7]

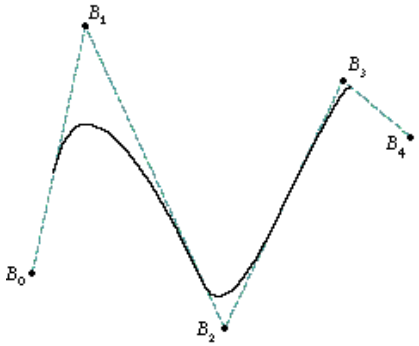


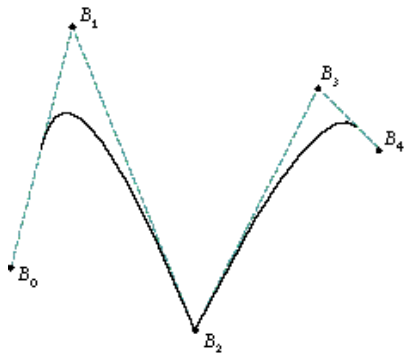
Knotenvektor = [0, 1, 2, 3.75, 4, 4.25, 6, 7]



Knotenvektor = [0, 1, 2, 3, 3, 5, 6, 7]





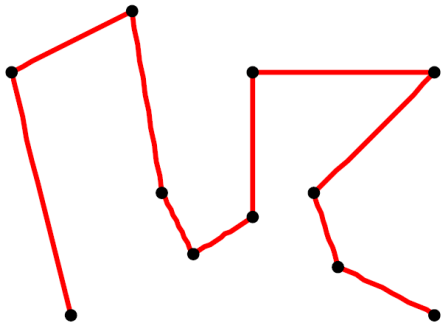


http://www.mactech.com/articles/develop/issue_25/schneider.html

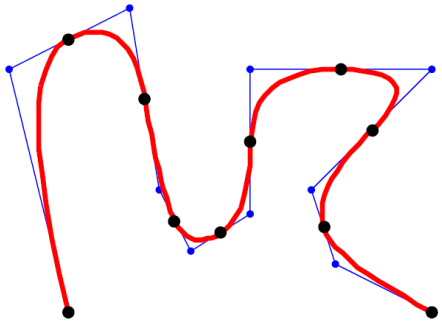
B-Spline-Kurven

- Auswirkung des B-Spline-Grades: Beispiele für Approximationen einer Kurve bei $k = 1, 2, 3, 4$

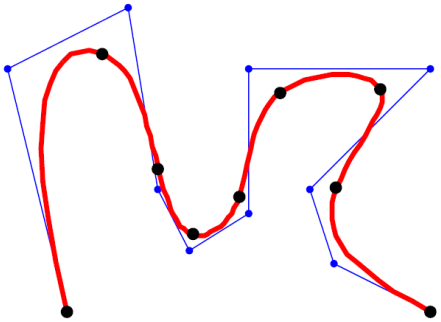
$k = 1$



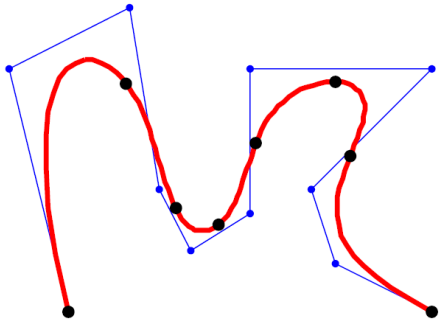
$k = 2$



$k = 3$



$k = 4$



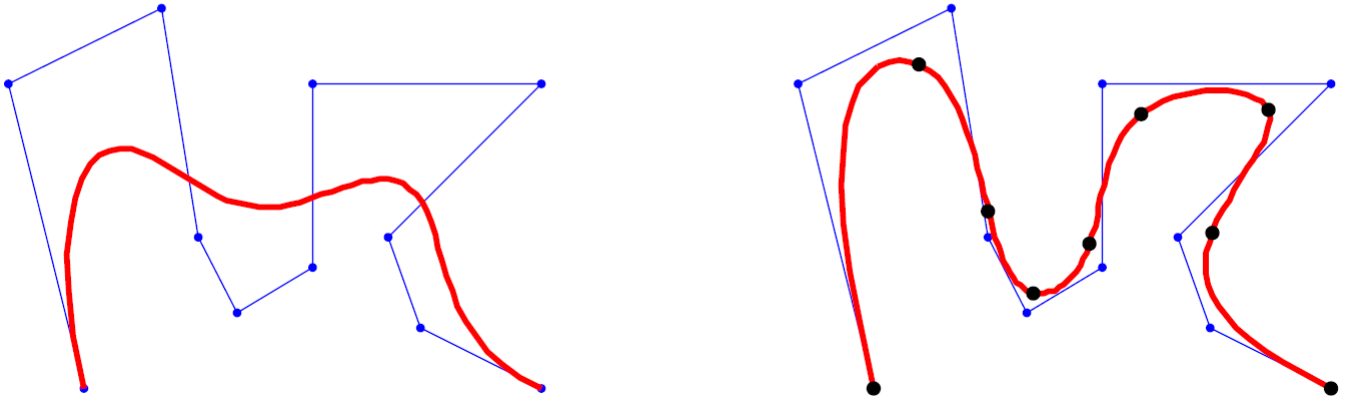
S. Terzi: „Splines in der Datenverarbeitung“, Seminararbeit Fachhochschule Aachen, 2012

B-Spline-Kurven

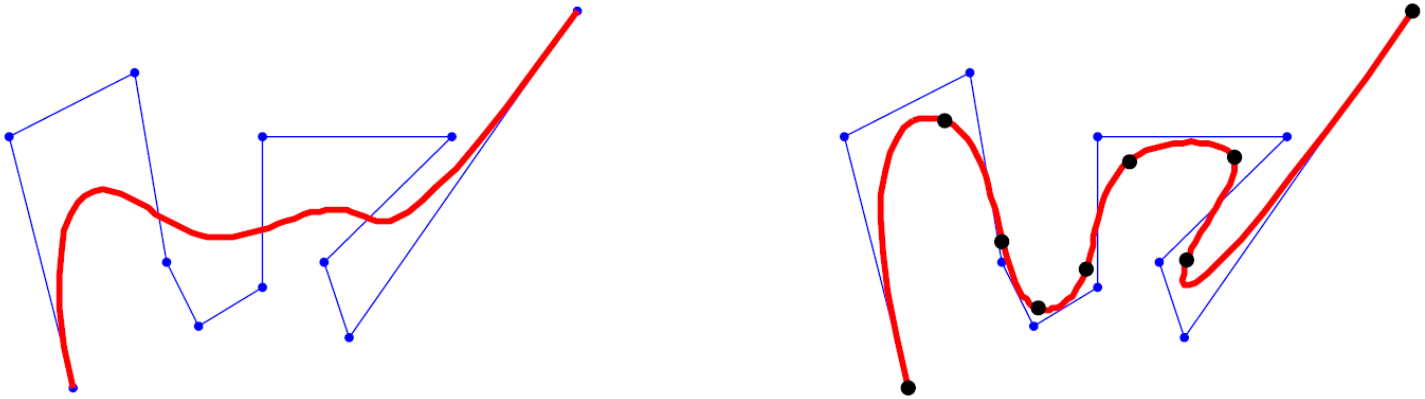
- Zusammenfassend: Definition eines B-Splines durch
  - Kontrollpolygon (= Stützpunkte)
  - Knotenvektor
  - Grad bzw. Ordnung der Kurvensegmente
- Eigenschaften:
  - Grad der Kurve wird nicht durch Anzahl der Stützpunkte angegeben
  - Maximale Ordnung der Kurve entspricht Anzahl der Stützpunkte
  - Kurve liegt innerhalb der konvexen Hülle der Stützpunkte
  - Invarianz gegenüber affinen Transformationen

