Chapter 1

Three-Dimensional Object Representations

1.1 Overview

The main goal of three-dimensional computer graphics is to generate two-dimensional images of a scene or of an object based on a a description or a model.

The internal representation of an object depends on several implications:

- The object may be a real object or it exists only as a computer representation
- The manufacturing of the object is bound closely to the visualization:
  - Interactive CAD systems
  - Modeling and visualization as a tool during design and manufacturing
  - More than just 2-D output possible!

1.1 Overview

Implications (continued)

- The precision of the internal computer representation depends on the application. For example, an exact description of the geometry and shape in CAD applications vs. an approximation sufficient for rendering of the object.
- For interactive applications, the object may be described by several internal representations. These representations may be generated in advance or on-the-fly.
  - Level-of-detail (LOD) techniques

1.1 Overview

The modeling and representation of an object involves the following in particular:

- Generation of 3-D geometry data

  CAD interface, digitizer, laser scanner (reverse engineering), analytic techniques (e.g. sweeping), image (2-D) and video (3-D) analysis

- Representation, efficient data access and conversion

  Polygonal nets (e.g. triangulation), is the most common representation for rendering objects. Alternatives: finite elements (FEM), constructive solid geometry (CSG), boundary representation (B-rep), implicit surfaces (isosurfaces), surface elements (surfels = points + normals), …

- Manipulation of objects (change shape, …)

  e.g. Boolean operations, local smoothing, interpolation of features (e.g. boundary curves), “engraving” of geometric details, …

1.1 Overview

The topics of this chapter will be:

- Polygonal representations
- Rendering Polygons with OpenGL
- Quadric surfaces
- Blobby Objects
- Spline representations
  - Cubic splines
  - Bézier splines
  - B-Splines
  - Rational splines
- Octree, BSP tree

1.2 Polygonal Representation

Properties/Characteristics:

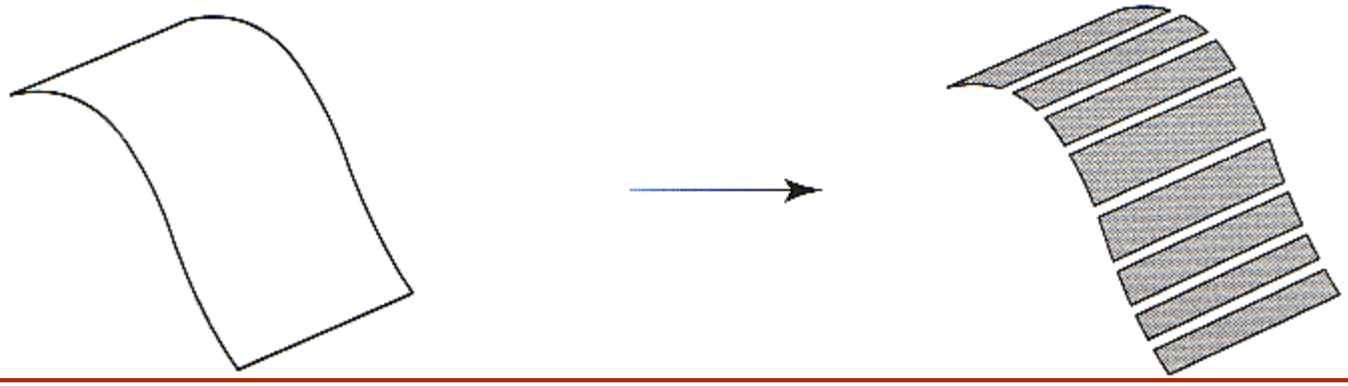
- The precision of the approximation (number and size of polygons) can be chosen depending on the application, but several questions arise, e.g.:
  - What polygonal resolution is required for a precise representation?
  - What polygonal resolution is required for the renderer to make the piecewise approximation appear smooth?
  - What is the correlation between number of polygons and the size of the final display of the object?

  ➞ Often the following rule of thumb is used: Choose the polygonal resolution based on the curvature of the object

1.2 Polygonal Representation

Properties/Characteristics:

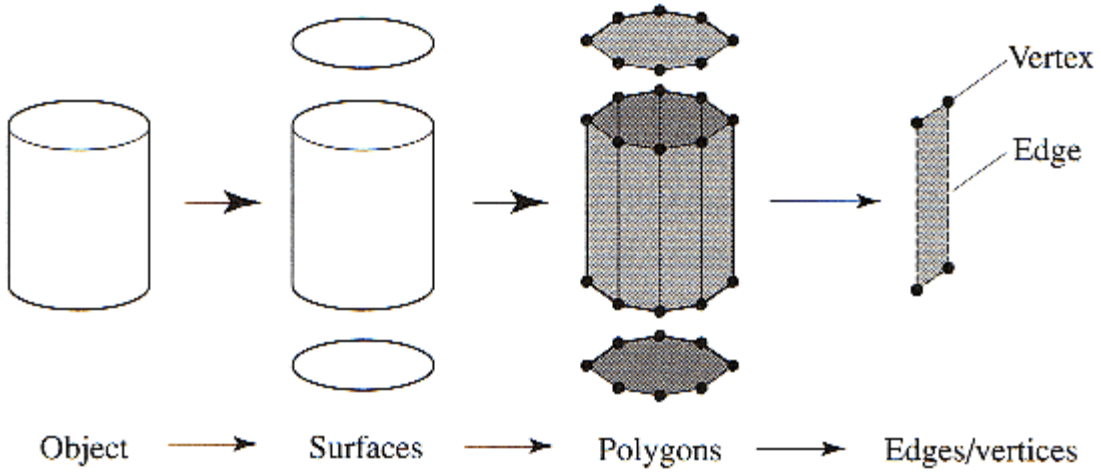
- Classic representations of three-dimensional objects in computer graphics
- Object is represented by a net of polygonal surfaces (usually triangles) ➞ piecewise linear interpolation
- The polygonal surfaces are usually an approximation of the curved surface, representing the object's boundary.



1.2 Polygonal Representation

Hierarchy of the representation:

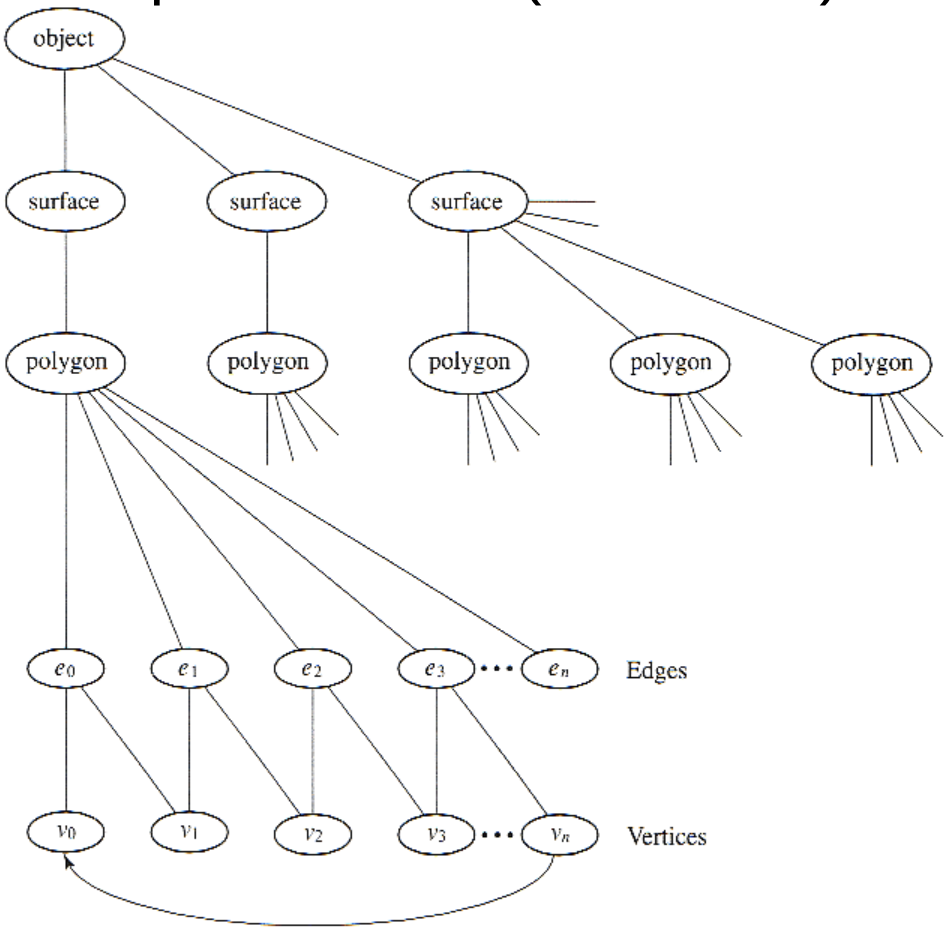
Concept: The object constitutes of several surface elements. Each surface element is represented by several polygons. Every polygon has vertices and edges.



1.2 Polygonal Representation

Hierarchy of the representation (continued):

Topology:



object

surfaces

polygons

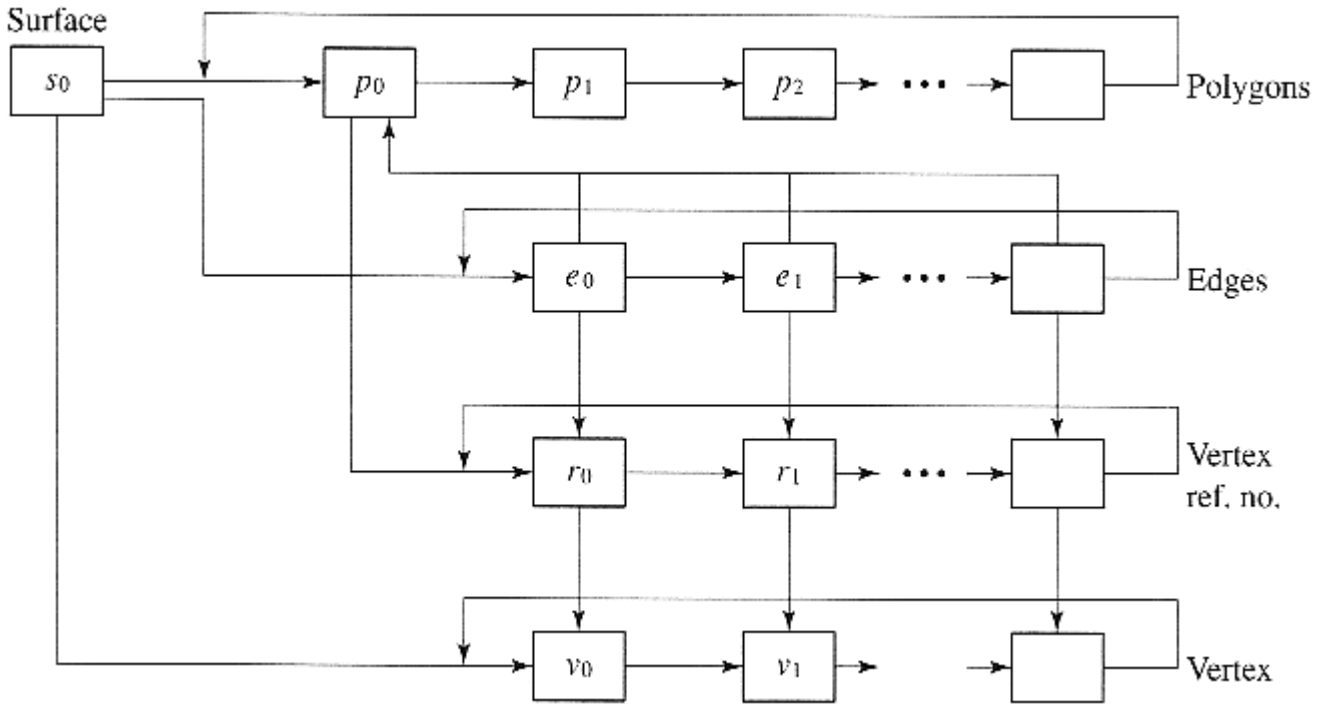
edges

vertices

1.2 Polygonal Representation

Hierarchy of the representation (continued):

Data structure:



Vertices are stored only once

1.2 Polygonal Representation

Comment on data structures:

Data structures can contain – besides geometry information – special attributes required for the application or for the rendering:

- Surface attributes:

  Representation (triangle, polygon, free-form surface), coefficients, normal vector, properties (plane, convex, holes, …), reference to vertices (and edges, if necessary)

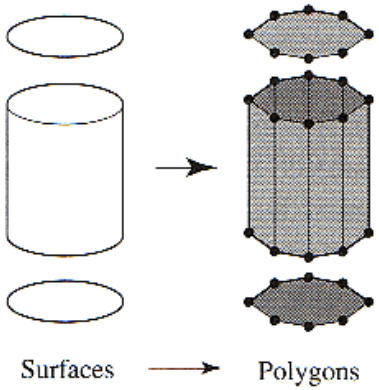
- Edge attributes:

  Length, type (round edge, feature line, virtual edge, reference to vertices and/or polygon

- Vertex attributes:

  Normal vector, color, texture coordinates, reference to polygon and/or edge

1.2 Polygonal Representation



Comment on edges:

Obviously, there are two different kinds of edges involved in the approximate representation:

- Sharp edges (feature lines)
  - This type of edge should be visible
- Virtual edges ("inside" a smooth surface)
  - These should be invisible after rendering
  - Interpolative shading algorithms

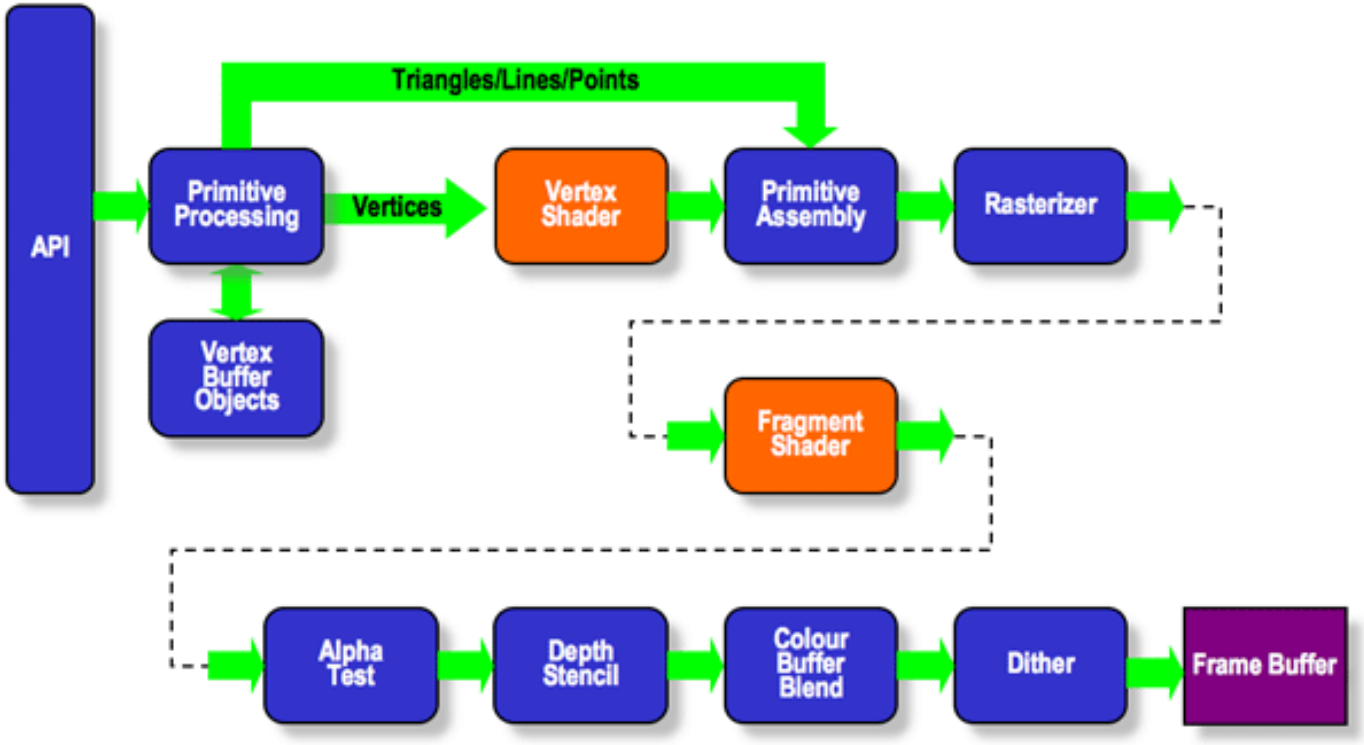
  ➞ flat, Gouraud, Phong shading (now implemented in hardware)

Which kind of edge is to be used can be enforced by the data structure by storing edges multiple times (see image).