Vergleich: Bézier vs. B-Splines

Bessere Modellierungseigenschaften der B−Splines:



Lokalitätseigenschaft der B−Splines:



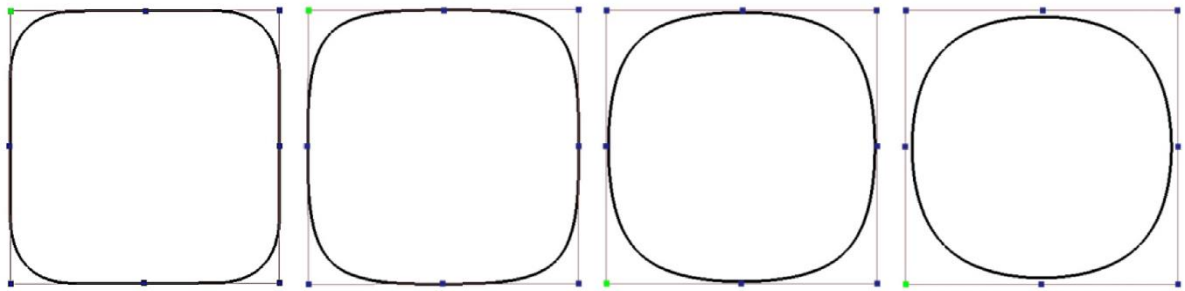
S. Terzi: „Splines in der Datenverarbeitung“, Seminararbeit Fachhochschule Aachen, 2012

B-Spline-Kurven

- Kontrolle über den Kurvenverlauf:
  - Änderung des Knotenvektors (periodisch uniform, offen uniform, nicht-uniform)
  - Änderung des Grades k
  - Änderung der Zahl/Position der Stützpunkte des Kontrollpolygons

  ➔ Deutlich mehr Flexibilität als Bézierkurven

- Limitationen
  - Können z.B. einen Kreis nicht exakt darstellen, nur approximieren



NURBS

- NURBS = Non-uniform rational B-Splines
  - Erweiterung der Basisfunktion auf rationale Funktion
  - Einführung eines Gewichtungsfaktors w für jeden Stützpunkt
- Kurve ist definiert durch:

$$p(t) = \frac{1}{\sum_{i=0}^{n-1} N_{i,k}(t) \cdot w_i} \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t) \cdot w_i$$

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot R_{i,k}(t)$$

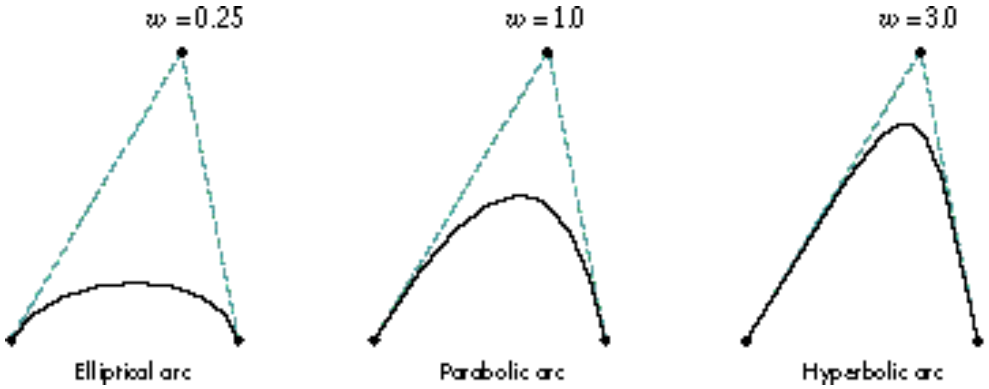
mit

$$R_{i,k}(t) = \frac{N_{i,k}(t) \cdot w_i}{\sum_{j=0}^{n-1} N_{j,k}(t) \cdot w_j}$$

NURBS Basisfunktion

NURBS

- Einfluss der Gewichte
  - Beispiel: 3 Stützpunkte, $w_0 = 1$, $w_2 = 1$, Knotenvektor $[0\ 0\ 0\ 1\ 1\ 1]$



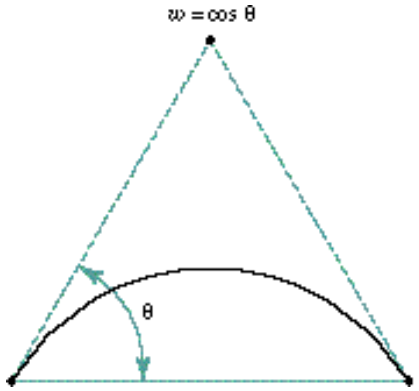
  - Erhöhung des Gewichts eines Stützpunktes erhöht dessen Einfluss

    → Kurve wird zum Stützpunkt hingezogen

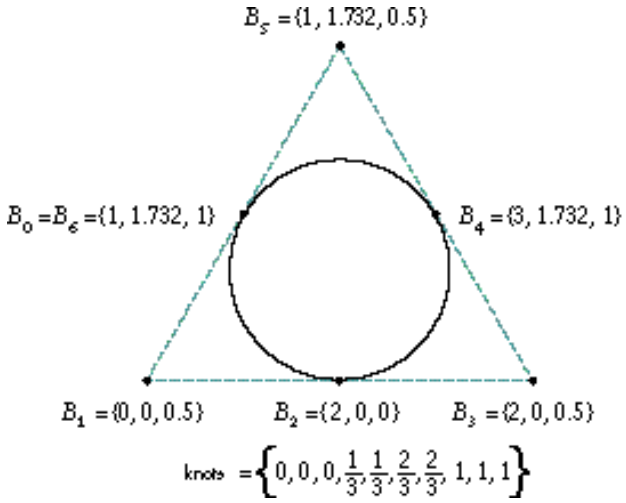
http://www.mactech.com/articles/develop/issue_25/schneider.html

NURBS

- Beispiel: Kreis aus NURBS durch Konstruktion von Kreisbögen
  - Kontrollpolygon gleichschenklig
  - Winkel θ so groß wie halbe Größe des Kreisbogens.
    z.B. Kreisbogen $120° \rightarrow \theta = 60°$
  - Gewicht des inneren Knotens $w_1 = \cos\theta$.
    z.B. $\theta = 60° \rightarrow w_1 = 0{,}5$
  - Knotenvektor = $[0\ 0\ 0\ 1\ 1\ 1]$



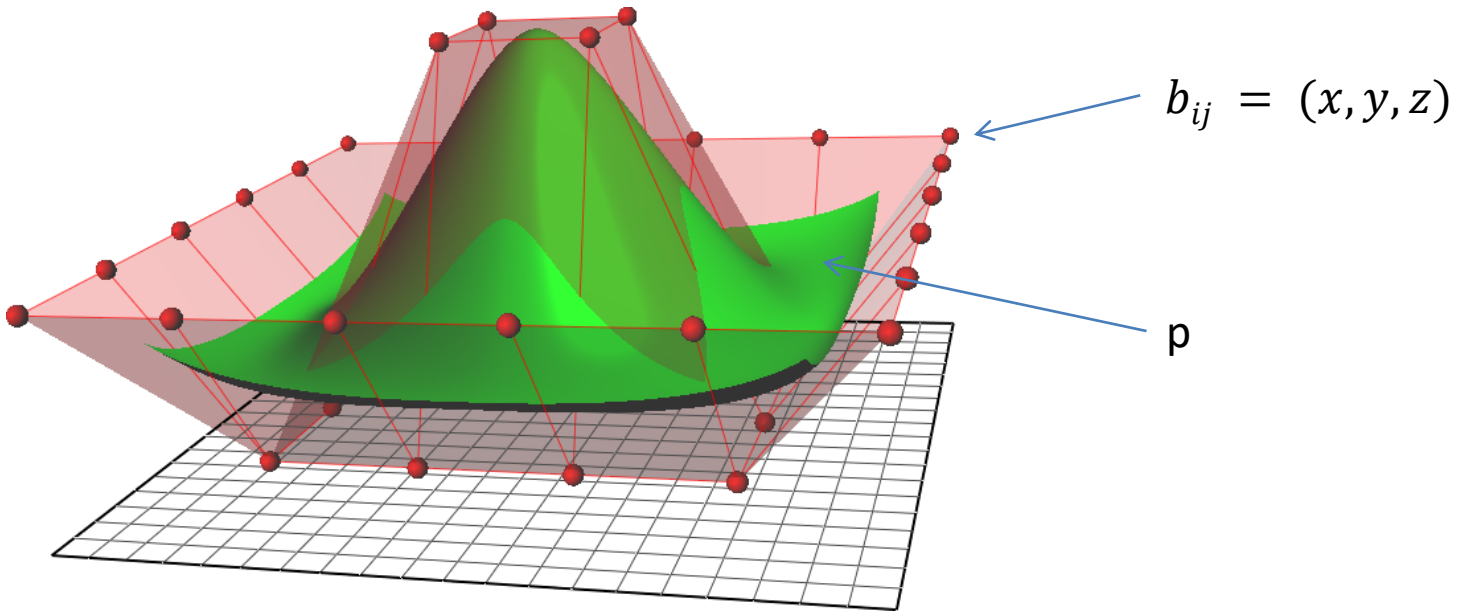
- Zusammengesetzt:



http://www.mactech.com/articles/develop/issue_25/schneider.html

B-Spline-/NURBS-Flächen

- Erweiterung der Definitionen auf NURBS-Flächen analog zu Bézierflächen.
  - Statt t nun zwei parametrische Richtungen: s, t
  - Stützpunkte in einem Gitter b_{ij} angeordnet.



https://de.wikipedia.org/wiki/Non-Uniform_Rational_B-Spline

Freiformkurven/-flächen in OpenGL

- „Evaluator“-Konzept für Bézierflächen
  - Definition eines eindimensionalen Evaluators: **`glMap1*()`**
  - Auswerten der Funktion an diskreten Stellen: **`glEvalCoord1()`**

- Eindimensionaler Evaluator:

Parameter target	Bedeutung
GL_MAP1_VERTEX_3	x, y, z Vertex Koordinaten
GL_MAP1_VERTEX_4	x, y, z, w Vertex Koordinaten
:	:

Definiert Intervall von *t: [t1 t2]*

Vektorlänge pro Stützpunkt

```
void glMap1fd(GLenum target, TYPE t1, TYPE t2, GLint stride,
              GLint order, const TYPE *points);
```

Ordnung des Polynoms
(= Anzahl der Stützpunkte)

Datenstruktur der Stützpunkte

Freiformkurven/-flächen in OpenGL

- Auswertung an diskreten Stellen (Vertexposition bestimmen):

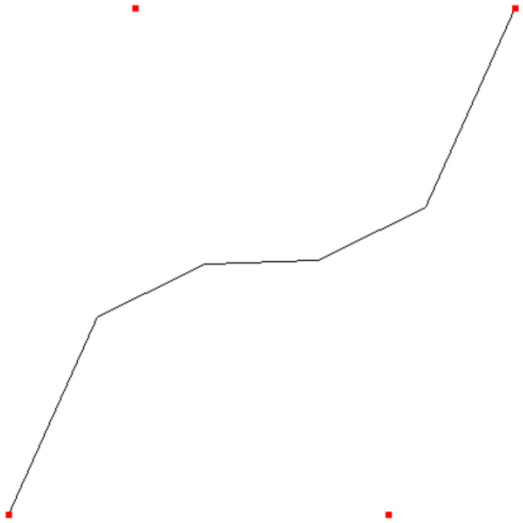
  `glEvalCoord1{fd} (TYPE t);`

- Beispiel:

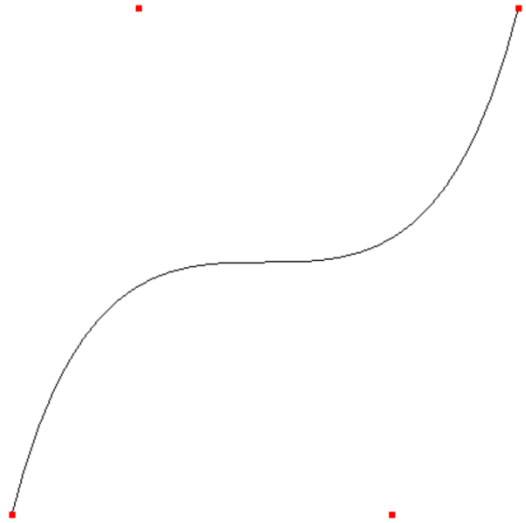
```
#define STEPS 5

GLfloat ctrlpts[4][3] = {{-4.0,-4.0,0.0}, {-2.0,4.0,0.0}, {2.0,-4.0,0.0}, {4.0,4.0,0.0}};

void init(void){
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpts[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
}

void display(void) {
    int i;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
        for (i = 0; i <= STEPS; i++)
        glEvalCoord1f((GLfloat) i / (GLfloat) STEPS);
    glEnd();
}
```

Freiformkurven/-flächen in OpenGL

- Beispiel



STEPS = 5



STEPS = 30

- Analog: zwei-dimensionale Evaluatoren für Flächen

Freiformkurven/-flächen in OpenGL

- GLU NURBS interface: high-level Interface zum Evaluator-Konzept
  - Erzeugung eines NURBS Objektes

    ```
    theNurb = gluNewNurbsRenderer();
    ```

  - Einstellung der NURBS Rendering Eigenschaften

    ```
    gluNurbsProperty();
    ```

  - Kurve zeichnen

    ```
    gluBeginCurve(theNurb);
        gluNurbsCurve(theNurb, knotCount, knots, stride,
                      &ctrlpts, order, type);
    gluEndSurface(theNurb);
    ```

    Anzahl der Knoten

    Knotenvektor

  - Flächen äquivalent:
    - `gluBeginSurface(theNurb);`
    - `gluNurbsSurface(…);`