1.3 Polygon Rendering with OpenGL

OpenGL rendering pipeline:

Both, vertex and fragment shader are programmable



1.3 Polygon Rendering with OpenGL

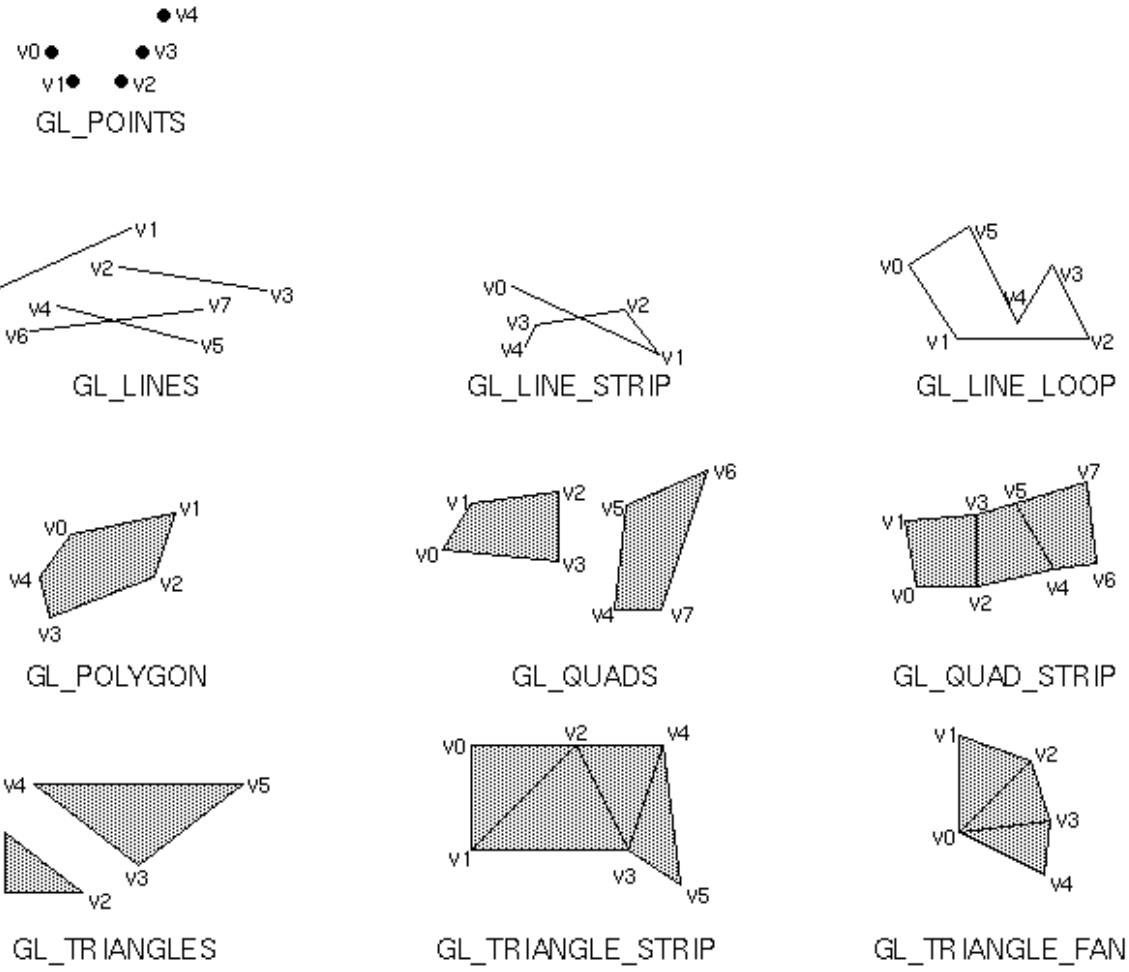
OpenGL supports several types of polygons:

```
GL_POLYGON
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
```

Convenience functions exist for certain objects:

```
glutSolidTetrahedron glutWiredTetrahedron
glutSolidCube        glutWireCube
...
```

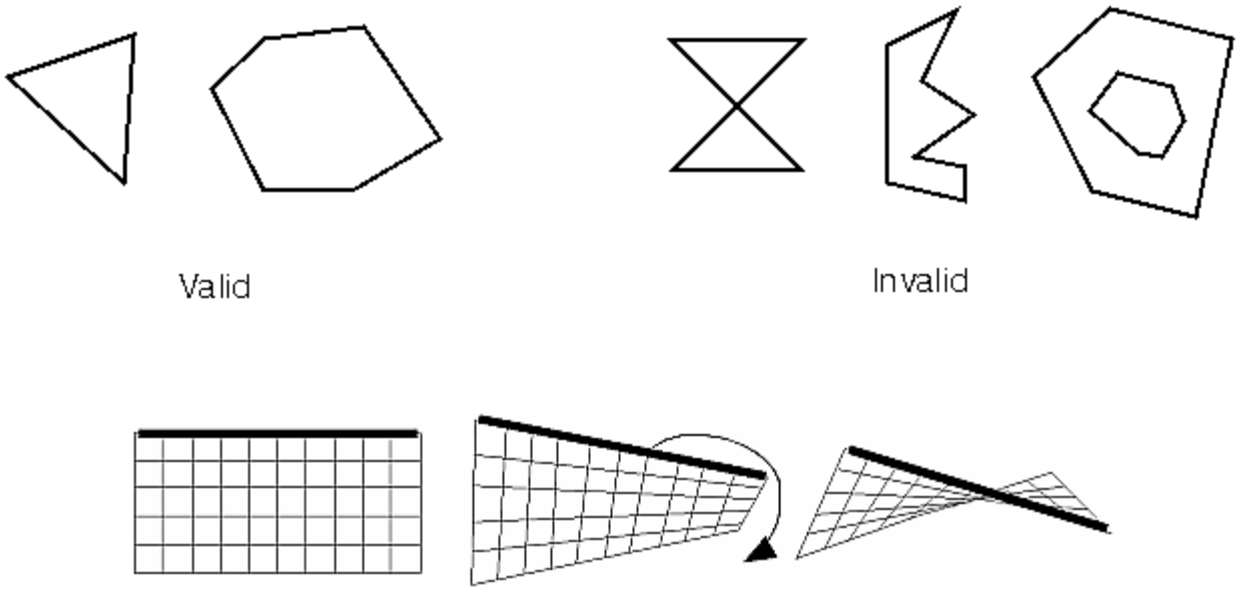
1.3 Polygon Rendering with OpenGL



1.3 Polygon Rendering with OpenGL

Beware:

OpenGL will ignore invalid polygons, e.g. self intersecting, non-convex, or non-planar polygons



1.3 Polygon Rendering with OpenGL

There are basically four different ways to render geometric objects with OpenGL:

- Direct rendering
- Display lists
- Vertex arrays
- Vertex buffer objects

1.3 Polygon Rendering with OpenGL

Direct rendering:

```
glBegin (GL_TRIANGLES);
glNormal3f ( … );
glVertex3f ( … );
…
glNormal3f ( … );
glVertex3f ( … );
glEnd ();
```

In case of polygons with a fixed number of vertices, i.e. triangles, quads, etc., you can generate several such polygons using one `glBegin`/`glEnd` block.

1.3 Polygon Rendering with OpenGL

Display lists:

Stores OpenGL API commands in graphics memory for faster access.

```
GLuint index = glGenLists (1);
if (index != 0) {
  glNewList (index, GL_COMPILE);
   … // draw something
  glEndList ();
}
glCallList (index);
```

Using `GL_COMPILE_AND_EXECUTE` instead of `GL_COMPILE` makes the `glCallList` unnecessary.

1.3 Polygon Rendering with OpenGL

Vertex arrays:

Store vertices in bulk arrays to reduce number of OpenGL function calls.

```
GLfloat vertices[] = { … };
GLfloat normals[] = {… };
glEnableClientState (GL_VERTEX_ARRAY);
glEnableClientState (GL_NORMAL_ARRAY);
glNormalPointer (GL_FLOAT, 0, normals);
glVertexPointer (3, GL_FLOAT, 0, vertices);
glDrawArrays (GL_TRIANGLE_STRIP, 0, 10);
```

This constructs a triangle strips using the first ten elements. The 0 as argument for the arrays is the stride parameter allowing you to skip elements within the arrays.

1.3 Polygon Rendering with OpenGL

Vertex buffer objects (VBO):

Vertex buffer objects are like vertex arrays, but stored in graphics memory for faster access.

Fill the VBO with data; use indices to remember them:

```
GLuint vbovertices, vbonormals;
GLfloat vertices[] = { … }, normals[] = {… };
glGenBuffers (1, vbovertices);
glGenBuffers (1, vbonormals);
glBindBuffer (GL_ARRAY_BUFFER vbovertices);
glBufferData (GL_ARRAY_BUFFER, datasize,
              vertices, GL_STREAM_DRAW);
glBindBuffer (GL_ARRAY_BUFFER, vbonormals);
glBufferData (GL_ARRAY_BUFFER, datasize,
              normals, GL_STREAM_DRAW);
```

1.3 Polygon Rendering with OpenGL

Vertex buffer objects (continued):

Now, draw the previously generated VBOs:

```
glBindBuffer (GL_ARRAY_BUFFER vbovertices);
glVertexPointer (3, GL_FLOAT, 0, (GLvoid *)0);
glBindBuffer (GL_ARRAY_BUFFER, vbonormals);
glNormalPointer (GL_FLOAT, 0, (GLvoid *)0);
glDrawArrays (GL_TRIANGLE_STRIP, 0, count);
```

Notes:

- There is no actual data pointer required for the `glVertexPointer` and `glNormalPointer` calls since the VBOs are used as data repository.
- The client states need to be set just like with vertex arrays

1.6 Spline Representations

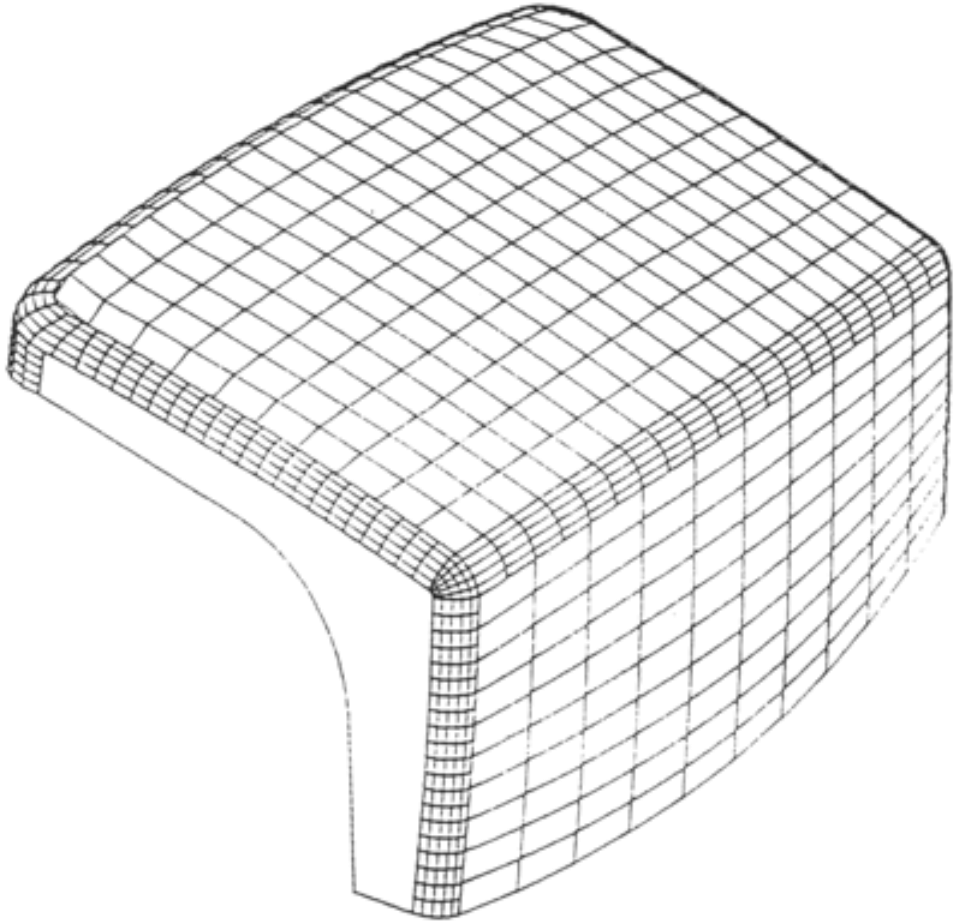
Motivation:

Free-form-curves and –surfaces became very popular during the last decade, particularly due to their application in several engineering disciplines. Free-form-curves are nowadays a fundamental design method in CAD/CAM software. This section will introduce basic concepts.

Specifically, we will cover:

- Interpolation using polynomials and splines
- Bézier curves, B-splines
- Rational B-splines

1.6 Spline Representations

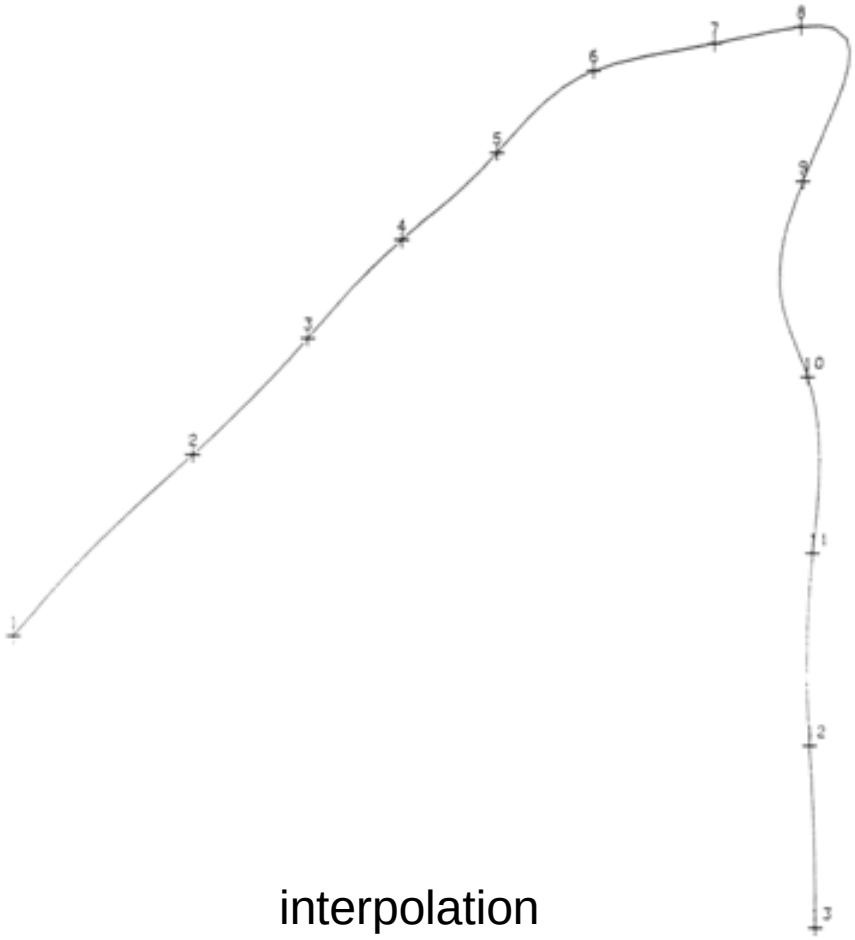


Surface consisting of several segments (patches)

1.6 Spline Representations

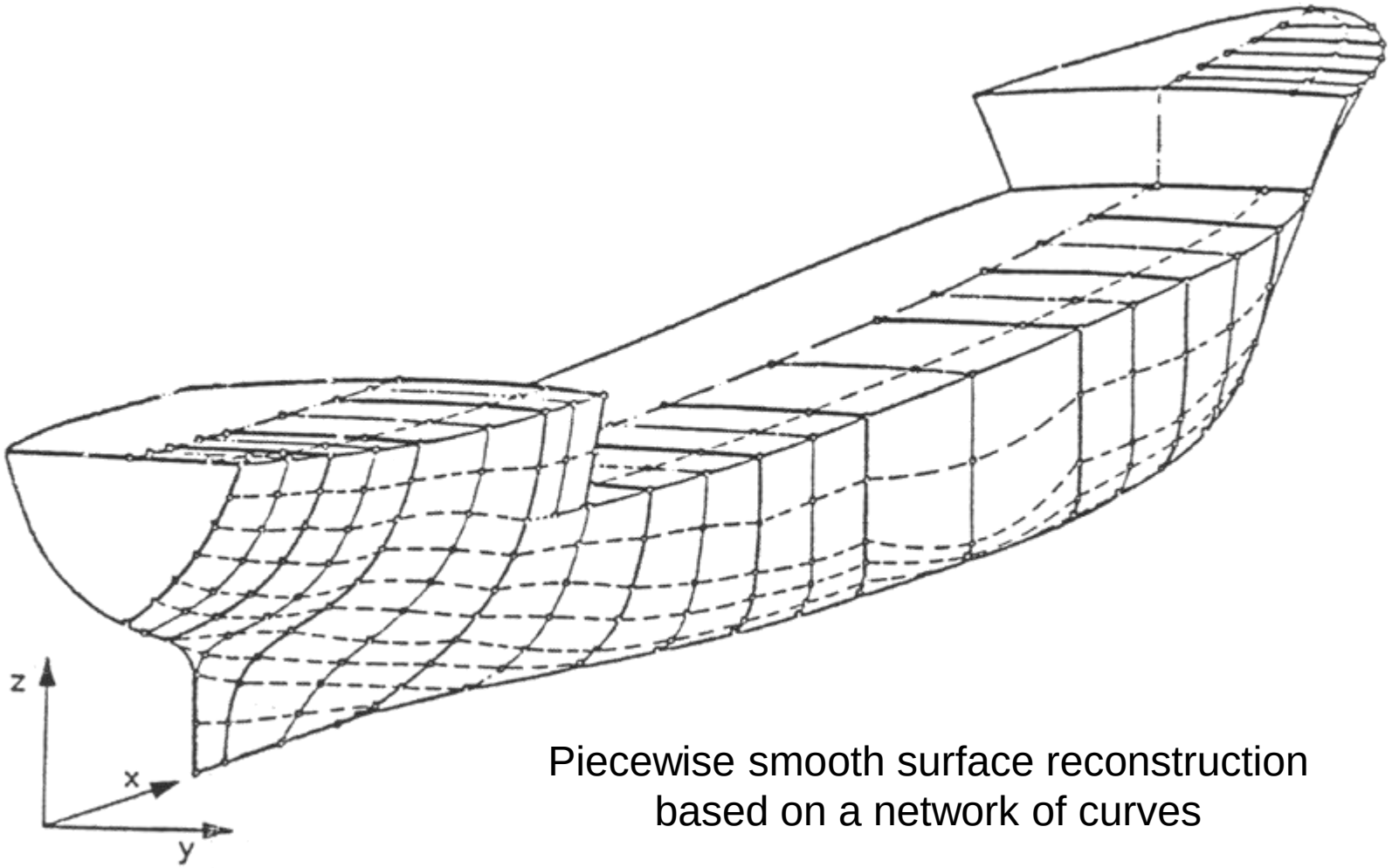
interpolation

1.6 Spline Representations



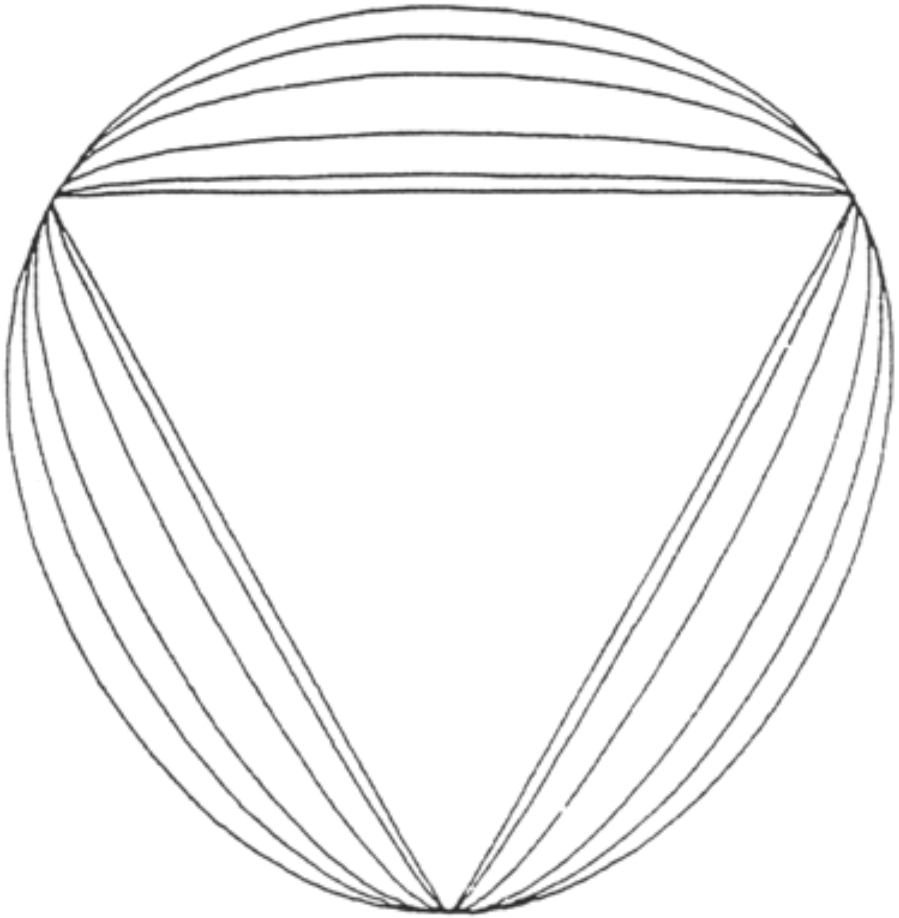
interpolation

1.6 Spline Representations



Piecewise smooth surface reconstruction
based on a network of curves

1.6 Spline Representations



Usually, more than one possible solution exist to the interpolation problem

1.6 Spline Representations

Interpolation problem:

Let $(t_i, f_i) \quad i = 0,..., n$

be a set of pairs of real numbers with pairwise unequal nodes t_i ,

$$t_i \neq t_j \quad \text{for} \quad i \neq j$$

A polynomial p of degree less than $n+1$,

$$p(t) = \sum_{j=0}^{n} c_j \cdot t^j$$

with real coefficients c_j is called interpolation problem of

(t_i, f_i)

if $p(t_i) = f_i$ for $i = 0,..., n$

1.6 Spline Representations

Questions:

- Does a unique solution to this problem exist?
- Is there an algorithmic method to solve the problem? Is it efficient enough?
- Is the quality good enough for the application?

Theorem 1: There exist a unique solution to the interpolation problem.

1.6 Spline Representations

Proof:

Plugging in the definition of the polynomial into the interpolation problem results in a system of linear equations:

$$\begin{pmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & \cdots & t_1^n \\ \vdots & & & & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}$$

or $A \cdot c = f$. The matrix $\boldsymbol{A}$ is the well-known Vandermonde matrix with the property:

$$\det A = \prod_{i,j=0;i>j}^{n} (t_i - t_j)$$

Since $\boldsymbol{A}$ is regular, we have proven the theorem.

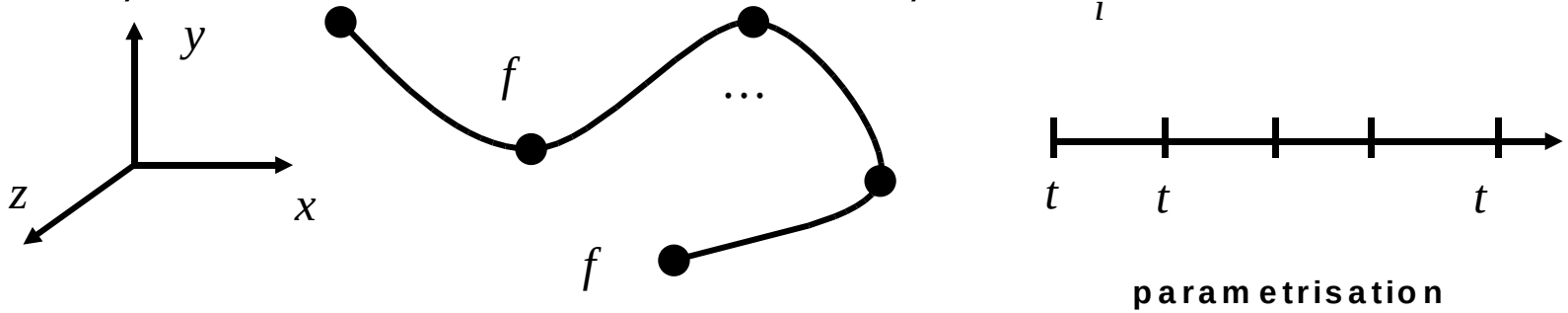
1.6 Spline Representations

Note: Let $f_i = (x_i, y_i, z_i) \in IR^3$ be a set of $n+1$ points at different nodes t_i.

Then, an interpolation 3-D curve p with $p(t_i) = f_i$

can be determined in an analog fashion by solving the linear equation system: $Ac = f$

Here, the coefficients are vectors, i.e. $c_i \in IR^3$



For each coordinate, we get a linear system with an identical matrix A.

1.6 Spline Representations

Lagrange interpolation

The solution of the linear equation system of the interpolation problem usually has complexity $O(n^3)$. However, we would prefer a set of basis functions with the property:

$$\left\{ L_i(t) \right\}_{i=0}^{n} \quad \text{such that} \quad L_i(t_j) = \delta_{ij}$$

Using these so called blending functions simplifies the matrix of the linear equation system to the identity matrix, i.e. $c_i = f_i \quad (i = 0, \ldots, n)$ and the resulting polynomial can be written as

$$p(t) = \sum_{i=0}^{n} f_i L_i(t)$$

1.6 Spline Representations

Theorem 2: The Lagrange polynomials

$$L_i(t) = \prod_{k=0,\,k\neq i}^{n} \frac{t - t_k}{t_i - t_k}$$

fulfill the desired property:

$$L_i(t_j) = \delta_{ij}.$$

1.6 Spline Representations

Proof:

Plug in t_i

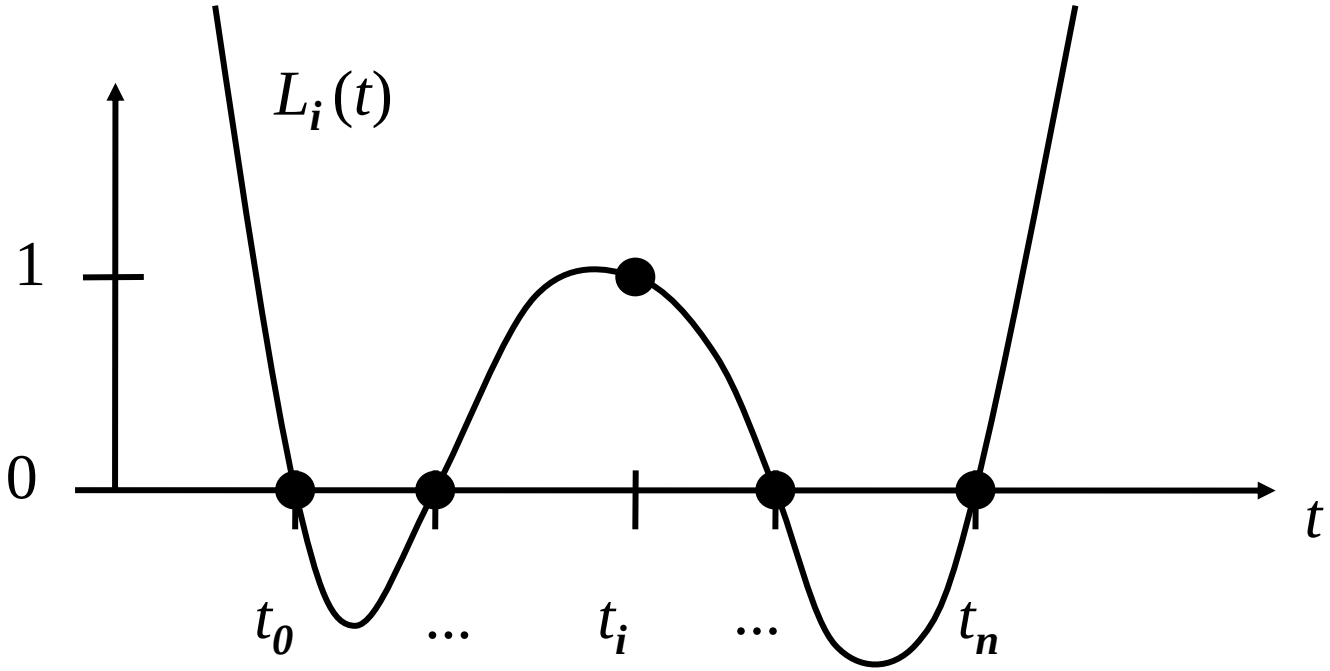
$$L_i(t_i) = \prod_{k=0,\, k \neq i}^{n} \frac{t_i - t_k}{t_i - t_k} = 1$$

and for

$$j \neq i : L_i(t_j) = \ldots \frac{t_j - t_j}{t_i - t_j} \ldots = 0$$

1.6 Spline Representations

Lagrange polynomials



1.6 Spline Representations

Example:

1. n=1, i.e. linear interpolation

   Interpolate the points (x_0,y_0) and (x_1, y_1) (use x_0,x_1 as nodes)

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \qquad L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

$$P(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0} = \frac{y_0(x - x_1) + y_1(x_0 - x)}{x_0 - x_1}$$

2. Interpolate the following set of points:

i	x_i	y_i
0	0	1
1	1	0.5
2	2	0.2

1.6 Spline Representations

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 1)(x - 2)}{(0 - 1)(0 - 2)} = \frac{1}{2}(x^2 - 3x + 2)$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x - 0)(x - 2)}{(1 - 0)(1 - 2)} = -x^2 + 2x$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x - 0)(x - 1)}{(2 - 0)(2 - 1)} = \frac{1}{2}(x^2 - x)$$

$$P(x) = \frac{1}{2}(x^2 - 3x + 2) \cdot 1 + (-x^2 + 2x) \cdot 0.5 + \frac{1}{2}(x^2 - x) \cdot 0.2$$

$$= 0.1 \cdot x^2 - 0.6 \cdot x + 1$$

1.6 Spline Representations

Example:

cubic Lagrange polynomials with uniform nodes $t_i = \frac{i}{3}$

1.6 Spline Representations

$$L_0(t) = \left(\frac{t - t_1}{t_0 - t_1}\right)\left(\frac{t - t_2}{t_0 - t_2}\right)\left(\frac{t - t_3}{t_0 - t_3}\right) = \left(\frac{t - \frac{1}{3}}{-\frac{1}{3}}\right)\left(\frac{t - \frac{2}{3}}{-\frac{2}{3}}\right)\left(\frac{t - 1}{-1}\right)$$

$$= \frac{t^3 - 2t^2 + \frac{11}{9}t - \frac{2}{9}}{-\frac{2}{9}} = -\frac{9}{2}t^3 + 9t^2 - \frac{11}{2}t + 1$$

$$L_1(t) = \left(\frac{t - t_0}{t_1 - t_0}\right)\left(\frac{t - t_2}{t_1 - t_2}\right)\left(\frac{t - t_3}{t_1 - t_3}\right) = \left(\frac{t}{\frac{1}{3}}\right)\left(\frac{t - \frac{2}{3}}{-\frac{1}{3}}\right)\left(\frac{t - 1}{-\frac{2}{3}}\right)$$

$$= \frac{t^3 - 2t^2 + \frac{11}{9}t - \frac{2}{9}}{-\frac{2}{9}} = -\frac{27}{2}t^3 + \frac{45}{2}t^2 - 9t$$

$$L_2(t) = L_1(1 - t) = -\frac{27}{2}t^3 + 18t^2 - \frac{9}{2}t$$

$$L_3(t) = L_0(1 - t) = \frac{9}{2}t^3 - \frac{9}{2}t^2 + t$$

1.6 Spline Representations

Newton interpolation

The Newton scheme has the advantage of being a dynamic scheme, i.e. additional nodes can be added without having to re-compute all basis functions.

For this scheme, the following basis functions are used:

$$N_i(t) = \prod_{k=0}^{i-1} (t - t_k)$$

with properties $N_i(t_j) = 0$ for $i > j$

$$N_i(t_i) \neq 0 \; .$$

1.6 Spline Representations

The coefficients a_i for the solution

$$p(t) = \sum_{i=0}^{n} a_i N_i(t)$$

Are computed recursively using the k-th divided differences $f[t_j, \ldots, t_{j+k}]$

$$f[t_j] := f_j \qquad (j = 0, \ldots, n)$$

$$f[t_j, \ldots, t_{j+k}] := \frac{f[t_{j+1}, \ldots, t_{j+k}] - f[t_j, \ldots, t_{j+k-1}]}{t_{j+k} - t_j}$$

$$a_i = f[t_0, \ldots, t_i]$$

1.6 Spline Representations

This results in the following scheme:

	k = 0	1	2	3
t_0	$f_0 = f[t_0] = a_0$			
		$f[t_0, t_1] = a_1$		
t_1	$f_1 = f[t_1]$		$f[t_0, t_1, t_2] = a_2$	
		$f[t_1, t_2]$		$f[t_0, \ldots, t_3] = a_3$
t_2	$f_2 = f[t_2]$		$f[t_1, t_2, t_3]$	…
		$f[t_2, t_3]$	…	
t_3	$f_3 = f[t_3]$	…		
⋮	⋮			

1.6 Spline Representations

Note: the coefficients a_i can also be determined using a linear equation system. Since the matrix of the resulting linear system is a triangular matrix, solving this system would be equivalent to the scheme using the divided differences.

Example: $(t_i, f_i) \in \{(0,1); (2,3); (4,5)\}$

t_i	f_i		
0	$1=a_0$		
		$1=a_1$	
2	3		$0=a_2$
		1	
4	5		

$$p(x) = a_0 + a_1(x-t_0) + a_2(x-t_0)(x-t_1)$$
$$= 1 + x$$

1.6 Spline Representations

Beware: the interpolating polynomial interpolating $n+1$ nodes is not necessarily of degree n, but at most of degree n.

Note:

- The order of the nodes does not change the result when using Newton's scheme
- Interpolating a continuous function f on the interval $[a,b]$ using n points does not necessarily ensure that the series of interpolating polynomials f_n converges to f.

Conclusion: Using more points does not necessarily improve the quality of the resulting interpolating polynomial!

1.6 Spline Representations

Disadvantages of polynomial interpolation (particularly with respect to CAD/CAM):

- Interpolating polynomials of degree larger than 5 often are quite “wavy”

  Remedy: introduce additional conditions, such as minimization of folding energy

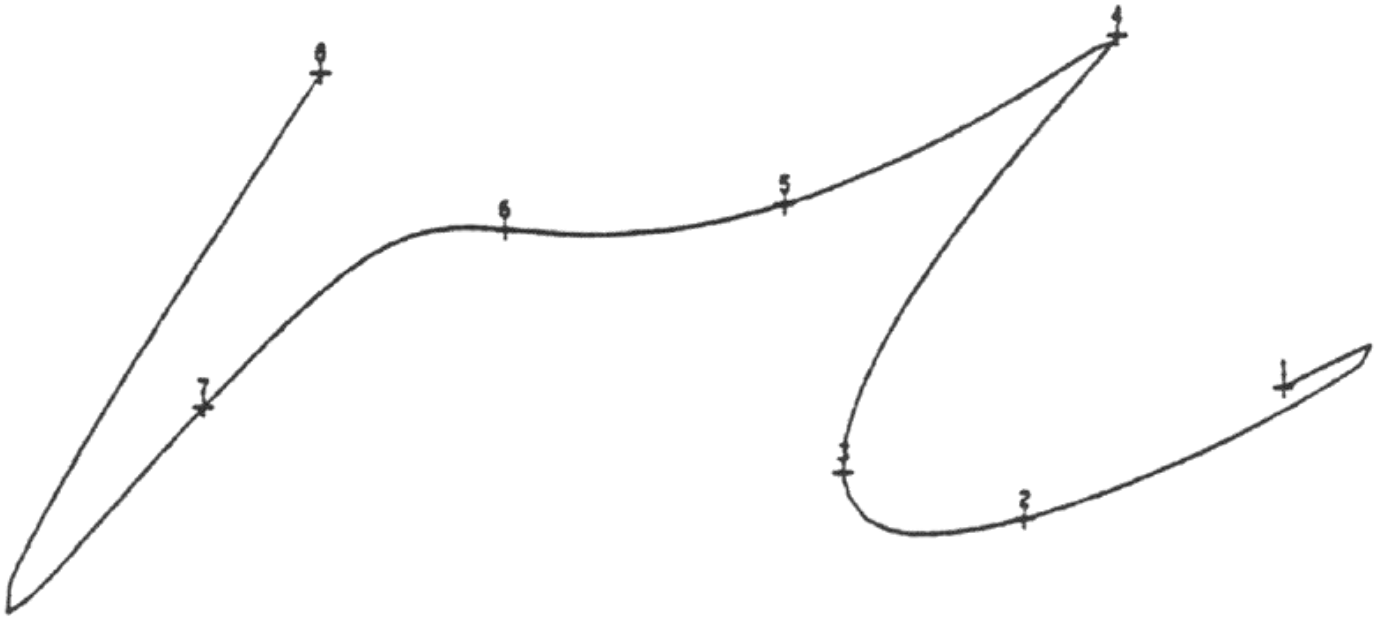
  (→ splines)

- Each point that is to be interpolated influences the resulting curve globally

  Remedy: Use basis functions with local influence

1.6 Spline Representations

The higher the degree of a polynomial, the more wavy its shape, especially at the end points of the interval. The parameterization (choice of nodes) influences the quality of the resulting curve.



1.6 Spline Representations

Interpolating derivatives

Let t_i $(i=0,\ldots,n)$ be different nodes and for each i the values of the first n_i-1 derivatives are known:

$$f_i,\ f_i^{(1)},\ \ldots,\ f_i^{(n-1)}\quad (i=0,\ldots,n)$$

We are looking for a polynomial of degree $\leq m = \sum_{i=0}^{n} n_i - 1$

$$p(t) = \sum_{j=0}^{m} c_j t^j$$

such that $p^{(j)}(t_i) = f_i^{(j)}$ $(i=0,\ldots n;\ j=0,\ldots,n_i\text{-}1)$

1.6 Spline Representations

Plugging in the desired conditions into the polynomial equation – similar to the previous interpolation problem – results in a linear system of equations.

Theorem 3: There is a unique solution to this linear system of equations

Proof: Overall, there are $m+1$ conditional equations with $m+1$ coefficients. The system is regular if the homogenous problem $(f_i^{(j)}=0)$ only allows the trivial solution. This is exactly the case: since p has exactly $m+1$ zeros (including multiplicities) and p is of degree $\leq m$, the polynomial p has to be zero.