ZUSAMMENFASSUNG

Zusammenfassung

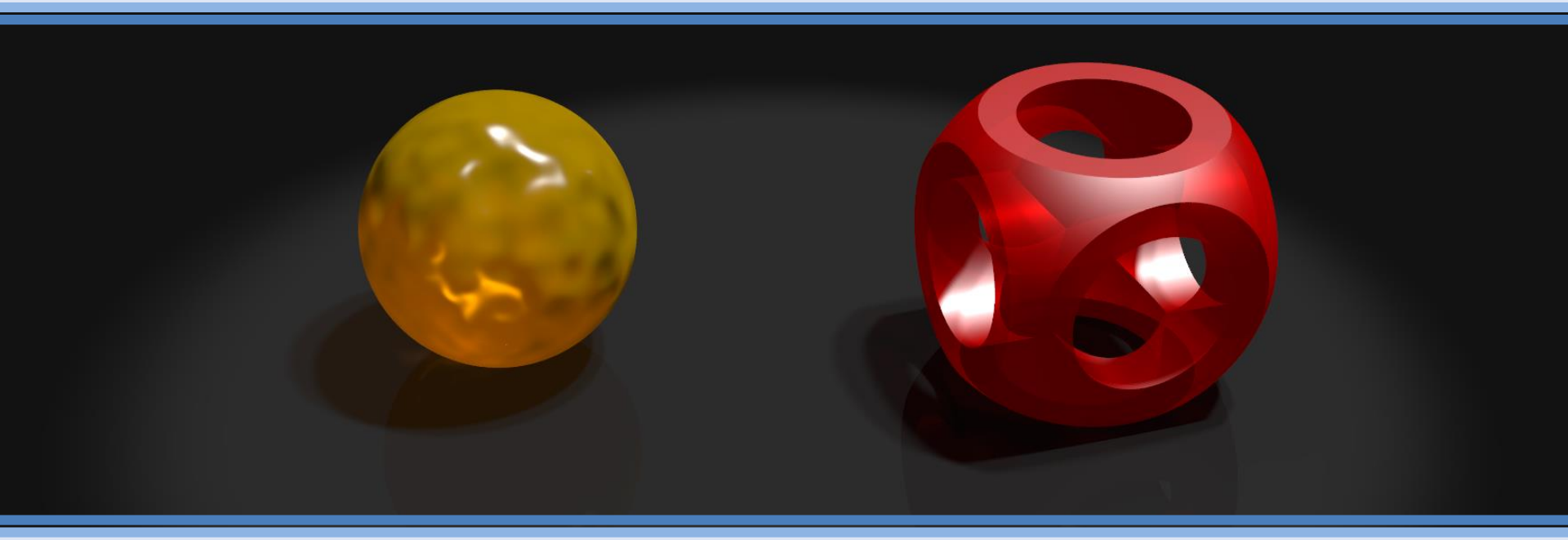
- Geometrische Beschreibung von Objekten durch Zusammensetzen von Grundobjekten, meist planare Polygone
  - Datenstrukturen: Knoten-/Kantenliste, …
- Polygonisierung/ Triangulation zum Erzeugen von Polygonnetzen
  - Daten aus Isoflächen, Punktwolken etc.
  - Z.B. per Marching Cubes Algorithmus, Triangulation von Polygonen, Triangulation von Punktwolken
  - Gewisse Eigenschaften sollten eingehalten werden, z.B. keine degenerierten oder sehr flache Dreiecke → Delaunay Triangulation
  - Nachbearbeitungsschritte: Meshverfeinerung, Meshdezimierung, Meshglättung
- Darstellung gekrümmter Kurven/Flächen über parametrische Funktionen
  - Bézierkurven/-flächen, stückweise Bézierkurven/-flächen
  - B-Splines
  - NURBS

Übungsfragen Kapitel 3

- Was ist Front- und Backface-Culling? Wie wird bestimmt ob ein Betrachter auf Vorder- oder Rückseite eines Polygons blickt?
- Was ist adaptives Meshing? Welchen Vorteil bietet es?
- Wann ist ein Polygon a) einfach? b) monoton bzgl. einer Achse? c) planar, d) konvex
- Beschreiben Sie eine Möglichkeit zur Generierung eines Polygonnetzes aus Punktwolken
- Wann erfüllt ein Polygonmesh die Delaunay-Eigenschaft?
- Erläutern Sie ein Verfahren zur Meshglättung
- Was ist ein offener B-Spline?
- Wie kann eine B-Spline-Kurve einen Stützpunkt interpolieren?

Computergrafik

T. Hopp



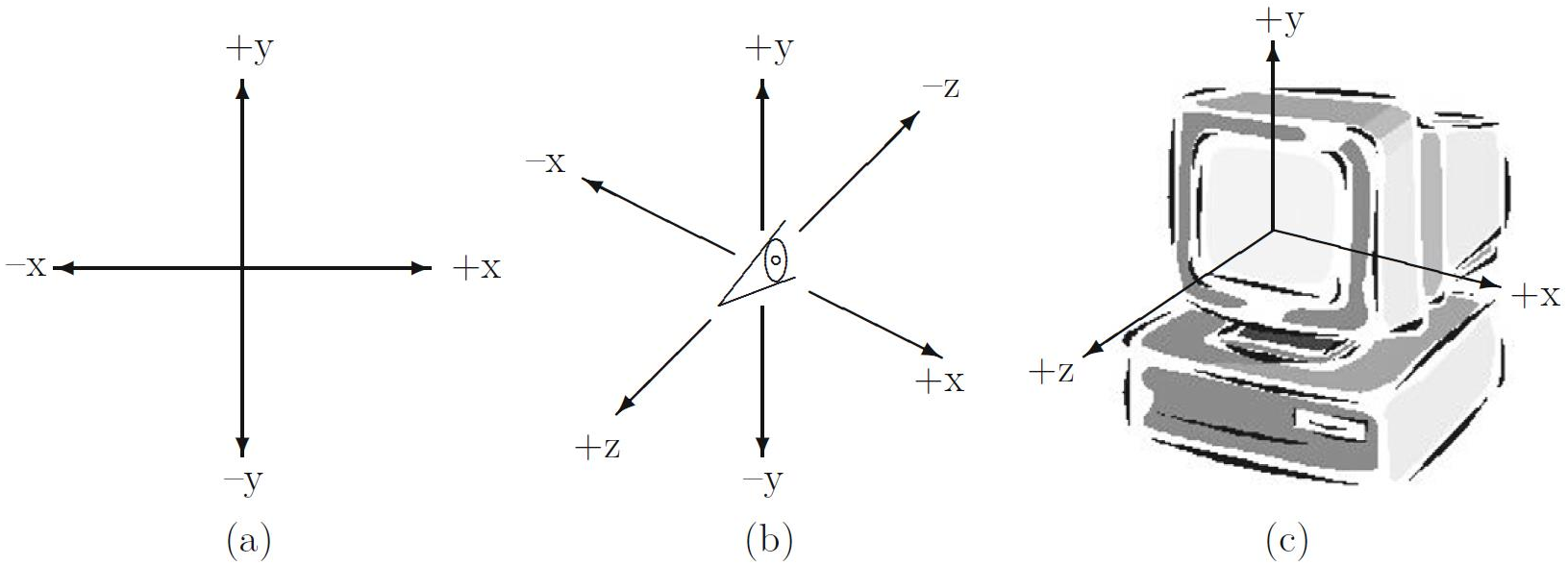
Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. **Koordinatensysteme und Transformationen**
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