1.6 Spline Representations

Example: We are looking for the cubic polynomial which interpolates $f(0)$, $f'(0)$, $f(1)$, and $f'(1)$. A cubic polynomial can be described as:

$$p(t) = c_3t^3 + c_2t^2 + c_1t + c_0$$

And the derivative:

$$p'(t) = 3c_3t^2 + 2c_2t + c_1$$

This gives us:

$$f(0) = c_0$$
$$f'(0) = c_1$$
$$f(1) = c_3 + c_2 + c_1 + c_0$$
$$f'(1) = 3c_3 + 2c_2 + c_1$$

1.6 Spline Representations

Or in matrix form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} f(0) \\ f'(0) \\ f(1) \\ f'(1) \end{pmatrix}$$

1.6 Spline Representations

Hermite interpolation:

In analogy to the Lagrange interpolation, we can find basis polynomials that are optimal for interpolating derivatives. These are called **Hermite polynomials.**

Example: we are looking for cubic Hermite polynomials for the system resulting from the previous example. The coefficients can be determined by inverting the matrix:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} f(0) \\ f'(0) \\ f(1) \\ f'(1) \end{pmatrix}$$

1.6 Spline Representations

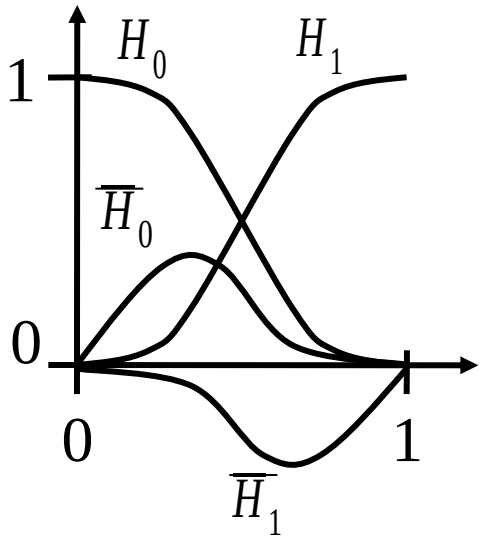
Then, the resulting Hermite polynomials are:

$$H_0(t) = 2t^3 - 3t^2 + 1$$

$$\overline{H}_0(t) = t^3 - 2t^2 + t$$

$$H_1(t) = -2t^3 + 3t^2$$

$$\overline{H}_1(t) = t^3 - t^2$$



The Hermite polynomials have the following properties:

$$H_i(j) = \delta_{ij} \qquad H_i'(j) = 0$$

$$\overline{H}_i(j) = 0 \qquad \overline{H}_i'(j) = \delta_{ij}$$

$$i, j = 0,1$$

Interpolation: $X(t) = H_0 f(0) + H_1 f(1) + \overline{H}_0 f'(0) + \overline{H}_1 f'(1)$

1.6 Spline Representations

Bézier segments:

Modeling of individual curve segments is easier and more intuitive if there is a correlation between the coefficients (design parameters) and the geometry of the curve. Interpolated points are not very suitable for this matter because the waviness cannot be controlled by the interpolated points. **Bézier segments** are polynomial curves, which are defined through a **control polygon**. This control polygon is approximated by the resulting curve, but only interpolates at the end points. The vertices of the control polygon (control points, here: Bézier points) constitute the coefficients of the representation using Bernstein polynomials (new basis).

1.6 Spline Representations

Bézier segments:

By changing the geometric layout of the control polygon, the curve can be easily modified. One of the properties of Bézier segments is that the number of inflection points of the curve is less or equal to the number of inflection points of the control polygon. This variation diminishing property allows for good control of the waviness of the curve.

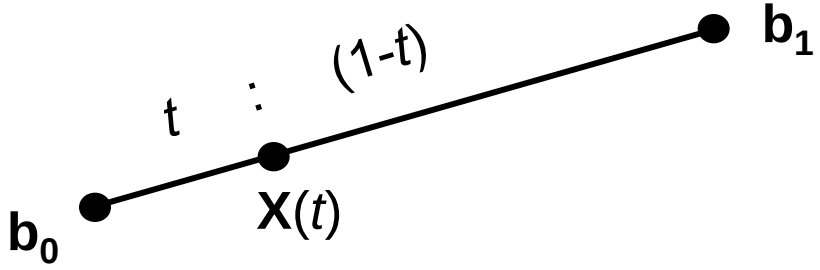
Bézier segments have many applications and are often used for modeling of composite curves and surfaces (Bézier splines). The Bézier technique is further explained in the following.

1.6 Spline Representations

Bézier segments: the de Casteljau algorithm

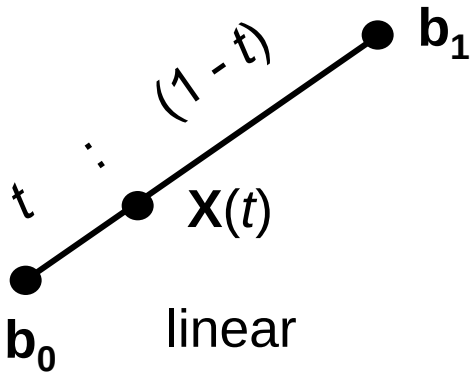
The de Casteljau algorithm [Cast.59][Böhm84] generalizes linear interpolation of polynomial curves. Let b_0 and b_1 be two points and t a parameter between 0 and 1. Then t uniquely defines a point on the linear segment connecting b_0 and b_1:

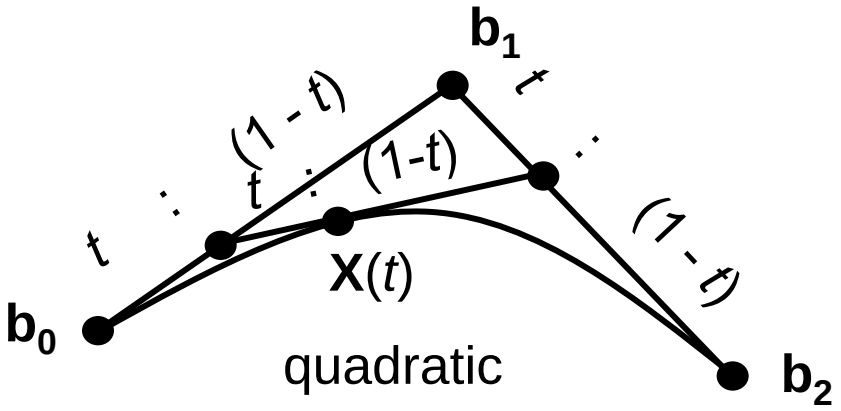
$$X(t) = (1 - t)\, b_0 + t\, b_1$$

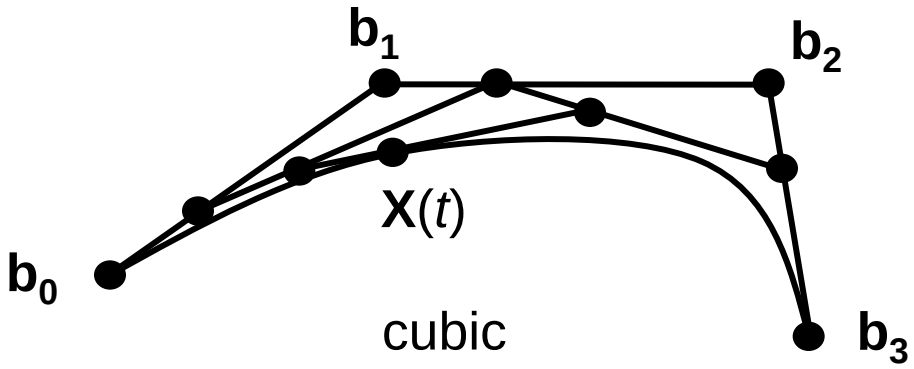


1.6 Spline Representations

de Casteljau algorithm







1.6 Spline Representations

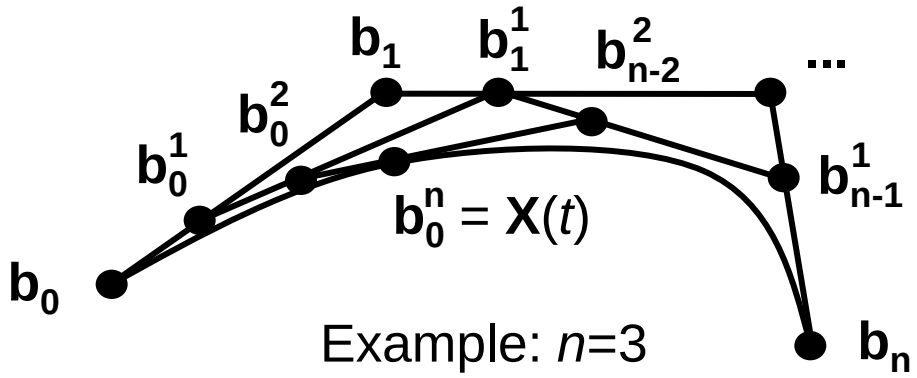
de Casteljau algorithm

Let b_i $(i=0,\ldots,n)$ be $n+1$ Bézier points. Then, the de Casteljau algorithm for evaluating a Bézier segment is based on the following recursion:

$$b_i^0 = b_i \quad (i = 0,\ldots, n)$$

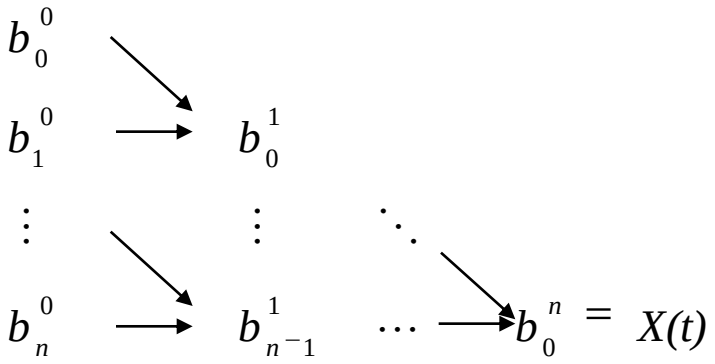
$$b_i^j = (1-t)b_i^{j-1} + tb_{i+1}^{j-1} \quad (i = 0,\ldots, n-j,\ j = 1,\ldots, n)$$

$$X(t) = b_0^n$$



1.6 Spline Representations

de Casteljau algorithm



Scheme of the de Casteljau algorithm

Every point b_i^j is a convex combination of its predecessors b_i^{j-1} and b_{i+1}^{i-1}, weighted using $(1-t)$ and t, respectively.

$b_i^j(t)$ is a polynomial of degree j (or less).

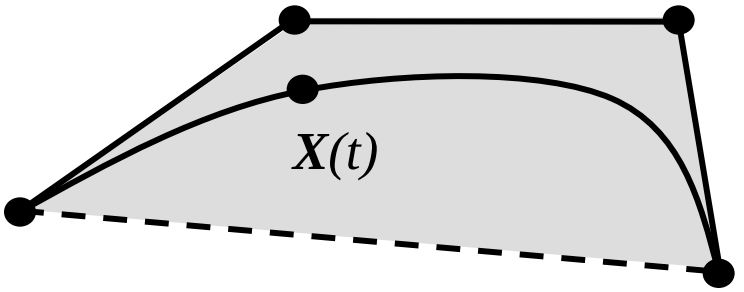
1.6 Spline Representations

Properties of Bézier segments

1) **Convex hull**: the curve $X(t)$, $t \in [0,1]$ is located within the convex hull of the control polygon, i.e. there are weights $\alpha_i \geq 0$ with

$$X(t) = \sum_{i=0}^{n} \alpha_i b_i$$

$$\sum_{i=0}^{n} \alpha_i = 1$$

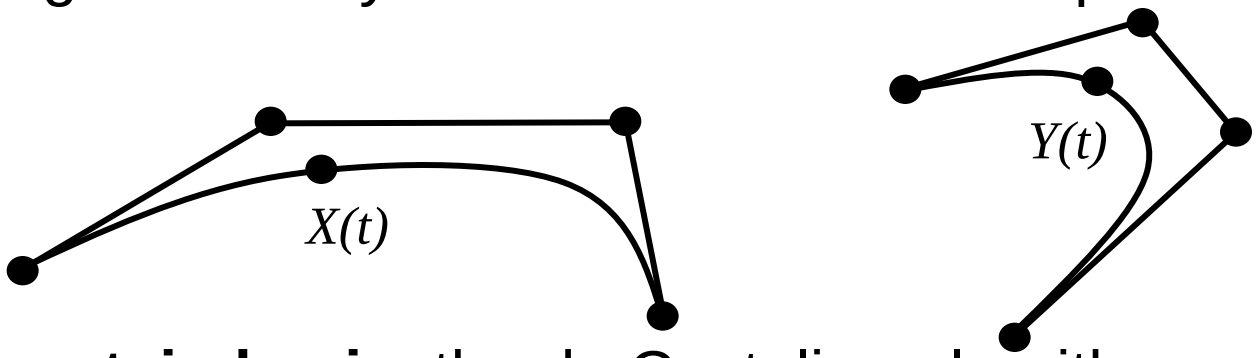


2) **Variation diminishing**: an arbitrary straight line intersects the curves as often or less than the control polygon (within the plane).

3) **End point interpolation**: $X(0) = b_0$, $X(1) = b_n$.

1.6 Spline Representations

4) **Affine invariance**: let φ be an affine mapping $\varphi(p) = Ap+v$. Then, $\varphi(X(t)) = Y(t)$, with the curve Y being defined by the transformed Bézier points $\varphi(b_i)$.



5) **Bernstein basis**: the de Casteljau algorithm results in a curve of (maximal) degree n. As basis the so called **Bernstein polynomials** are used:

$$X(t) = \sum_{i=0}^{n} b_i B_i^n(t), \quad B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

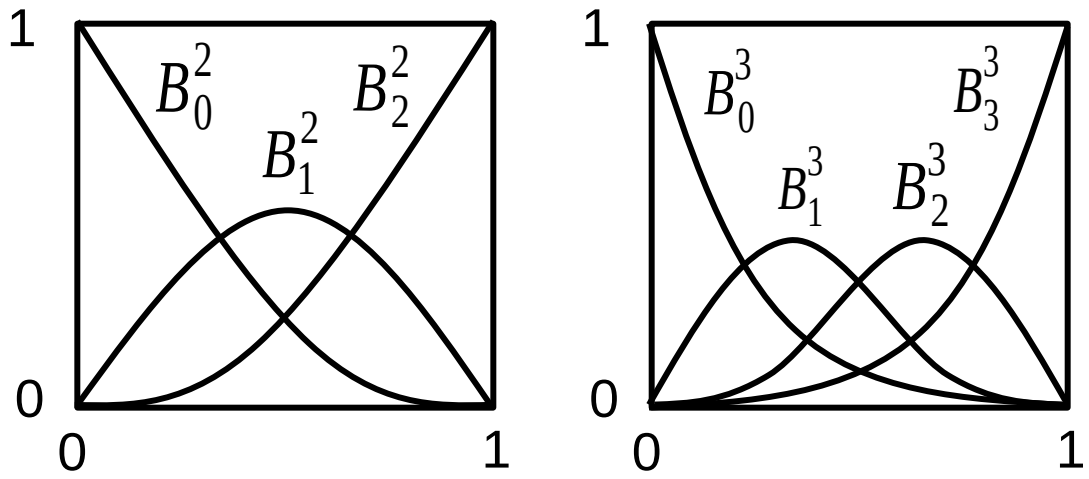
1.6 Spline Representations

6) **Properties of the Bernstein polynomials**: the Bernstein polynomials are symmetric, not negative between *[0,1]*, and the sum of all polynomials is one:

$$B_i^n(t) = B_{n-i}^n(1-t)$$

$$B_i^n(t) \geq 0, \quad t \in [0,1]$$

$$\sum B_i^n(t) = 1$$



1.6 Spline Representations

7) **Symmetry**: inverting the sequence of the control points b_i of $X(t)$ results in a Bézier segment $Y(t)$ with inverted parameterization: $Y(t) = X(1-t)$.

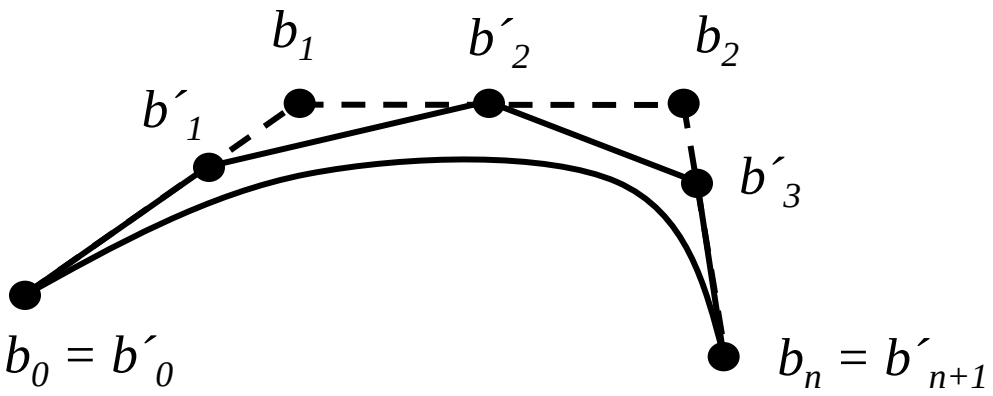
8) **Pseudo-local control**: the Bernstein polynomials influence the curve globally, however, their maxima are located in the proximity of the control points: $max(B_i^n) = B_i^n(i/n)$.

   Moving a control point b_i results in a limited change of the curve. The change is maximal at $t=i/n$. (Interpolated polynomials do not allow for pseudo-local control). However, small changes to the control points can change the curve significantly.

1.6 Spline Representations

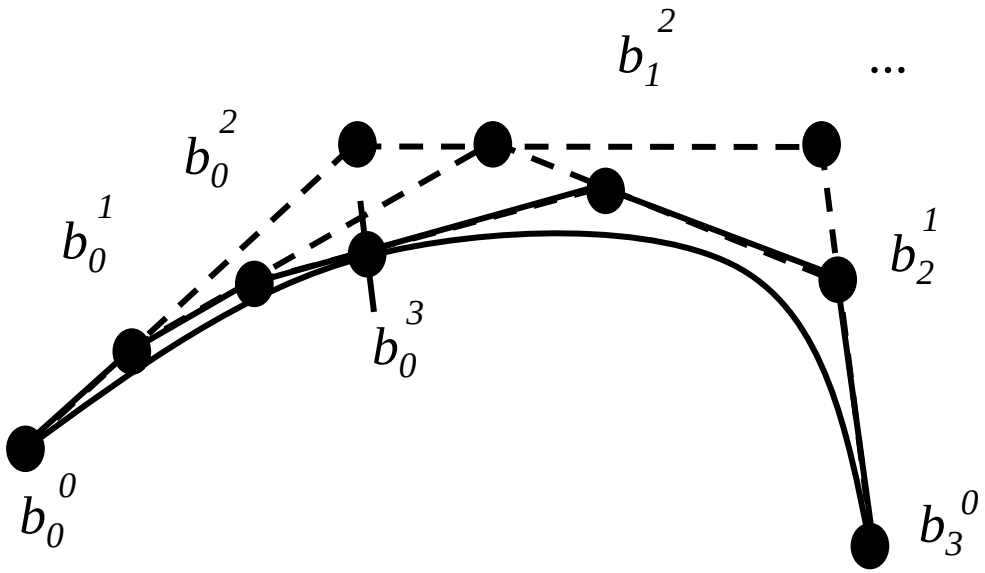
9) **Degree increase**: a Bézier segment of degree n can be represented as a Bézier segment of degree n+1 (of course, the actual degree will be the same but the representation will use the higher degree Bernstein polynomials):

$$b_i' = \frac{i}{n+1} b_{i-1} + \left(1 - \frac{i}{n+1}\right) b_i, \qquad (i = 0, \ldots, n+1)$$



1.6 Spline Representations

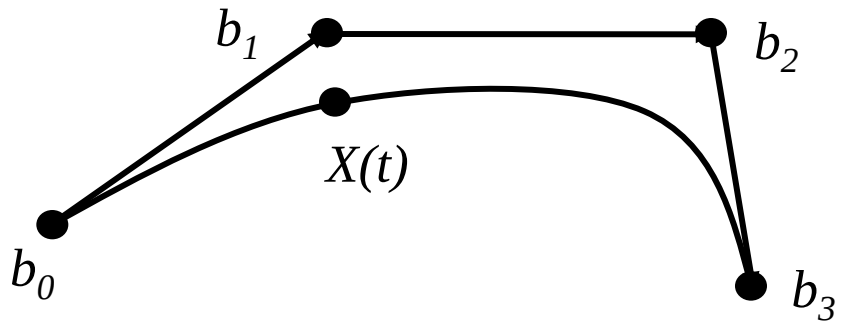
10) Subdivision: a Bézier segment can be separated at a location t into two Bézier segments using the de Casteljau algorithm. The new control points are b_0^i and b_i^{n-i} $(i=0,\ldots,n)$.