4.1. KOORDINATENSYSTEME

(OpenGL-) Koordinatensystem

- In der Regel 3D Euklidisches Koordinatensystem mit x-, y-, z-Achse
- Betrachter (=Augpunkt) standardmäßig im Ursprung ($x = 0, y = 0, z = 0$)



OpenGL Koordinatensystem aus
(a) Sicht des Augpunktes, (b) Sicht eines externen Beobachters
(c) mit Blick auf den Bildschirm

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Koordinatensysteme in der CG: Unterscheidung

- In der Computergrafik gibt es nicht nur ein Koordinatensystem. Es werden i.d.R. die folgenden unterschieden:

- **Objektkoordinaten:**

  Lokale Koordinatensysteme für 3D Objekte
- **Weltkoordinatensystem:**

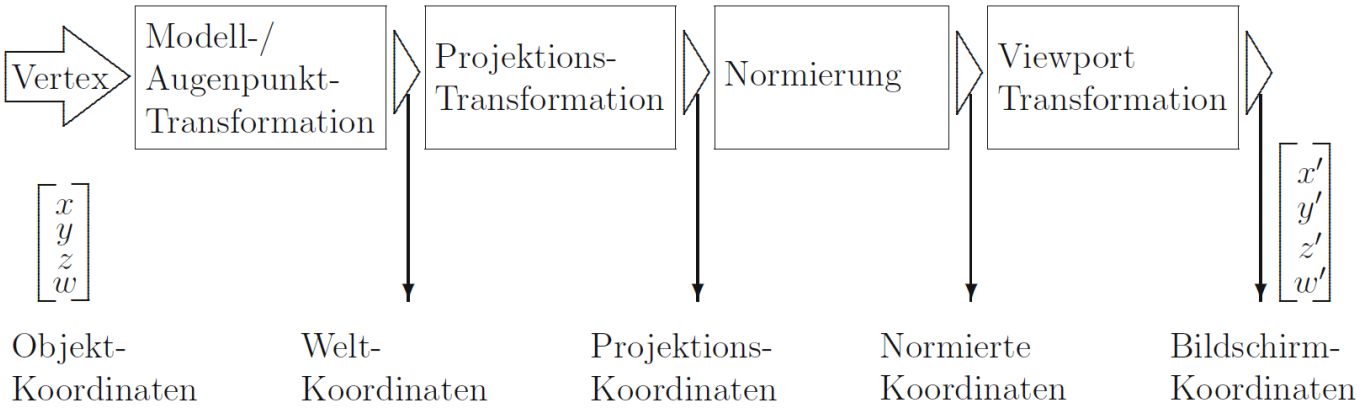
  Koordinaten die für die gesamte Szene gelten
- **Projektionskoordinaten:**

  Koordinaten nach perspektivischer bzw. orthogonaler Projektion
- **Normierte Koordinaten:**

  Auf vorgegebenen Wertebereich beschränkte Koordinaten, die nach Division der Projektionskoordinaten mit dem inversen Streckungsfaktor *w* entstehen.
- **Bildschirmkoordinaten:**

  Koordinaten, die Szene in der gewählten Fenstergröße darstellen

Transformationskette



- **Modell-/Augenpunkttransformation**: Positionierung von Objekten in der Szene
  (= ModelView Matrix)
- **Projektionstransformation:** Definition des sichtbaren Volumens, z.B. Blickwinkel
  (= Frustum)
- **Normierung:** Transformation der Koordinaten auf Intervall [-*w*, +*w*] und Division durch inversen Streckungsfaktor *w*
  (= Normalized Device Coordinates)
- **Viewport-Transformation:** Positionierung der Vertices im Fenster

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

4.2. HOMOGENE KOORDINATEN

Transformationen in 3D

- Ein Punkt im 3-dimensionalen Euklidischen Raum kann durch drei Koordinaten x, y, z beschrieben werden
  - Darstellung in Vektorform: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ bzw. (x, y, z) in transponierter Schreibweise
- Eine Transformation T des Ortsvektors des Punktes $v = (x, y, z)$ kann durch Multiplikation mit einer 3 x 3 Transformationsmatrix M durchgeführt werden: $v' = Mv$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\begin{aligned} x' &= m_{11} \cdot x + m_{12} \cdot y + m_{13} \cdot z \\ y' &= m_{21} \cdot x + m_{22} \cdot y + m_{23} \cdot z \\ z' &= m_{31} \cdot x + m_{32} \cdot y + m_{33} \cdot z \end{aligned}$$

- Translation kann durch die Transformationsmatrix nicht abgebildet werden!

Homogene Koordinaten

- Beschreibung eines Punktes im 3D Raum durch vier Komponenten:

$$v = (x_h, y_h, z_h, w)$$

„inverser Streckungsfaktor“

- Euklidische Koordinate (x, y, z) berechnet sich als

$$x = \frac{x_h}{w}, \qquad y = \frac{y_h}{w}, \qquad z = \frac{z_h}{w}$$

- Werte für *w:*
  - $w = 1$: homogene Koordinaten = Euklidische Koordinaten
  - $w < 1,\ w > 1$: Streckung/Stauchung um Faktor $1/w$
  - $w = 0$ Division durch 0 → Richtungsvektoren
- Transformation mit homogenen Koordinaten

Translation

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Transformation mit homogenen Koordinaten

- Allgemein:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

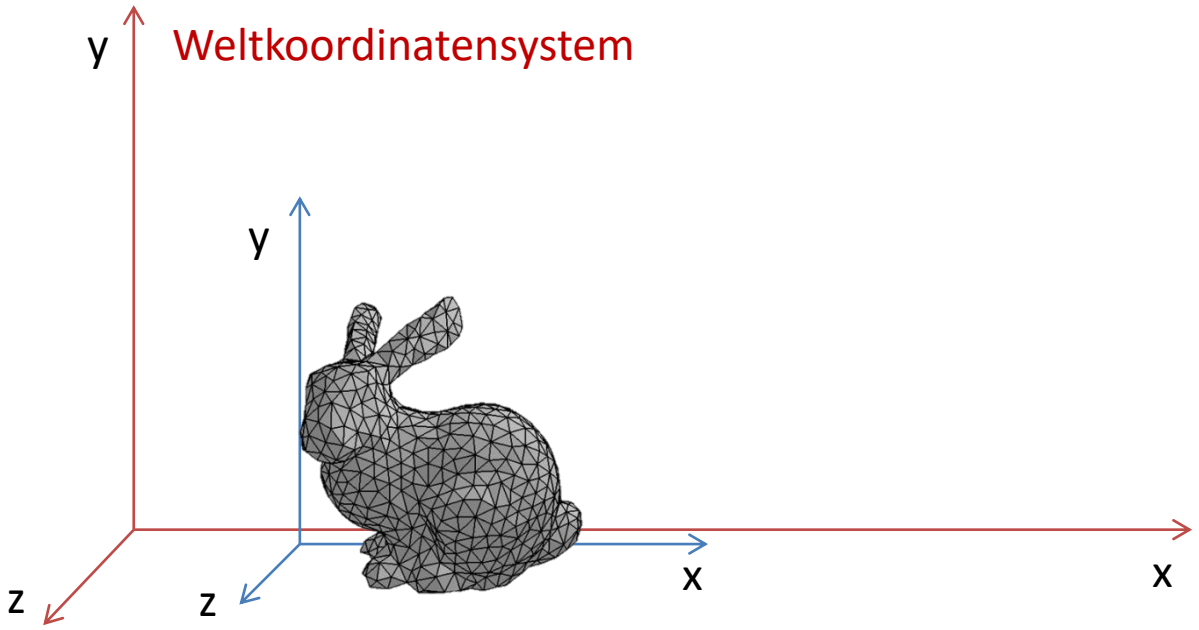
$$v' \quad = \quad \boldsymbol{M} \quad v$$

- Alle Transformationen der Transformationskette arbeiten mit homogenen Koordinaten → Effiziente Abbildung der Operationen in der Hardware
- Ausführung mehrerer Transformationen hintereinander
  - Möglichkeit 1: $v' = \left(\ldots \cdot \mathbf{M}_2 \cdot (\mathbf{M}_1 \cdot v)\right)$
  - Möglichkeit 2: $v' = (\ldots \cdot \mathbf{M}_2 \cdot \mathbf{M}_1) \cdot v$
- OpenGL nutzt Möglichkeit 2, da effizienter.
- Matrizen-Operationen in OpenGL:

```
glLoadMatrixf(const Glfloat *M);    // laden einer Matrix
glLoadIdentity(Glvoid);             // Identitätsmatrix
glMultMatrixf(const Glfloat *M);    // Multipl. mit Matrix im Speicher
```

4.3. MODELL- UND AUGPUNKTTRANSFORMATIONEN

Modell-Transformationen



- Definition von Objekten i.d.R. in lokalem Koordinatensystem
  - Feste Kopplung 3D-Objekt an lokales Koordinatensystem
- Positionierung in der 3D Szene: lokales Koordinatensystem wird im Weltkoordinatensystem „bewegt“:
  - Translation
  - Drehung
  - Skalierung

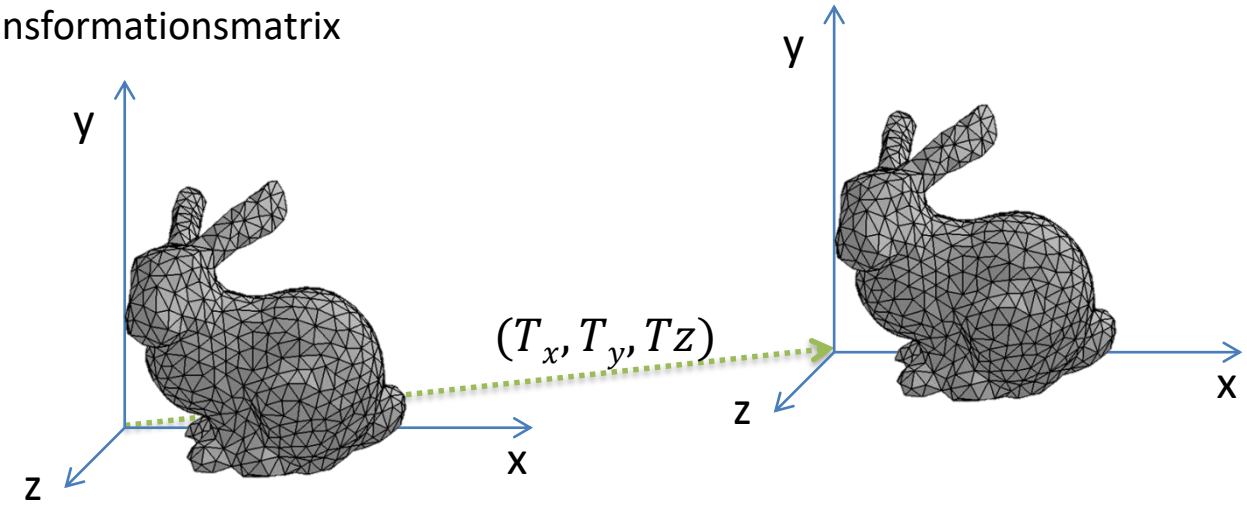
  = Affine Transformationen

Translation

- Translation eines Punktes um (T_x, T_y, T_z)

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- In OpenGL: **`glTranslatef(T_x,T_y,T_z)`**
  - Definition einer 4 x 4 Translationsmatrix anhand T_x, T_y, T_z
  - State Machine (!): Multiplikation mit der aktuell im Speicher vorliegenden Transformationsmatrix



Rotation

- Rotation um die x-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

```
glRotatef(α,1,0,0);
```

- Rotation um die y-Achse:

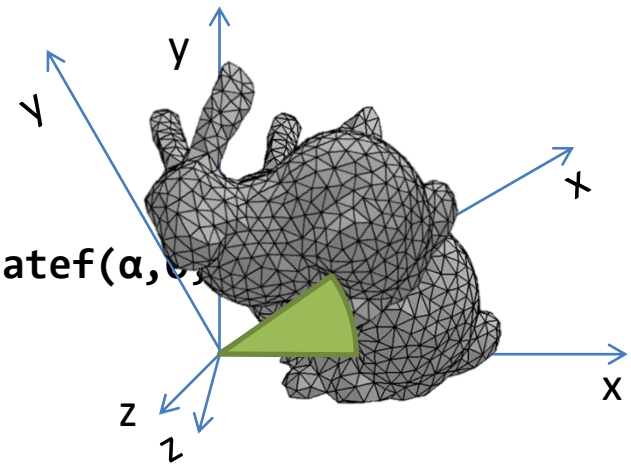
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

```
glRotatef(α,0,1,0);
```

- Rotation um die z-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

```
glRotatef(α,0
```



Rotation

- Rotation eines Punktes v um einen Richtungsvektor $r = (R_x, R_y, R_z)$:

$$v' = \boldsymbol{M}\, v$$

$$M = \begin{pmatrix} {R_x}^2(1-\cos\alpha)\cos\alpha & R_xR_y(1-\cos\alpha) - R_z\sin\alpha & R_xR_z(1-\cos\alpha) + R_y\sin\alpha & 0 \\ R_yR_x(1-\cos\alpha) + R_z\sin\alpha & {R_y}^2(1-\cos\alpha)\cos\alpha & R_yR_z(1-\cos\alpha) - R_x\sin\alpha & 0 \\ R_zR_x(1-\cos\alpha) - R_y\sin\alpha & R_zR_y(1-\cos\alpha) + R_z\sin\alpha & {R_z}^2(1-\cos\alpha)\cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- In OpenGL:

```
glRotatef(α,R_x,R_y,R_z);
```

Länge des Richtungsvektors = 1

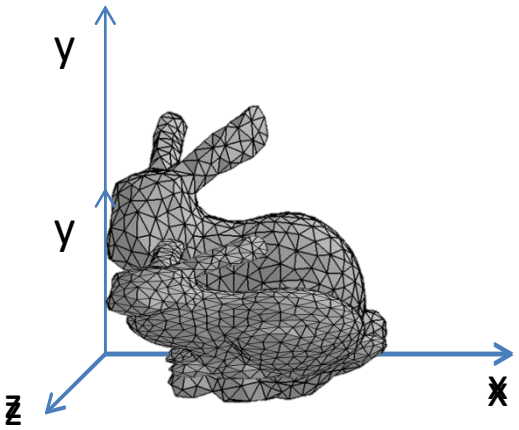
Skalierung

- Skalierungsmatrix: Skalierung um Faktoren (S_x, S_y, S_z)