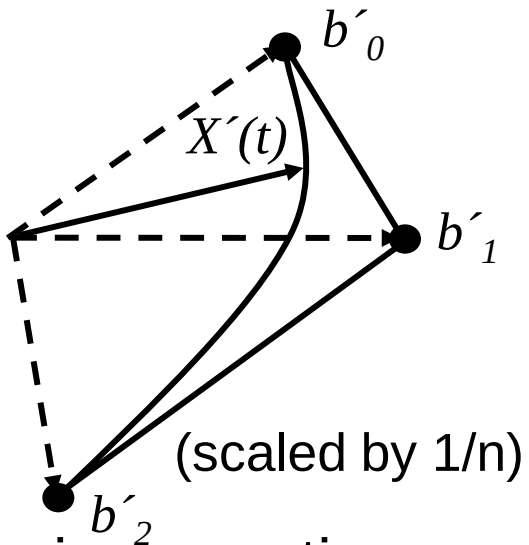


1.6 Spline Representations

11) Derivative: the derivative of a Bézier segment of degree n can be represented as a Bézier segment of degree n-1 using the control points

$$b_i' = n(b_{i+1} - b_i) \quad (i=0,\ldots,n).$$





This can be concluded from the following equation

$$\frac{d}{dt} B_i^n(t) = n\left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right) \qquad B_{-1}^{n-1}, B_n^{n-1} := 0$$

1.6 Spline Representations

Derivatives

For higher derivatives you can use the following recursive formula:

$$\frac{d^p}{dt^p} X(t) = \frac{n!}{(n-p)!} \sum_{i=0}^{n-p} \Delta^p b_i B_i^{n-p}(t)$$

with $\Delta^p b_i = \Delta^{p-1} b_{i+1} - \Delta^{p-1} b_i$ and $\Delta^0 b_i = b_i$

The derivatives can also be determined using the derivatives of the Bernstein polynomials:

$$\frac{d^p}{dt^p} B_i^n(t) = \frac{n!}{(n-p)!} \sum_{k=0}^{p} (-1)^k \binom{p}{k} B_{i-p+k}^{n-p}(t)$$

1.6 Spline Representations

Derivatives at end points

At the end points the p-th derivative only depend on $p+1$ control points:

$$X'(0) = n(b_1 - b_0)$$

$$X''(0) = n(n-1)(b_2 - 2b_1 + b_0)$$

$$X'''(0) = n(n-1)(n-2)(b_3 - 3b_2 + 3b_1 - b_0)$$

$$X'(1) = n(b_n - b_{n-1})$$

$$X''(1) = n(n-1)(b_n - 2b_{n-1} + b_{n-2})$$

$$X'''(1) = n(n-1)(n-2)(b_n - 3b_{n-1} + 3b_{n-2} - b_{n-3})$$

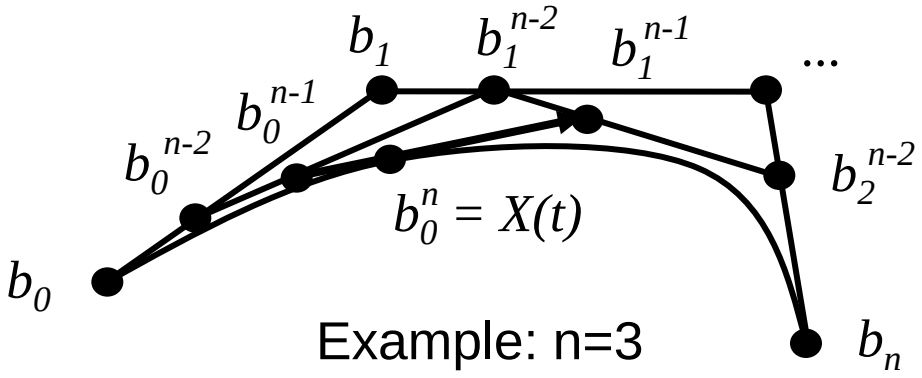
1.6 Spline Representations

Derivatives

The derivatives of a Bézier segment can also be calculated using the de Casteljau algorithm:

$$X'(t) = n(b_1^{n-1} - b_0^{n-1})$$

$$X''(t) = n(n-1)(b_2^{n-2} - 2b_1^{n-2} + b_0^{n-2})$$



1.6 Spline Representations

Comments:

1) Due to the convex hull property the bounding box of the control points encloses the Bézier segment. By splitting the segment recursively into sub-segments this enclosure can be refined.
2) The variation diminishing property means that the approximation using the Bernstein polynomials is at least as smooth as the control polygon itself. Hence, the waviness of the resulting curve can controlled by the control polygon.

1.6 Spline Representations

Example (n=3): basis transformation (to monomials)

$$X(t) = b_0(-t^3 + 3t^2 - 3t + 1)$$
$$+ b_1(3t^3 - 6t^2 + 3t)$$
$$+ b_2(-3t^3 + 3t^2)$$
$$+ b_3(t^3)$$

$$= \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

1.6 Spline Representations

Example (n=3): Equivalence of Bernstein basis and de Casteljau

$$X(t) = b_0(1-t)^3 + b_1 3(1-t)^2 t + b_2 3(1-t)t^2 + b_3 t^3$$

$$= \underbrace{(b_0(1-t) + b_1 t)}_{b_1^1}\underbrace{(1-t)}_{B_0^2}{}^2 + \underbrace{(b_1(1-t) + b_2 t)}_{b_2^1}\underbrace{(2(1-t)t)}_{B_1^2}$$

$$+ \underbrace{(b_2(1-t) + b_3 t)}_{b_2^1}\underbrace{t^2}_{B_2^2}$$

$$= \underbrace{(b_1^1(1-t) + b_2^1 t)}_{b_2^2}\underbrace{(1-t)}_{B_0^1} + \underbrace{(b_2^1(1-t) + b_3^1 t)}_{b_3^2}\underbrace{t}_{B_1^1}$$

$$= b_2^2(1-t) + b_3^2 t = b_3^3 B_0^0 = b_3^3$$

1.6 Spline Representations

Example (de Casteljau algorithm)

The control polygon has the vertices *(0,0)*, *(2,4.5)*, *(8.5,6.5)*, and *(11,2)*. Compute *X(0.6)* of the cubic Bézier segment:

Splitting ratio					
	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$				
0.4	$\begin{pmatrix} 2 \\ 4.5 \end{pmatrix}$	$\begin{pmatrix} 1.2 \\ 2.7 \end{pmatrix}$			
0.6	$\begin{pmatrix} 8.5 \\ 6.5 \end{pmatrix}$	$\begin{pmatrix} 5.9 \\ 5.7 \end{pmatrix}$	$\begin{pmatrix} 4.02 \\ 4.5 \end{pmatrix}$		
	$\begin{pmatrix} 11 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 10 \\ 3.8 \end{pmatrix}$	$\begin{pmatrix} 8.36 \\ 4.56 \end{pmatrix}$	$\begin{pmatrix} 6.63 \\ 4.54 \end{pmatrix}$	$\Rightarrow X(0.6) = \begin{pmatrix} 6.63 \\ 4.54 \end{pmatrix}$

1.6 Spline Representations

Example (continued)

Compute the first derivative $X'(0.6)$ and second derivative $X''(0.6)$:

$$X'(0.6) = 3 \cdot \left(b_3^2 - b_2^2 \right) = 3 \cdot \left(\begin{pmatrix} 8.36 \\ 4.56 \end{pmatrix} - \begin{pmatrix} 4.02 \\ 4.50 \end{pmatrix} \right) = \begin{pmatrix} 13.02 \\ 0.18 \end{pmatrix}$$

$$X''(0.6) = 6 \cdot \left(b_3^1 - 2b_2^1 + b_1^1 \right) = 6 \cdot \left(\begin{pmatrix} 10.00 \\ 3.80 \end{pmatrix} - 2 \cdot \begin{pmatrix} 5.9 \\ 5.7 \end{pmatrix} + \begin{pmatrix} 1.2 \\ 2.7 \end{pmatrix} \right) = \begin{pmatrix} -3.6 \\ -29.4 \end{pmatrix}$$

1.6 Spline Representations

Generalization

Often, a series of Bézier segments compose a spline curve. Therefore, the individual segments might be parameterized differently (e.g. on the interval $[a,b]$ instead of $[0,1]$) so that a more general Bézier segment can be defined as:

$$X(t) = \sum_{i=0}^{n} b_i B_i^n\left(\frac{1-a}{b-a}\right) = \sum_{i=0}^{n} b_i \binom{i}{n} \frac{(t-a)^i (b-t)^{n-i}}{(b-a)^n}$$

Caution: changing the parameterization also changes the derivative so that the p-th derivative gets scaled by $\frac{1}{(b-a)^p}$

1.6 Spline Representations

Integrating a Bézier segment

Let $X(t) = \sum_{i=0}^{n} b_i B_i^n\left(\frac{1-a}{b-a}\right)$ a Bézier segment defined on the interval [a,b]. The integral can then be computed by:

$$\int_a^b X(t)\,dt = \frac{b-a}{m+1}(b_0 + b_1 + \ldots + b_m)$$

Proof:

From $\int_0^1 B_i^m(t) = \frac{1}{m+1} \quad (i = 0, \ldots, m)$ we get

$$\int_a^b B\left(\frac{t-a}{b-a}\right)dt = \int_0^1 B(s)(b-a)\,ds \quad \text{after substituting with} \quad g(s) = (b-a)s + a$$

1.6 Spline Representations

Interpolating with Bézier segments

Interpolating with Bézier segments can be achieved using a system of linear equations, which is derived directly from the interpolating condition:

$$P_j = \sum_{i=0}^{n} b_i B_i^n(t_j) \qquad (j = 0, \ldots, n)$$

There is a unique solution to this system of linear equations. This is obviously the case since the solution could be computed using the Lagrange basis as well.

1.6 Spline Representations

Interpolating with Bézier segments

The system for interpolating the Points $P_0, \ldots, P_r$ using a Bézier segment is given by:

$$X(t_i) = \sum_{j=0}^{r} b_j B_j^r(t_i) \equiv P_i \qquad (i = 0, \ldots, r)$$

Example: $P_0 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ $P_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ $P_2 = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ $P_3 = \begin{pmatrix} 6 \\ 6 \end{pmatrix}$

Chordal parameterization (i.e. choosing the parameter intervals based on the Euclidean distance between the points): $[a,b]=[0,8.13]$; $t_0=0$; $t_1=1.42$; $t_2=3.66$; $t_3=8.13$

1.6 Spline Representations

Interpolating condition

$$X(t_0) = X(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \qquad X(t_1) = X(1.42) = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$X(t_2) = X(3.66) = \begin{pmatrix} 4 \\ 2 \end{pmatrix} \qquad X(t_3) = X(8.13) = \begin{pmatrix} 6 \\ 6 \end{pmatrix}$$

Plugging in the definition of a Bézier segment:

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \sum_{i=0}^{3} b_i B_i^3(0) = b_0 \qquad \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \sum_{i=0}^{3} b_i B_i^3(1.42)$$

$$\begin{pmatrix} 4 \\ 2 \end{pmatrix} = \sum_{i=0}^{3} b_i B_i^3(3.66) \qquad \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \sum_{i=0}^{3} b_i B_i^3(0) = b_3$$

1.6 Spline Representations

This then results in the following system of linear equations:

$$0.36\, b_1^x + 0.07\, b_2^x = 1.38$$

$$0.41\, b_1^x + 0.33\, b_2^x = 3.29$$

$$0.36\, b_1^y + 0.07\, b_2^y = -0.18$$

$$0.41\, b_1^y + 0.33\, b_2^y = 1.12$$

with the solution $b_1 = \begin{pmatrix} 2.5 \\ -1.51 \end{pmatrix}$ and $b_2 = \begin{pmatrix} 6.86 \\ 5.27 \end{pmatrix}$

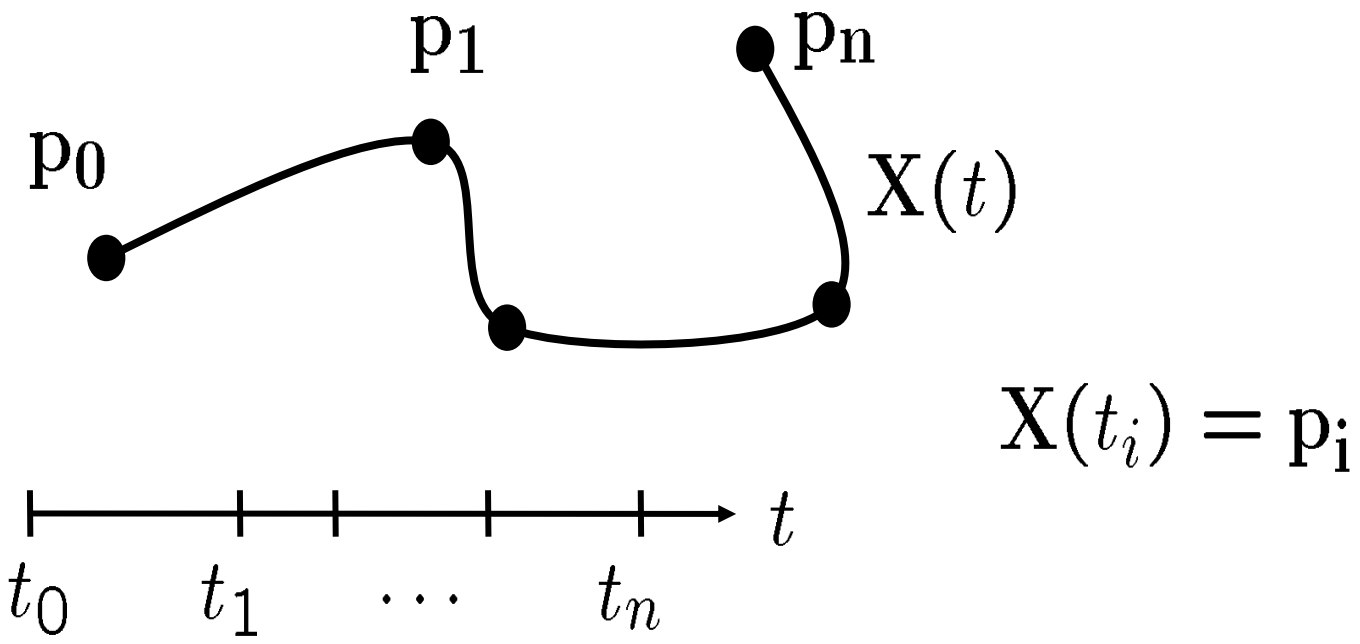
1.6 Spline Representations

Spline curves

Especially when using polynomials of higher degree for interpolation strong oscillating effects are the result. To control the waviness and to minimize the oscillating artifacts curves are often pieced together using several segments. These segments are then described by polynomials of lower degree (mostly three or five). The segments are usually defined in such a way that the transition between segments is “smooth”. Often, the segments are defined so that they connect two points that are to be interpolated to fulfill the interpolation condition $X(t) = p_i$.

1.6 Spline Representations

Example:



1.6 Spline Representations

Definition: C^k continuity

A function $f(t)$ is **C^k continuous** ($k \geq 0$) if the function itself and the first k derivatives are continuous. **$C^k[t_0,t_n]$** is the set of C^k continuous functions on the interval $[t_0,t_n]$.

Definition: Spline

Let $\tau=\{t_0,t_1,\ldots,t_n\}$ be a monotonic vector composed of nodes $t_i<t_{i+1}$. A function S is called **spline** of degree k-1 (of order k) if:

- S is a polynomial of degree k-1 in each of the intervals $[t_i,t_{i+1}]$
- S is C^{k-2} continuous on $[t_0,t_n]$

1.6 Spline Representations

Comments:

1) The spline S is called interpolating spline if $S(t_i)=p_i$ for a given set of points p_i.
2) The interpolating spline is generally not uniquely defined. There are k-2 additional degrees of freedom, i.e. further boundary conditions are required.

   For cubic splines ($k=4$) often the natural boundary conditions $S''(t_0) = 0$ and $S''(t_n) = 0$ are chosen.

   If the two nodes t_0 and t_n are associated with the same points, i.e. $p_0 = p_n$, then a closed spline is the result (without additional boundary conditions).

1.6 Spline Representations

Cubic splines

Instead of requiring a certain type of interpolating polynomials, e.g. a polynomial of a certain degree, we can demand properties, for example, a very smooth curve:

Therefore, we require that $\int_{t_0}^{t_n} \left\| g''(t) \right\|^2 dt$ is minimal

and use the boundary conditions

$$g(t_j) = p_j \quad (j = 0, \ldots, n), \quad g'(t_0) = p'_0, \text{ and } \quad g'(t_n) = p'_n$$

1.6 Spline Representations

Theorem: minimum norm property

Among all functions $g \in C^2[t_0, t_n]$ which fulfill the previous boundary conditions, the cubic spline is the function with the smallest value of $\int_{t_0}^{t_n} \left\| g''(t) \right\|^2 dt$. This is a well known fact from calculus of variations [de Boor, 1966].