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- In OpenGL:

```
glScalef(S_x,S_y,S_z);
```



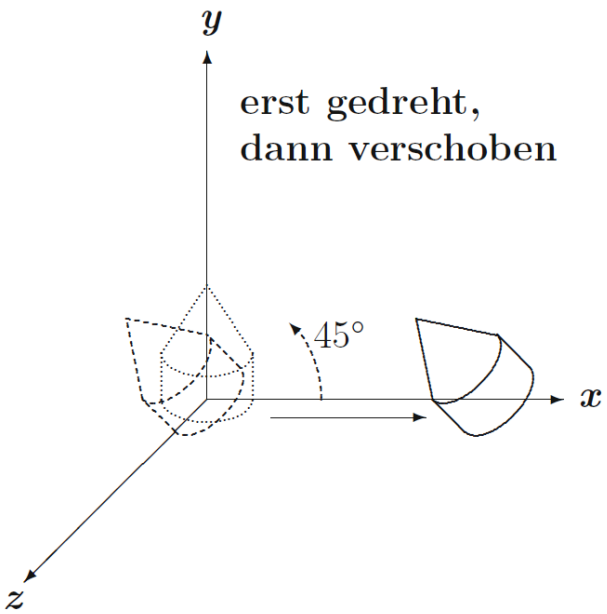
Beispiel für $S_x = 1, S_y = 0.5, S_z = 1$

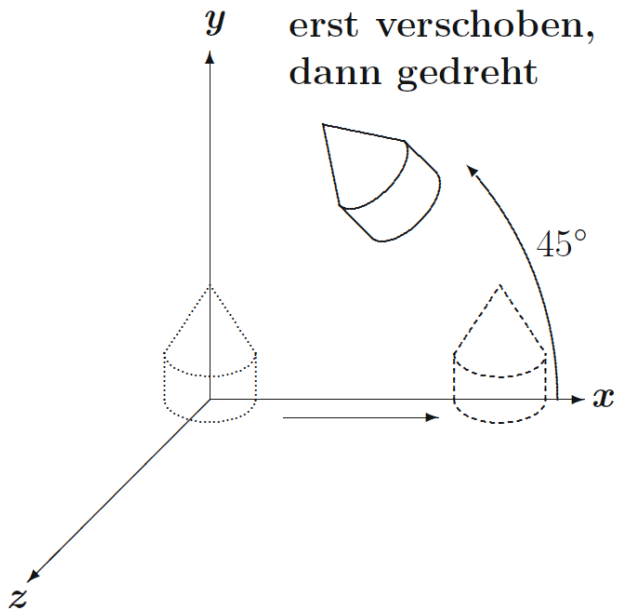
Skalierung

- $S_x = S_y = S_z$ uniforme Skalierung
- $S < 0$ Spiegelung an der jeweiligen Achse
- $S = 0$ unzulässiger Wert
- $|S| = 1$ Dimension unverändert (aber evtl. Spiegelung wenn $S = -1$)
- $0 < |S| < 1$ Stauchung, Dimension wird verkleinert
- $|S| > 1$ Streckung, Dimension wird vergrößert

Reihenfolge der Transformationen

- Reihenfolge der Transformationen ist wichtig, da sie nicht unabhängig von einander sind
- Unterscheidung zweier Denkweisen:
  1. Transformation eines Objektes im Weltkoordinatensystem

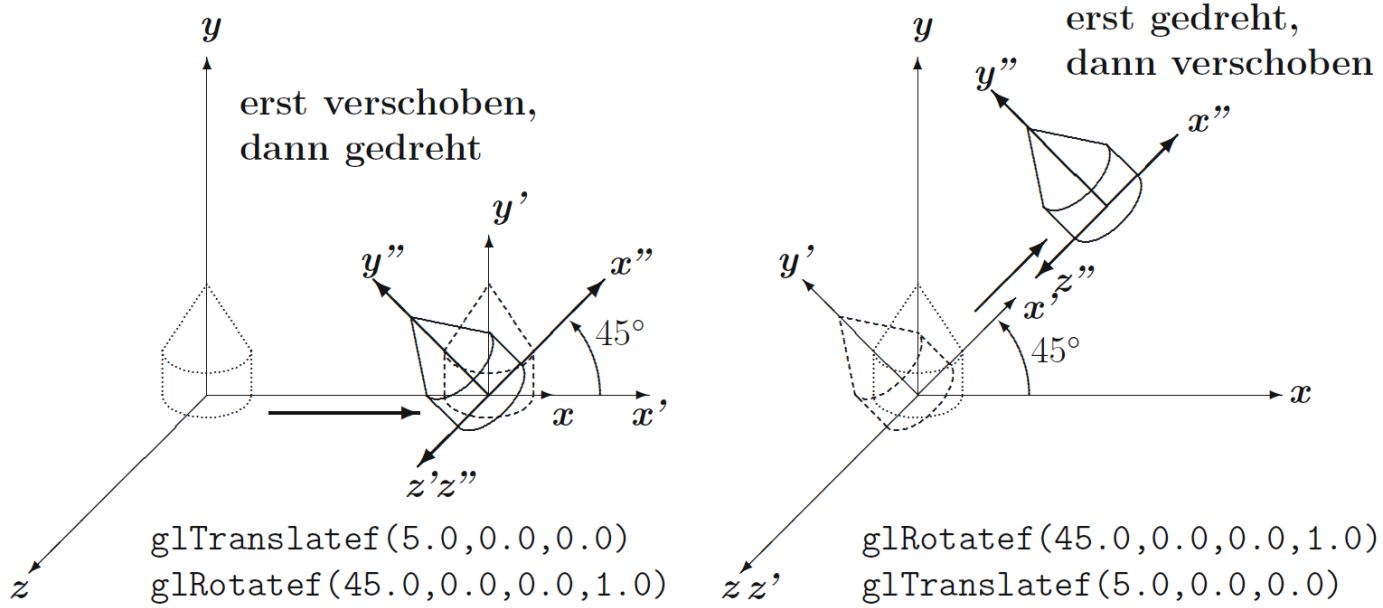




A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Reihenfolge der Transformationen

2. Transformation des lokalen Koordinatensystem im Weltkoordinatensystem



- Verwendung 2. Variante im Programmcode: Verkettung durch Multiplikation auf vorhandene Matrix (=Rechtsmultiplikation) → folgt der Denkweise „Transformation des lokales Koordinatensystem“

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Reihenfolge der Transformationen

- Implementierungsbeispiel:

```
glLoadIdentity();
glTranslatef(…);
glRotatef(…);
glBegin(GL_TRIANGLES);
glVertexfv(v);
:
glEnd();
```