There are different ways for using the additional degrees of freedom. The most common ones are:

$S''(t_0) = 0$ and $S''(t_n) = 0$ $\rightarrow$ natural spline

$S(t_0) = S(t_n), S'(t_0) = S'(t_n), S''(t_0) = S''(t_n)$ $\rightarrow$ periodic spline

1.6 Spline Representations

In order to better understand the algorithm for spline curves, we first start with computing the coefficients a_i, b_i, c_i, and d_i of a natural cubic spline:

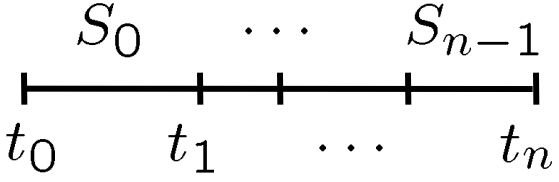
$$S(t) = S_i(t) = a_i + b_i(t-t_i) + c_i(t-t_i)^2 + d_i(t-t_i)^3$$

for $t \in [t_i, t_{i+1}]$; $i=0,\ldots,n-1$

This results in the following conditions for the polynomials (segments) S_i:

$$S_i(t) = p_i \quad i=0,\ldots,n$$

$$\left.\begin{aligned} S_i(t_i) &= S_{i-1}(t_i) \\ S_i'(t_i) &= S_{i-1}'(t_i) \\ S_i''(t_i) &= S_{i-1}''(t_i) \end{aligned}\right\} \; i=1,\ldots,n-1$$



1.6 Spline Representations

This means for the coeffcients of the individual segments:

$$
\begin{aligned}
a_i &= p_i && i=0,\ldots,n \\
a_i &= a_{i-1}+b_{i-1}(t_i-t_{i-1})+c_{i-1}(t_i-t_{i-1})^2+d_{i-1}(t_i-t_{i-1})^3 && i=1,..,n \\
b_i &= b_{i-1}+2c_{i-1}(t_i-t_{i-1})+3d_{i-1}(t_i-t_{i-1})^2 && i=1,\ldots,n-1 \\
c_i &= c_{i-1}+3d_{i-1}(t_i-t_{i-1})
\end{aligned}
$$

1.6 Spline Representations

By defining $\Delta_i := t_{i+1} - t_i$ we get the following after a few transformations:

$$c_{i-1}(\Delta_{i-1}) + c_i(2(\Delta_{i-1} + \Delta_i)) + c_{i+1}(\Delta_i)$$

$$= \frac{3}{\Delta_i}(a_{i+1} - a_i) - \frac{3}{\Delta_{i-1}}(a_i - a_{i-1}) \qquad i = 1, \ldots, n-1$$

$$d_i = \frac{1}{3\Delta_i}(c_{i+1} - c_i) \qquad i = 0, \ldots, n-1$$

$$b_i = \frac{1}{\Delta_i}(a_{i+1} - a_i) - \frac{\Delta_i}{3}(c_{i+1} + 2c_i) \quad i = 0, \ldots, n-1$$

1.6 Spline Representations

Since $a_i = p_i$, the coefficients a_i are well known. Hence, we have to solve a system of linear equations with n-1 equations and $n+1$ unknown variables c_i $(i=0,...,n)$.

For a natural cubic spline we use $S''(t_0) = S''(t_n) = 0$ which results in $c_0 = c_n = 0$.

In summary, we can use this criteria to formulate an algorithm:

1.6 Spline Representations

Algorithm for a natural cubic spline

Let t_i $(i=0,...,n)$ be a set of nodes with $t_0<t_1<...<t_n$ and $p_0,...,p_n$ a set of function values that are to be interpolated. We then look for the natural cubic spline S with the following representation:

$$S_i(t) = a_i+b_i(t-t_i)+c_i(t-t_i)^2+d_i(t-t_i)^3$$

for $t \in [t_i,t_{i+1}]$; $i=0,...,n-1$

We now need to describe an algorithm that computes the coefficients a_i, b_i, c_i, and d_i.

1.6 Spline Representations

Algorithm for a natural cubic spline (continued)

$$a_i = p_i$$

$$c_0 = c_n = 0$$

$$c_{i-1}(\Delta_{i-1}) + c_i(2(\Delta_{i-1} + \Delta_i)) + c_{i+1}(\Delta_i)$$

$$= \frac{3}{\Delta_i}(a_{i+1} - a_i) - \frac{3}{\Delta_{i-1}}(a_i - a_{i-1}) \qquad i = 1, \ldots, n-1$$

(linear equation system for c_i)

$$d_i = \frac{1}{3\Delta_i}(c_{i+1} - c_i) \qquad i = 0, \ldots, n-1$$

$$b_i = \frac{1}{\Delta_i}(a_{i+1} - a_i) - \frac{\Delta_i}{3}(c_{i+1} + 2c_i) \quad i = 0, \ldots, n-1$$

1.6 Spline Representations

Comments:

1) The previous system of linear equations can also be expressed in matrix form $Ac = b$:

$$A = \begin{pmatrix} 2(\Delta_0 + \Delta_1) & \Delta_1 & 0 & \cdots & \cdots \\ \Delta_1 & 2(\Delta_1 + \Delta_2) & \Delta_2 & \ddots & 0 \\ 0 & \Delta_2 & \ddots & \ddots & \Delta_{n-2} \\ \vdots & \vdots & 0 & \Delta_{n-2} & 2(\Delta_{n-2} + \Delta_{n-1}) \end{pmatrix}$$

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} \qquad b = 3 \cdot \begin{pmatrix} \frac{a_2 - a_1}{\Delta_1} - \frac{a_1 - a_0}{\Delta_0} \\ \vdots \\ \frac{a_n - a_{n-1}}{\Delta_{n-1}} - \frac{a_{n-1} - a_{n-2}}{\Delta_{n-2}} \end{pmatrix}$$

1.6 Spline Representations

2) The matrix A is tri-diagonal, symmetric, diagonal-dominant, positive definite, and also has positive elements. Hence, A is regular und there is a unique solution for the system of linear equations. For solving the system of linear equations the *LU* decomposition for tridiagonal matrices is appropriate due to its complexity of $O(n)$.

3) In case of a periodic cubic spline the additional boundary conditions is no longer necessary since the transistional conditions are already sufficient.

1.6 Spline Representations

Algorithm for a peridodic cubic spline

If the nodes t_0 and t_n are identified, i.e. $p_0 = p_n$ then the interpolating curve is closed and has a C^2 continuous transition at t_0 (or t_n, respectively). In this case, the matrix describing the system of linear equations changes to:

$$A = \begin{pmatrix} 2(\Delta_0 + \Delta_1) & \Delta_1 & \cdots & 0 & \Delta_0 \\ \Delta_1 & 2(\Delta_1 + \Delta_2) & \Delta_2 & \cdots & 0 \\ \vdots & \Delta_2 & \ddots & \ddots & \vdots \\ 0 & \vdots & \ddots & \ddots & \Delta_{n-1} \\ \Delta_0 & 0 & \cdots & \Delta_{n-1} & 2(\Delta_{n-1} + \Delta_0) \end{pmatrix}$$

1.6 Spline Representations

This matrix A is cyclic tri-diagonal, symmetric, diagonal-dominant, positive definite, and has only positive elements, i.e. A is well-conditioned. The system can again be solved with a complexity of $O(n)$.

Compared to the system for natural cubic splines, the matrix A has an additional row and an additional column due to the fact that there are n transitions between segments compared to n-1 in the previous case. In addition, there is an additional entry in the upper right and lower left corner of the matrix A because of the cyclic band-like structure.

1.6 Spline Representations

Since splines are often applied in CAD applications for modeling of 2-D or 3-D curves, we need the notion of vector-valued and parametric splines.

Definition: Let $\Delta=\{t_0,\ldots,t_n\}$, $a=t_0<t_1<\ldots<t_n=b$ a set of nodes subdividing the interval $[a,b]\subset IR$. Then, the mapping $X:[a,b]\rightarrow IR^3$ is called **parametric spline** of degree k-1 (order k) if the components (segments) of the spline are of degree k-1.

Particularly, the individual components x_i are C^{k-2} continuous, i.e. $x_i\in C^{k-2}[a,b]$ $(i=0,\ldots,n)$ which we abbreviate as $X\in C^{k-2}[a,b]$.

1.6 Spline Representations

A common criterion for local smoothness of three-dimensional curves, which are composed of concatenated segments is defined using derivatives:

Definition: If two parametric curves $X:[t_0,t_1]\rightarrow IR^3$ and $Y:[s_0,s_1]\rightarrow IR^3$ with $X\in C^m[t_0,t_1]$ and $Y\in C^m[s_0,s_1]$ and if both curves have the point $X(t_1)=Y(s_0)$ in common we call the transition between the two segments C^k if:

$$\frac{d^r}{dt^r}X(t_1) = \frac{d^r}{ds^r}Y(s_0)$$

For all r with $1\leq r\leq k$.

(If none of the derivatives are equal we call the transition C^0)

1.6 Spline Representations

Algorithm for parametric cubic curves

Let $p_i=(x_i,y_i,z_i),\ i=0,\ldots,n$ a set of 3-D points. We are then looking for an interpolating parametric cubic spline.

There are basically three steps for finding the interpolating curve:

Step 1: **parameterization**: Define the parameter values (nodes) $t_i\ (i=0,\ldots,n)$ corresponding to the points that are to be interpolated.

Step 2: choose **boundary conditions**

Step 3: Compute the spline curves for each component S_x, S_y, S_z individually, such that $S_x(t_i)=x_i,\ S_y(t_i)=y_i,\ S_z(t_i)=z_i$ using the previously defined spline algorithm.

1.6 Spline Representations

$$S_x(t)=S_{x,i}(t)=a_{x,i}+b_{x,i}(t-t_i)+c_{x,i}(t-t_i)^2+d_{x,i}(t-t_i)^3$$

$$S_y(t)=S_{y,i}(t)=a_{y,i}+b_{y,i}(t-t_i)+c_{y,i}(t-t_i)^2+d_{y,i}(t-t_i)^3$$

$$S_z(t)=S_{z,i}(t)=a_{z,i}+b_{z,i}(t-t_i)+c_{z,i}(t-t_i)^2+d_{z,i}(t-t_i)^3$$

$$t\in[t_i,t_{i+1}],\ i=0,\ldots,n$$

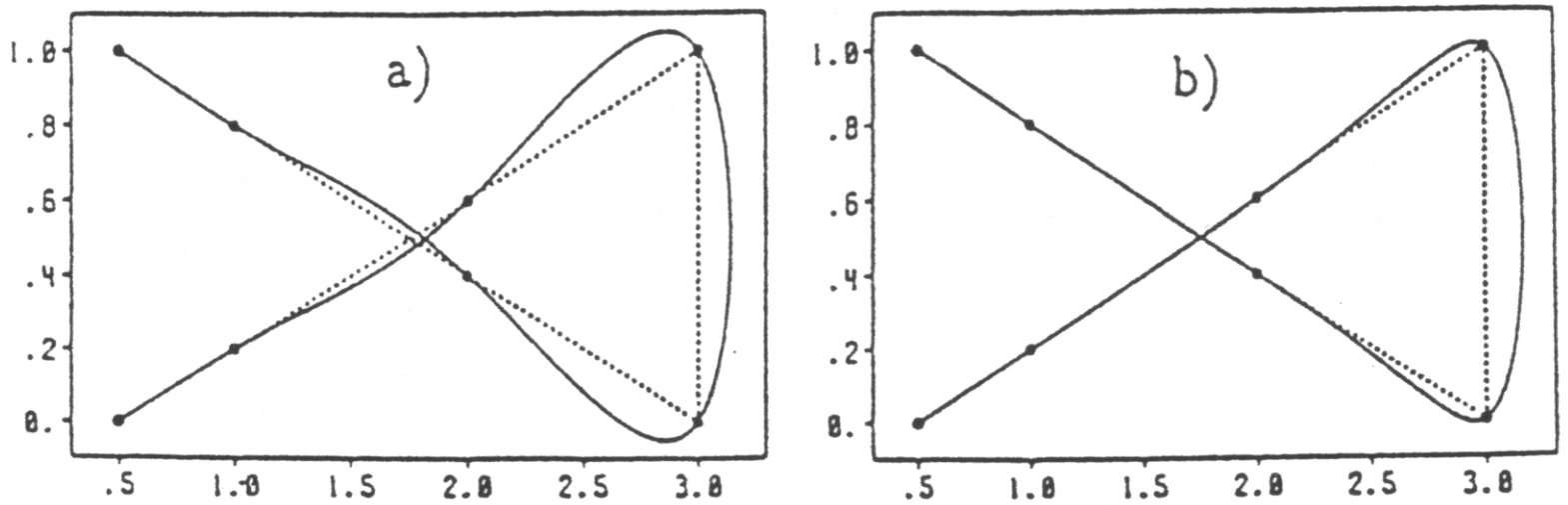
Comments:

For closed parametric curves, periodic splines can be used, if the curve is supposed to be smooth at every point. If there are cusps (C^0 transitions, e.g. the cross section of a wing of an airplane) then natural splines can be used with the cusp as start and end point.

1.6 Spline Representations

Parameterization

The choice of parameterization has great influence on the shape of the curve and therefore on the quality of the resulting curve (and surface).



Interpolation problem with different parameterizations

1.6 Spline Representations

Parameterizations

The effect of the chosen parameterization can be illustrated using the following analogon:

Interpret the parameter t as the time which determines how long it takes for a point X to walk along the curve $X(t)$.

We now introduce a few different parameterizations. Therefore, we assume that a curve interpolates a set of $n+1$ points on the parameter interval $[a,b]$.

1.6 Spline Representations

Equidistant parameterization

The equidistant parameterization assigns the same amount of time to each segment connecting the points (p_i,p_{i+1}).

$$\Delta_t = \frac{b - a}{n} \quad ; \qquad t_i = a + i \quad ; \qquad i = 0,\ldots, n$$

If the distances between consecutive points differs a lot, then the point X has to walk along the curve $X(t)$ at different speeds. For example, if a large distance between two points is followed by a short distance then the point X has to slow down which results in an "overshooting" effect.

1.6 Spline Representations

Chordal parameterization

The parameterization should reflect the “structure of the point set”. This can be achieved by using the chordal parameterization:

$$\Delta t_i = t_{i+1} - t_i = \frac{\left\| P_{i+1} - P_i \right\|}{s}$$

The parameter intervals are chosen proportional to the distances between two consecutive points and then normalized using a constant factor s (e.g. s if often set to the overall length of the polygon defined by the set of points that are to be interpolated)

1.6 Spline Representations

Centripetal parameterization

Another possibility for choosing a parameterization, which reflects the structure of the data, is the so called centripetal parameterization [Lee 1975]:

$$\Delta t_i = \frac{\sqrt{\left\| P_{i+1} - P_i \right\|}}{s}$$

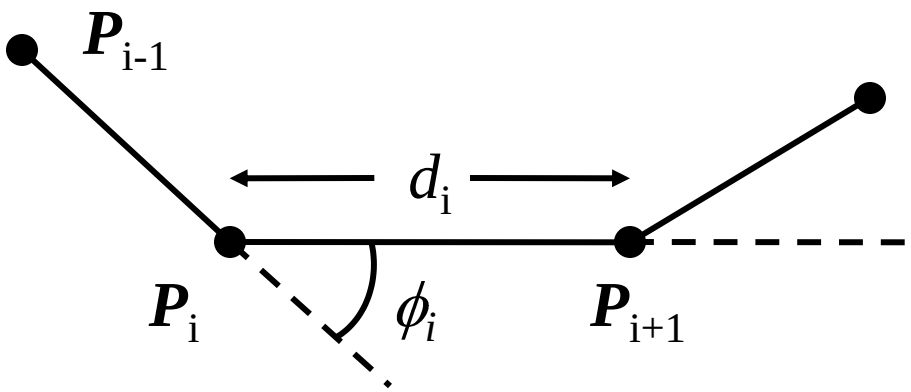
Here, the centripetal acceleration is approximately minimized. You can also combine the different types of parameterizations. A parameterization which not only considers distances but also takes the angular changes of the interpolated points into account was developed by T. Foley (see [Foley 1989])

1.6 Spline Representations

Foley parameterization

$$\Delta t_i = d_i \left(1 + 1.5 \cdot \frac{\phi_i \cdot d_{i-1}}{d_{i-1} + d_i} + 1.5 \cdot \frac{\phi_{i+1} \cdot d_{i+1}}{d_i + d_{i+1}} \right)$$

where $\phi_i = \min \left(\pi - \phi_i, \frac{\pi}{2} \right)$



As distance function we can use the Euclidian metric of the affine-invariant Nielson-Metric.

1.6 Spline Representations

Nielson metric

$$\left\|(x,y)\right\|_N^2 = (x,y)\cdot\begin{pmatrix} \frac{\sum y^2}{\Delta} & \frac{\sum xy}{\Delta} \\ \frac{-\sum xy}{\Delta} & \frac{\sum x^2}{\Delta} \end{pmatrix}\cdot(x,y)$$

with $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$; $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$;

$$\sum x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \; ; \quad \sum y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \; ;$$

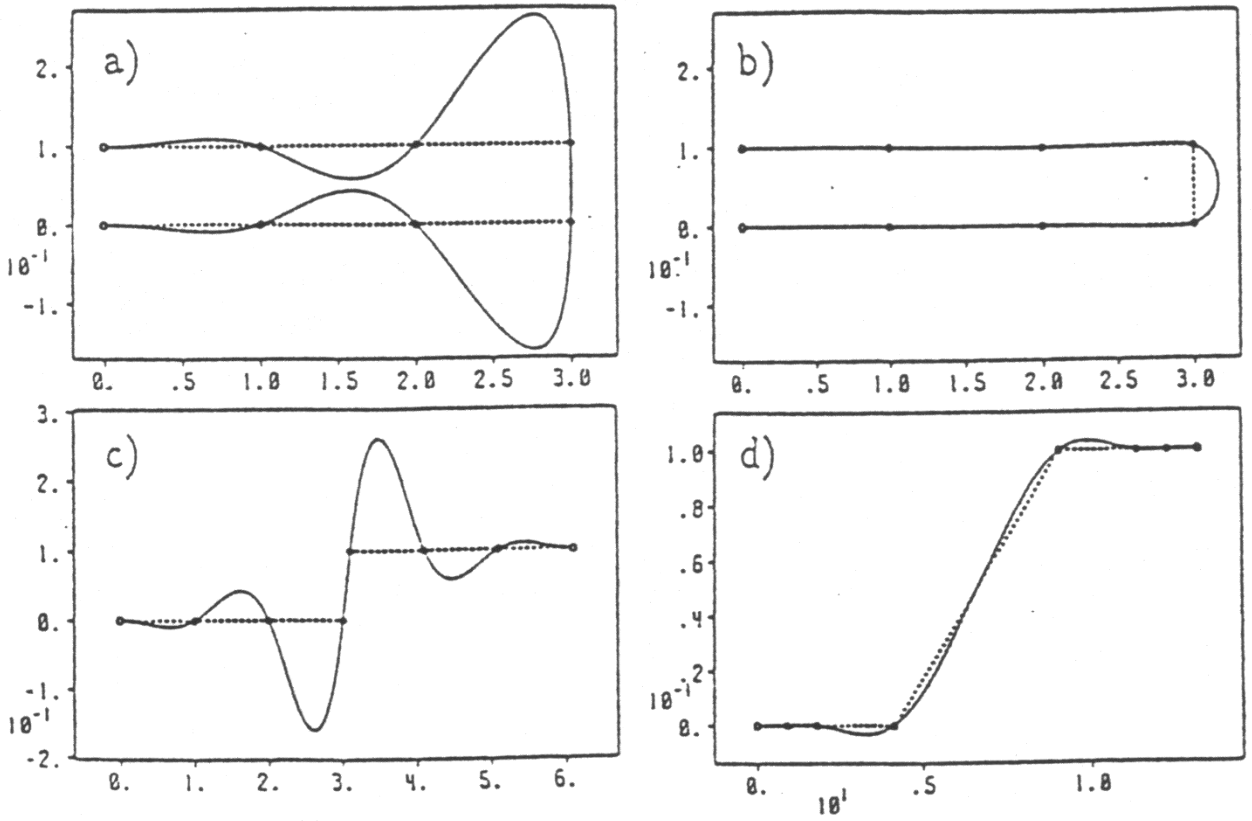
$$\sum xy = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})\cdot(y_i - \bar{y}) \; ; \quad \Delta = \sum x^2 \sum y^2 - \left(\sum xy\right)^2$$

1.6 Spline Representations

The Nielson metric first scales the set of points in such a way that the variance in all directions is unified (see also Principal Component Analysis). The distances are derived from the scaled arrangement of the points. This metric allows for a complete independence from coordinate systems and scaling (i.e. it is affine invariant).