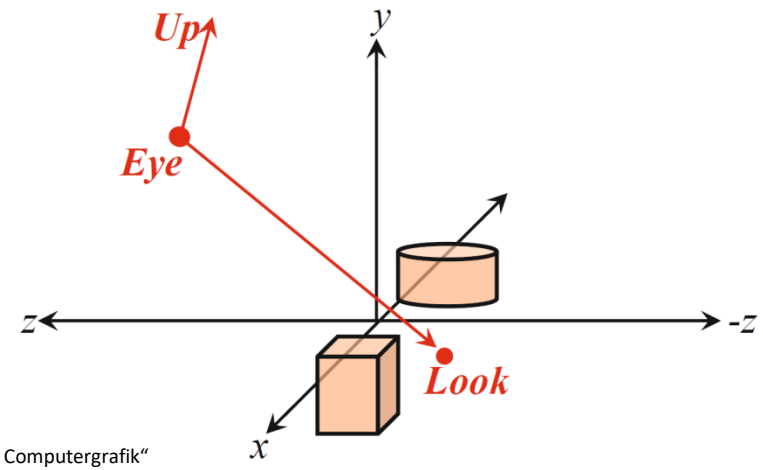
1. Laden der Identitätsmatrix $\boldsymbol{I}$ zur Initialisierung
2. Multiplikation der Translationsmatrix $\boldsymbol{T}$ von rechts auf $\boldsymbol{I}$: $\boldsymbol{I} \cdot \boldsymbol{T} = \boldsymbol{T}$
3. Multiplikation der Rotationsmatrix $\boldsymbol{R}$ von rechts auf $\boldsymbol{T}$: $\boldsymbol{T} \cdot \boldsymbol{R}$
4. Multiplikation der Vertices v von rechts auf $\boldsymbol{TR}$: $\boldsymbol{T} \cdot \boldsymbol{R} \cdot v$

   Dies entspricht: $\boldsymbol{T}(\boldsymbol{R}v) \Rightarrow v' = \boldsymbol{R}v, v'' = \boldsymbol{T}v'$

Augpunkttransformationen

- Ändern der Position und der Blickrichtung des Augpunktes
  - = Positionierung der Kamera
  - In OpenGL standardm. in Punkt $(0,0,0)$, Blick entlang der negativen z-Achse
- Gleichbedeutend mit Verschiebung der Szene (Modelltransformation)
- Zusammenfassung in einer Modelview-Matrix
- Augpunkttransformationen immer vor allen Transformationen ausführen!
- In OpenGL:

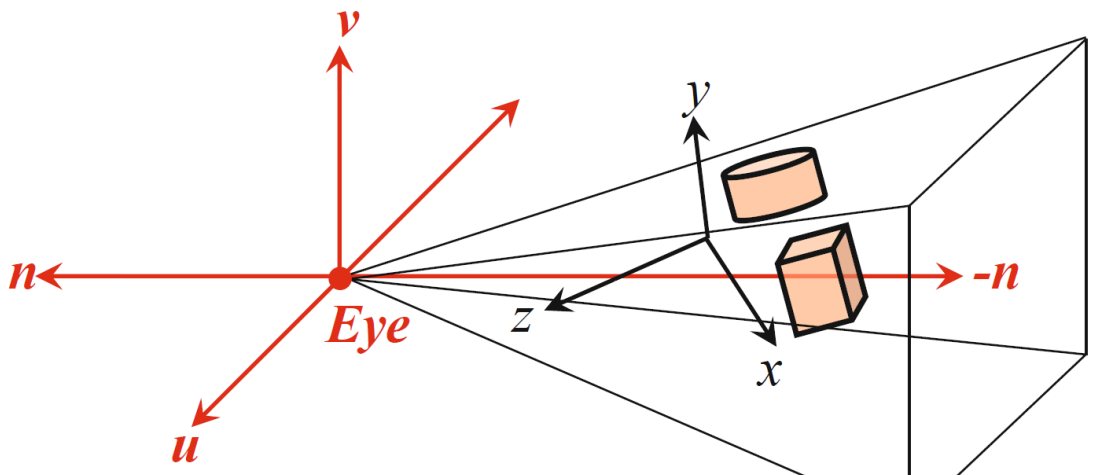
```
gluLookAt(Eye.x, Eye.y, Eye.z, Look.x, Look.y, Look.z, Up.x, Up.y, Up.z);
```



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Augpunkttransformationen

- **`gluLookAt`** Transformation ist zweiteilig:
  1. Drehung im Raum so dass Sichtachse n der z-Achse entspricht und gleichzeitig der Vektor nach oben (v, $\boldsymbol{Up}$) parallel zur y-Achse ausgerichtet ist
  2. Verschiebung des Ortsvektors $\boldsymbol{Eye}$ in den Ursprung $(0,0,0)$

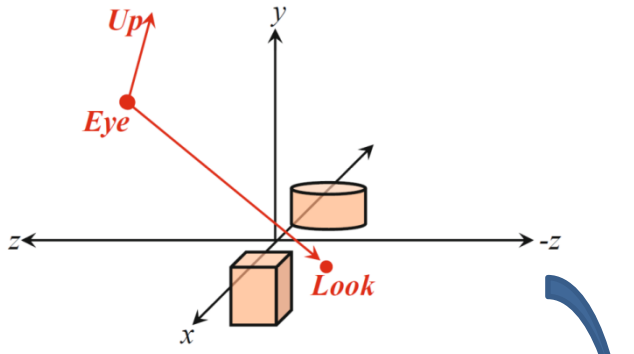


- Randbedingungen:
  - $\boldsymbol{Eye}$ und $\boldsymbol{Look}$ dürfen nicht identisch sein.
  - $\boldsymbol{Up}$ darf nicht parallel zu n gewählt werden.

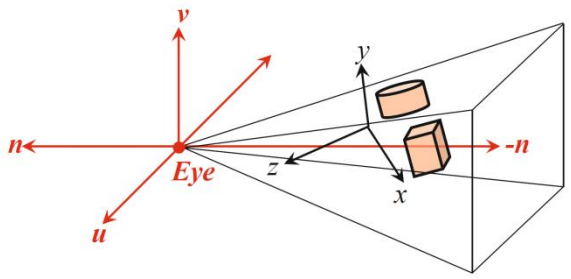
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Augpunkttransformationen

1. Berechnung der Rotationsmatrix:
   - $-n = \boldsymbol{Eye} - \boldsymbol{Look}$
   - $u, v \perp n$, $u \perp v$, v muss nach oben zeigen
     a.) $u = \boldsymbol{Up} \times n$
     b.) $v = n \times u$
   - Normierung der Länge von n, u, v auf 1.



- Rotationsmatrix ergibt sich als: $\mathbf{M}_R = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$



Augpunkttransformationen

2. Berechnung der Verschiebung:

- Einfache Translation um $-\boldsymbol{Eye}$ reicht nicht aus, da i.d.R. das Koordinatensystem gedreht wurde.
- Multiplikation der gesamten Augpunkt-Transformationsmatrix $\boldsymbol{M_{RT}}$ mit $\boldsymbol{Eye}$, gleichsetzen mit gewünschtem Ursprung in homogenen Koordinaten $(0,0,0,1)$

$$\begin{pmatrix} u_x & u_y & u_z & t_x \\ v_x & v_y & v_z & t_y \\ n_x & n_y & n_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Eye.x \\ Eye.y \\ Eye.z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} t_x = -u \cdot Eye \\ t_y = -v \cdot Eye \\ t_z = -n \cdot Eye \end{pmatrix}$$