The effects of different parameterizations will be illustrated using the following figures which show interpolation of the same set of points with different parameterizations:

1.6 Spline Representations



Parametric spline $(x(t), y(t))$ using a) chord length and b) affine invariant angle. Function component $y(t)$ using c) chord length and d) affine invariant angle on the y-data in a) and b).

1.6 Spline Representations

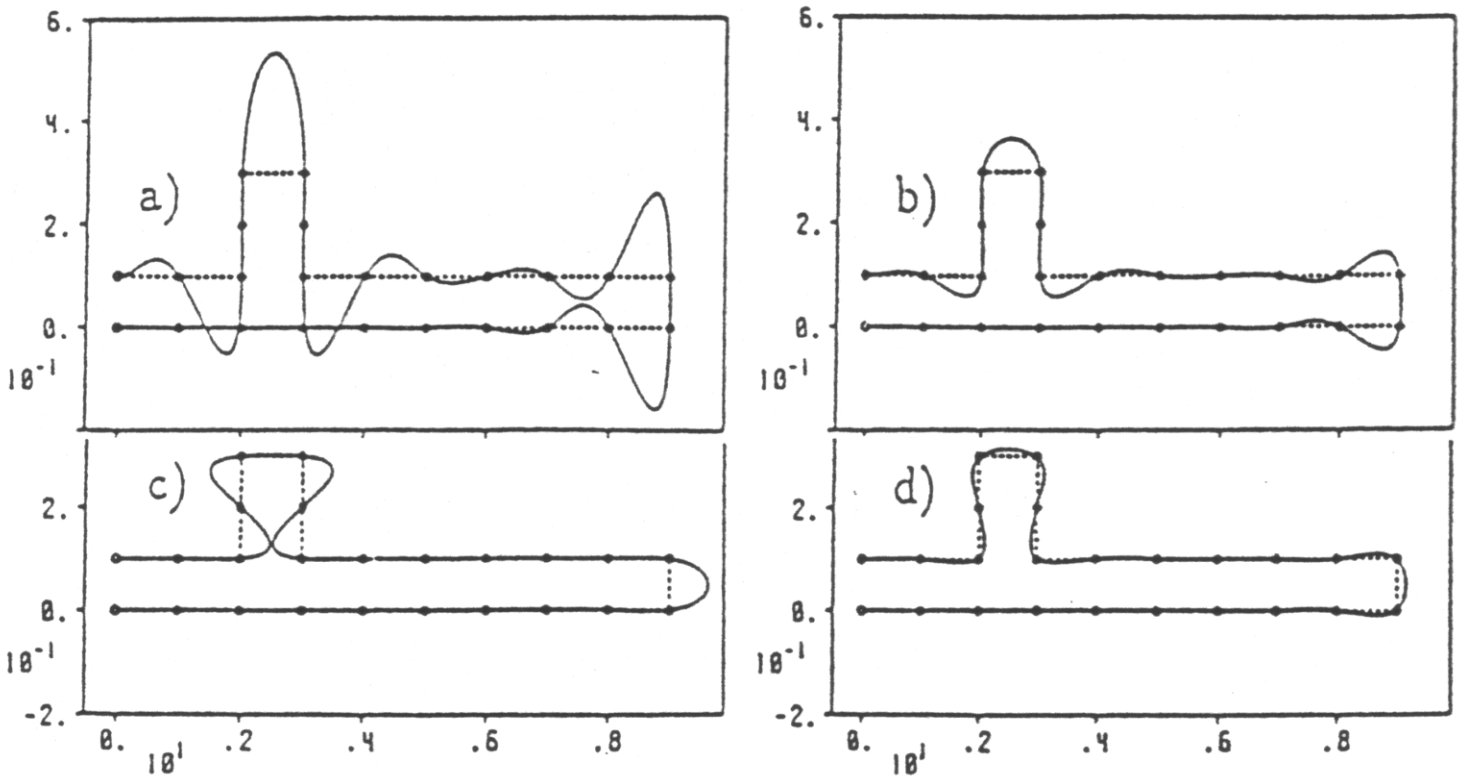


Figure 3. a) chord length, b) centripetal, c) affine invariant chord, d) affine invariant angle

1.6 Spline Representations

Parameter transformation

The parameterization of the interpolating points has great influence on the overall shape of the resulting curve. However, it is possible to change the parameterization afterwards without changing the shape of the curve.

Definition:

Let $X(t)$ be a curve and $\varphi(t)$ a bijective and continuous function. Then $Y(t)=X(\varphi(t))$ also defines a curve which results from X after **parameter transformation**.

If both φ and φ^{-1} are continuous differentiable we call φ a C^1 parameter transformation.

1.6 Spline Representations

Example: The length of an arc is

$$L(t_0, t_1) = \int_{t_0}^{t_1} \left\| \dot{X}(t) \right\| dt \quad , \qquad \dot{X} = \frac{dX}{dt}$$

A curve can then be re-parameterized in such a way that $\left\| \dot{X}(t) \right\| = 1$ and t represents the arc length of the curve.

1.6 Spline Representations

Bézier splines

Instead of calculating each segment by solving a system of linear equations we can also define the segments of a spline using Bézier segments. For example for a cubic spline, each segment is defined by four Bézier points. In order to achieve a smooth spline the Bézier points should be chosen in such a way that the transition between segments fulfills the continuity criterion, i.e. for a cubic spline the resulting curve should be C^2 continuous.

1.6 Spline Representations

Bézier splines

Bézier segments have the nice property that the derivatives can be computed easily from the locations of the Bézier points (control points). This property can be used to define a C^2 continuous cubic spline:

Let $t_0,\ldots,t_n$; $t_0<t_1<\ldots<t_n$ be a set of nodes and $p_0,\ldots,p_n$ the points that we want to interpolate. Then, we can use the following equation to define the Bézier segments of a Bézier spline of degree m :

$$S_i(t) = \sum_{l=0}^{m} b_{m\cdot i+l} B_l^m \left(\frac{t - t_i}{t_{i+1} - t_i} \right)$$

1.6 Spline Representations

Now, we have to define the Bézier points in such a way that the interpolation condition is fulfilled and the transition between segments is smooth.

From the interpolation condition and the fact that each Bézier segment interpolates the end points of the control polygon we get: $b_{m \cdot i} = p_i \ (i=0,\ldots,n)$.

We can also compute the derivatives using the formulas we derived before:

1.6 Spline Representations

$$\Delta_i = t_{i+1} - t_i$$

$$S_i'(t_{i+1}) = \frac{m}{\Delta_i}(b_{m(i+1)} - b_{m(i+1)-1})$$

$$S_i''(t_{i+1}) = \frac{m(m-1)}{\Delta_i^2}(b_{m(i+1)} - 2b_{m(i+1)-1} + b_{m(i+1)-2})$$

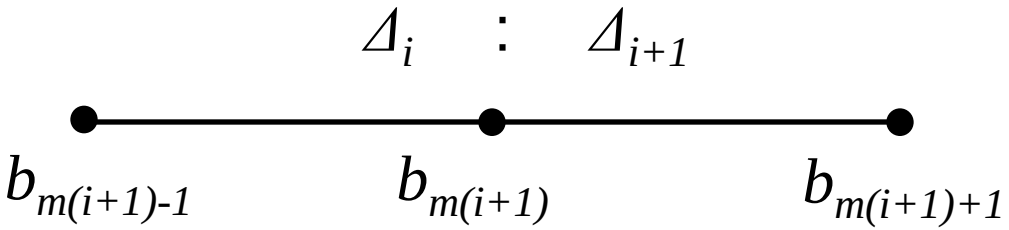
$$S_{i+1}'(t_{i+1}) = \frac{m}{\Delta_{i+1}}(b_{m(i+1)+1} - b_{m(i+1)})$$

$$S_{i+1}''(t_{i+1}) = \frac{m(m-1)}{\Delta_{i+1}^2}(b_{m(i+1)+2} - 2b_{m(i+1)+1} + b_{m(i+1)})$$

1.6 Spline Representations

For a C^1 transition we need that $S'_i(t_{i+1}) = S'_{i+1}(t_{i+1})$, i.e.:

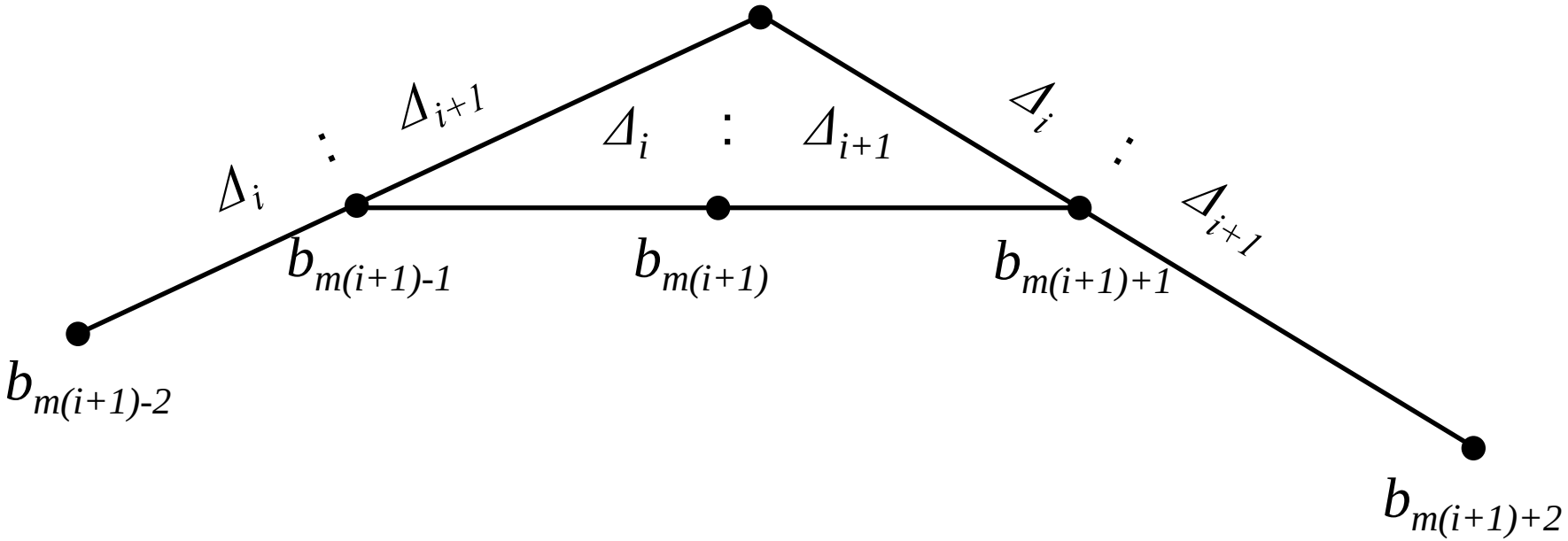
$$\frac{m}{\Delta_i}\left(b_{m(i+1)} - b_{m(i+1)-1}\right) = \frac{m}{\Delta_{i+1}}\left(b_{m(i+1)+1} - b_{m(i+1)}\right)$$



1.6 Spline Representations

For a C^2 transition we need that $S''_i(t_{i+1}) = S''_{i+1}(t_{i+1})$, i.e.:

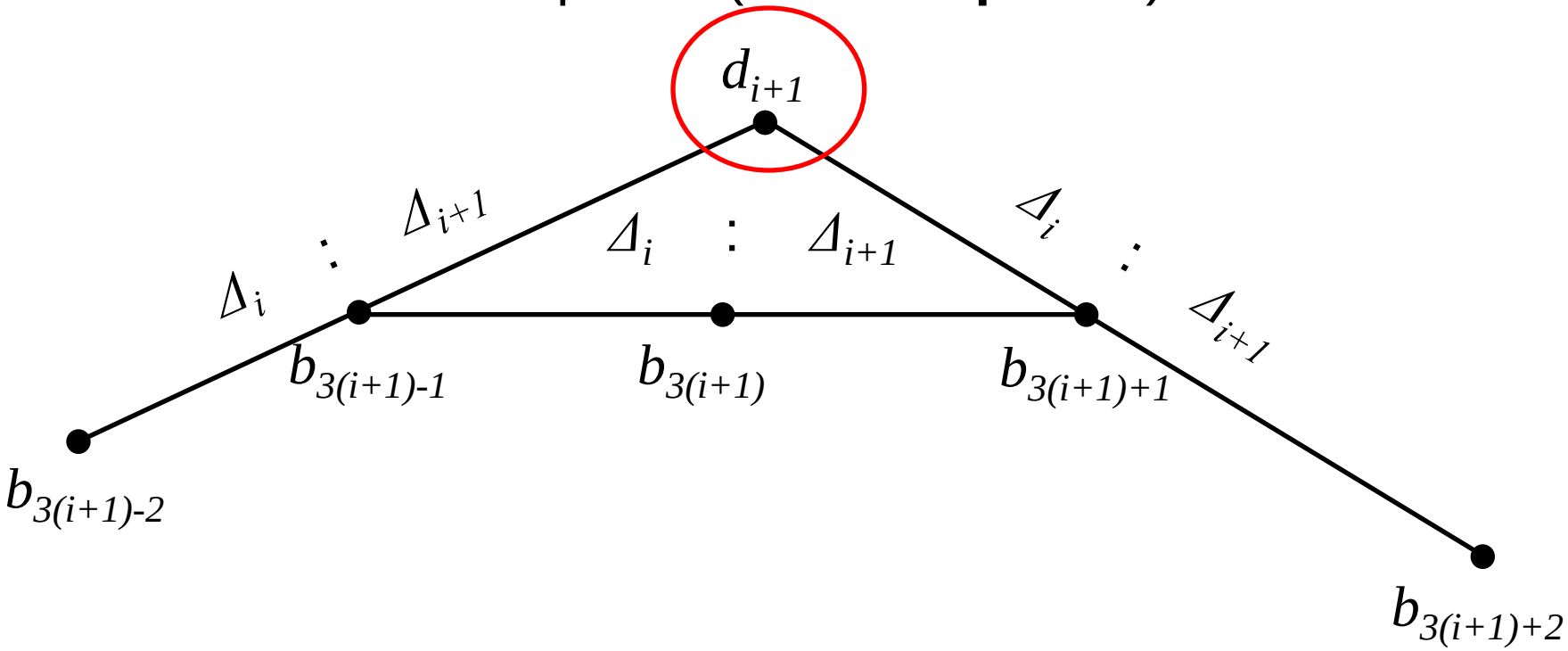
$$\frac{m(m-1)}{\Delta_i^2}\left(b_{m(i+1)} - 2b_{m(i+1)-1} + b_{m(i+1)-2}\right) = \frac{m(m-1)}{\Delta_{i+1}^2}\left(b_{m(i+1)+2} - 2b_{m(i+1)+1} + b_{m(i+1)}\right)$$



1.6 Spline Representations

How can we solve the interpolation problem with, for example, a cubic Bézier spline?

Introduce additional points (**de Boor points**):



1.6 Spline Representations

From the previous cartoon we can derive the following three equations for an interpolating cubic Bézier spline:

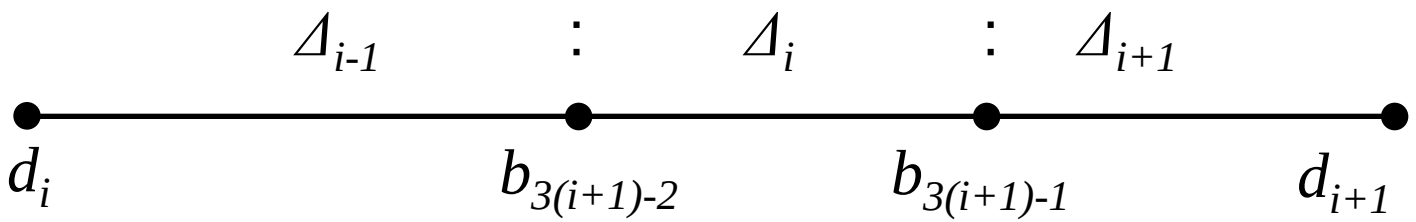
$$1)\ b_{3(i+1)+1} = b_{3(i+1)} + \frac{\Delta_{i+1}}{\Delta_i}(b_{3(i+1)} - b_{3(i+1)-1})$$

$$2)\ d_{i+1} = b_{3(i+1)-1} + \frac{\Delta_{i+1}}{\Delta_i}(b_{3(i+1)-1} - b_{3(i+1)-2})$$

$$3)\ b_{3(i+1)+2} = b_{3(i+1)+1} + \frac{\Delta_{i+1}}{\Delta_i}(-d_{i+1} + b_{3(i+1)+1})$$

1.6 Spline Representations

If we take a closer look at the newly introduced points we can see that there is a straight connection between two Bézier points and these de Boor points:



This gives us:

$$(\Delta_i + \Delta_{i+1} + \Delta_{i-1})\, b_{3(i+1)-2} = (\Delta_i + \Delta_{i+1})\, d_i + \Delta_{i-1} d_{i+1}$$

and similarily

$$(\Delta_i + \Delta_{i+1} + \Delta_{i-1})\, b_{3(i+1)-1} = \Delta_{i+1} d_i + (\Delta_{i-1} + \Delta_i)\, d_{i+1}$$

1.6 Spline Representations

Due to the fact that

$$(\Delta_i + \Delta_{i+1}) b_{3(i+1)} = \Delta_{i+1} b_{3(i+1)-1} + \Delta_i b_{3(i+1)+1}$$

we can substitute with the previous result :

$$(\Delta_i + \Delta_{i+1} + \Delta_{i-1})(\Delta_i + \Delta_{i+1}) b_{3(i+1)} = \Delta_{i+1}(\Delta_{i+1} d_i + (\Delta_{i-1} + \Delta_i) d_{i+1}) +$$
$$\Delta_i((\Delta_{i+1} + \Delta_{i+2}) d_{i+1} + \Delta_i d_{i+2})$$

Note :

For the substituti on we used the fact that $b_{3(i+2)-2} = b_{3(i+1)+1}$.

1.6 Spline Representations

Overall, we get the following system of linear equations:

$$(\Delta_i + \Delta_{i+1} + \Delta_{i-1}) b_{3(i+1)-2} = (\Delta_i + \Delta_{i+1}) d_i + \Delta_{i-1} d_{i+1}$$

$$(i = 0, \ldots, n-2)$$

$$(\Delta_i + \Delta_{i+1} + \Delta_{i-1})(\Delta_i + \Delta_{i+1}) b_{3(i+1)} = \Delta_{i+1}(\Delta_{i+1} d_i + (\Delta_{i-1} + \Delta_i) d_{i+1}) + \Delta_i((\Delta_{i+1} + \Delta_{i+2}) d_{i+1} + \Delta_i d_{i+2})$$

$$(i = 0, \ldots, n-2)$$

$$(\Delta_i + \Delta_{i+1} + \Delta_{i-1}) b_{3(i+1)-1} = \Delta_{i+1} d_i + (\Delta_{i-1} + \Delta_i) d_{i+1}$$

$$(i = 1, \ldots, n-1)$$

Note: set $\Delta_{-1} = \Delta_n = 0$

1.6 Spline Representations

Then, we can derive the three steps for computing an interpolating cubic Bézier spline:

1) Choose the boundary condition, i.e. the Bézier points b_1 and $b_{n \cdot m-1}$.
2) Determine the location of the de Boor points based on a system of linear equations as previously shown.
3) Using the previous equations 1) and 3) the missing Bézier points can be calculated.

1.6 Spline Representations

Special case:

Cubic Bézier spline with equidistant parameterization, such that $\Delta_i = 1$.

In this case, the system of linear equations reduces to:

$$3b_{3(i+1)-2} = 2d_i + d_{i+1} \qquad (i = 0,..., n-2)$$

$$6b_{3(i+1)} = d_i + 4d_{i+1} + {}_i\, d_{i+2} \quad (i = 0,..., n-2)$$

$$3b_{3(i+1)-1} = d_i + 2d_{i+1} \qquad (i = 1,..., n-1)$$

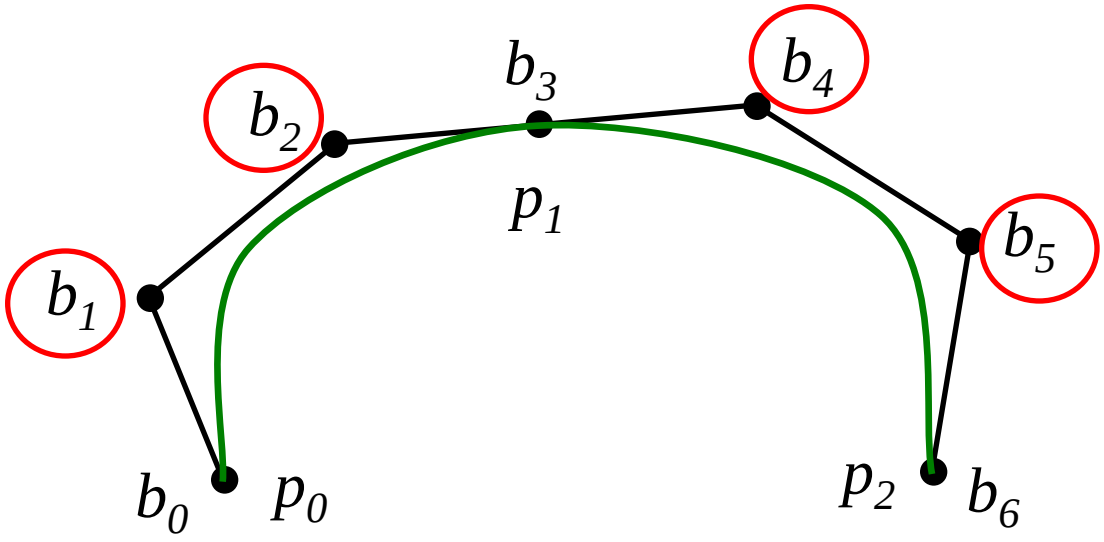
The matrix of this system is tri-diagonal; hence, it can be solved efficiently using the LU decomposition.

1.6 Spline Representations

Example:

Interpolation of three points p_0, p_1, and p_2 using a cubic Bézier spline with two segments ($\Delta_1=\Delta_2=1$), such that the Bézier spline has the given points as end points.

(i) $b_0 = p_0, b_3 = p_1, b_6 = p_2$



1.6 Spline Representations

(ii) Pick the additional degrees of freedom, i.e. b_1 and b_5

(iii) Determine the de Boor points d_0, d_1, and d_2 using the system of linear equations:

$$3b_1 = 2d_0 + d_1$$

$$6b_3 = d_0 + 4d_1 + d_2$$

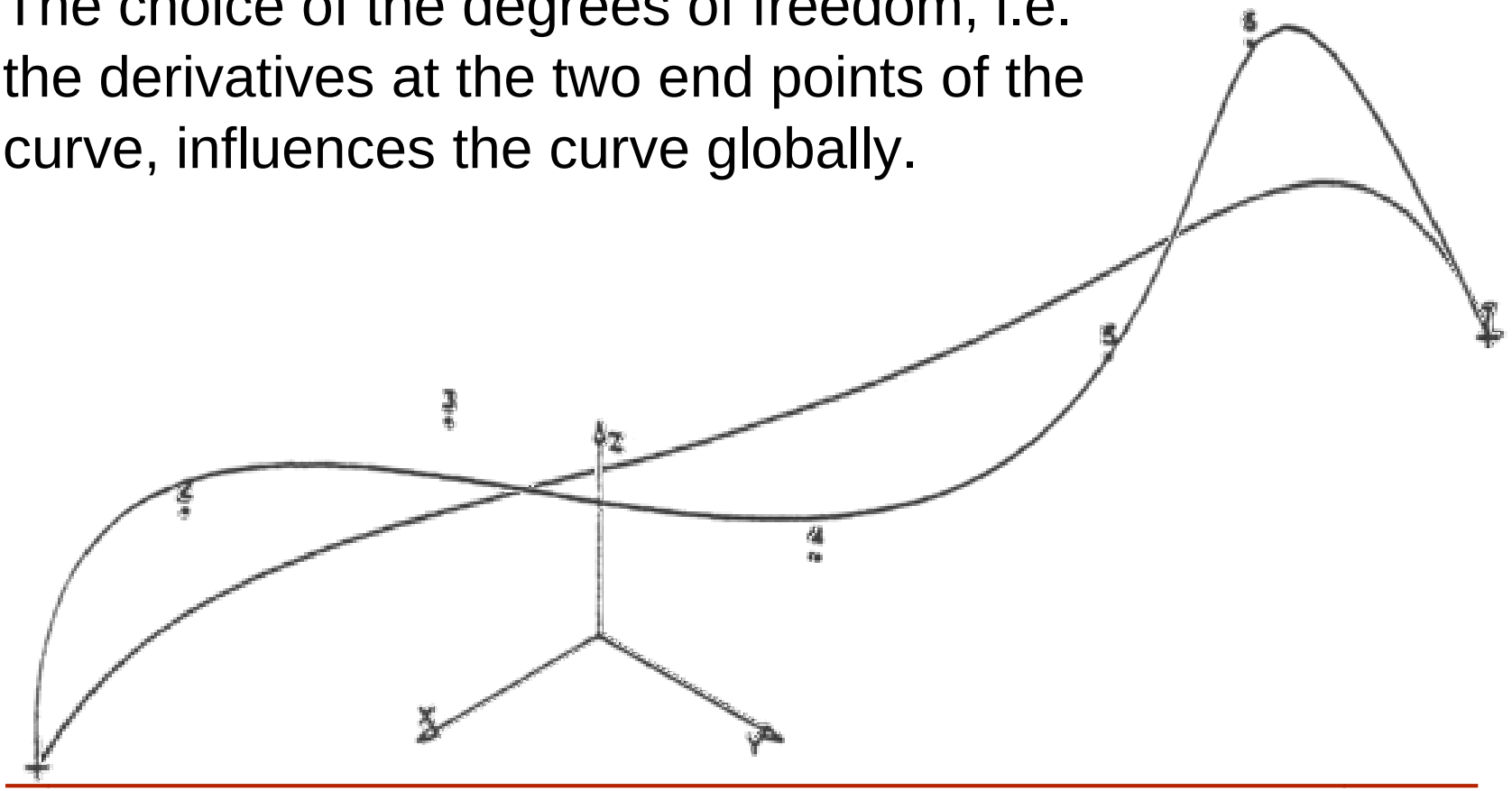
$$3b_5 = d_1 + 2d_2$$

(iv) Using previous equations 1) and 3), calculate the missing Bézier points b_2 and b_4.

1.6 Spline Representations

Comment:

The choice of the degrees of freedom, i.e. the derivatives at the two end points of the curve, influences the curve globally.



1.6 Spline Representations

B-splines

Motivation:

We would like a spline curve that has two advantages over Bézier splines:

1) Control points should have only local influence on the resulting curve
2) A continuous transition should be guarantied automatically without having to place the control points accordingly.

Idea: find a new set of basis functions.

1.6 Spline Representations

For a control polygon described by points $d_0,...,d_n$, define blending functions as new basis functions. First, we need to define a **knot vector**:

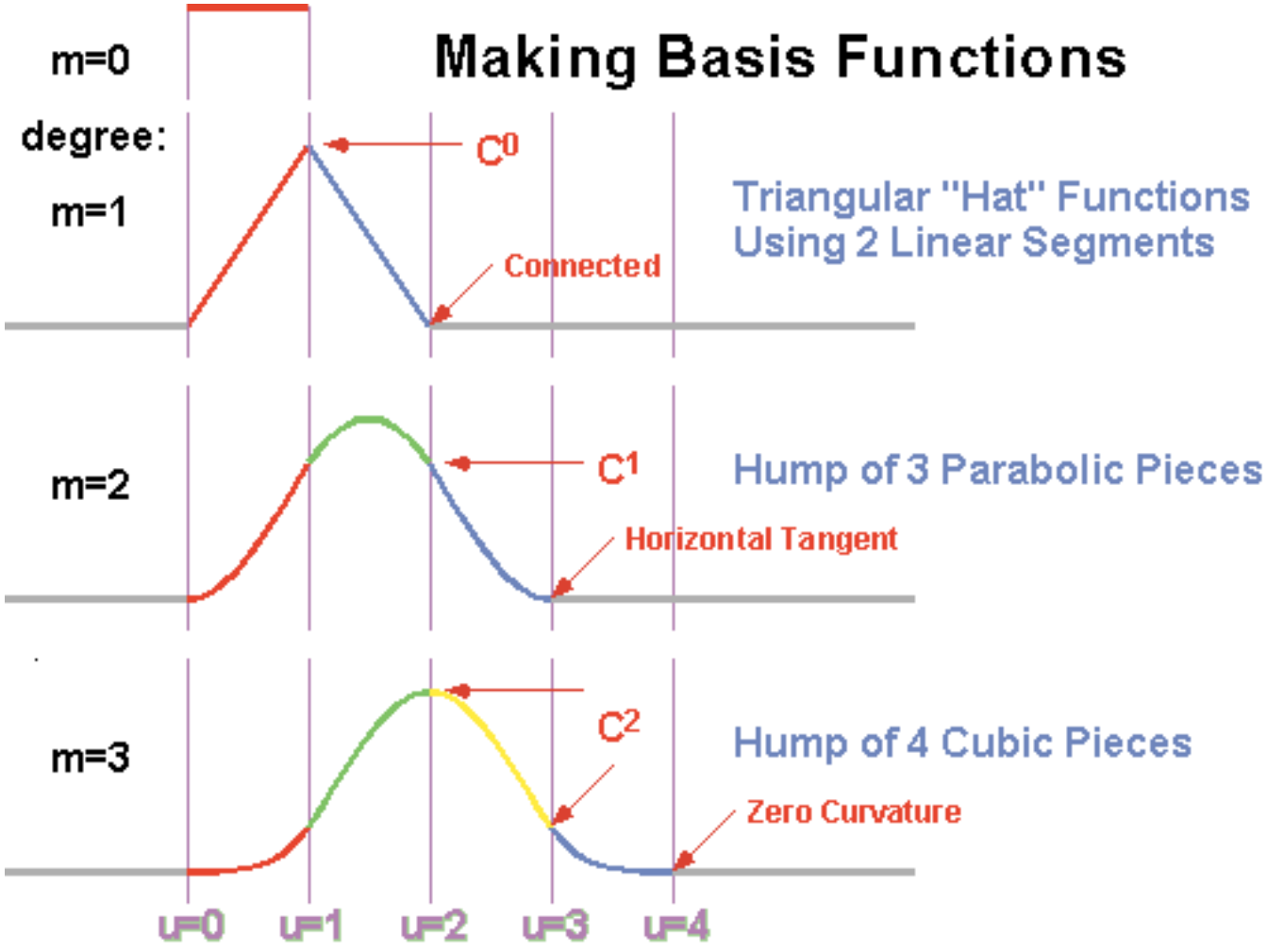
$$(u_0, ..,u_{n+m});\ u_i \leq u_{i+1}$$

Then, the new basis functions are recursively defined as:

$$N_{k,1}(u) = \begin{cases} 1 & \text{if } u_k \leq u \leq u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{k,m}(u) = \frac{u - u_k}{u_{k+m-1} - u_k} N_{k,m-1}(u) + \frac{u_{k+m} - u}{u_{k+m} - u_{k+1}} N_{k+1,m-1}(u)$$

1.6 Spline Representations



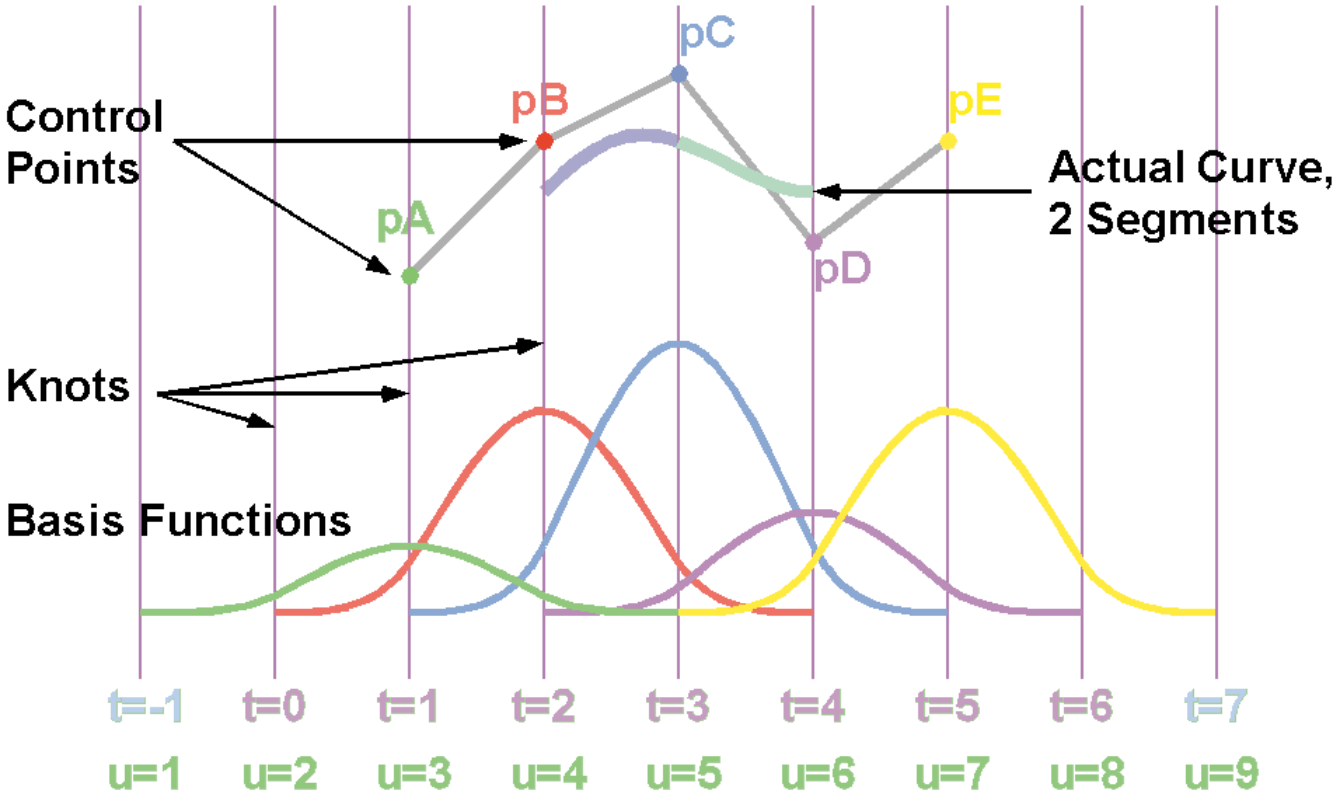
1.6 Spline Representations

Based on these blending functions a **B-spline curve** of degree m-1 (order m) with control points $d_0,\ldots,d_n$, where $n>m$ is defined as:

$$X(u) = \sum_{i=0}^{n} d_i N_{i,m}(u) \qquad u_{m-1} \le u \le u_{n+1}$$

1.6 Spline Representations

Cubic (4-th Order) B-Spline Basics



SLIDE: order 4; controlpointlist (pA pB pC pD pE); {uses knots 9}

1.6 Spline Representations

Comments:

1) The knot vector can have individual knots with multiplicity greater than one, i.e. the same value can appear more than once.
2) The resulting B-spline has degree m-1 is C^{m-2} continuous. At knots with multiplicity M this reduces to C^{m-1-M} continuity.
3) If $n=m$-1 and a knot vector with multiplicity m at both ends these blending functions resemble the Bernstein polynomials.

1.6 Spline Representations

4) Each section of the B-spline (between two successive knot values) is influenced by m control points.
5) Any control point can affect the shape of at most m curve sections.
6) For any $u_{m-1} \leq u \leq u_{n+1}$, the sum of all blending functions equal to one: $\sum N_{k,m}(u)=1$. Hence, B-splines fulfill the convex hull property just like Bézier splines.
7) The B-spline is called uniform if the knot values are spaced uniformly, i.e. equidistant (except multiplicities at both ends of the know vector). Otherwise we speak of a nonuniform B-spline.

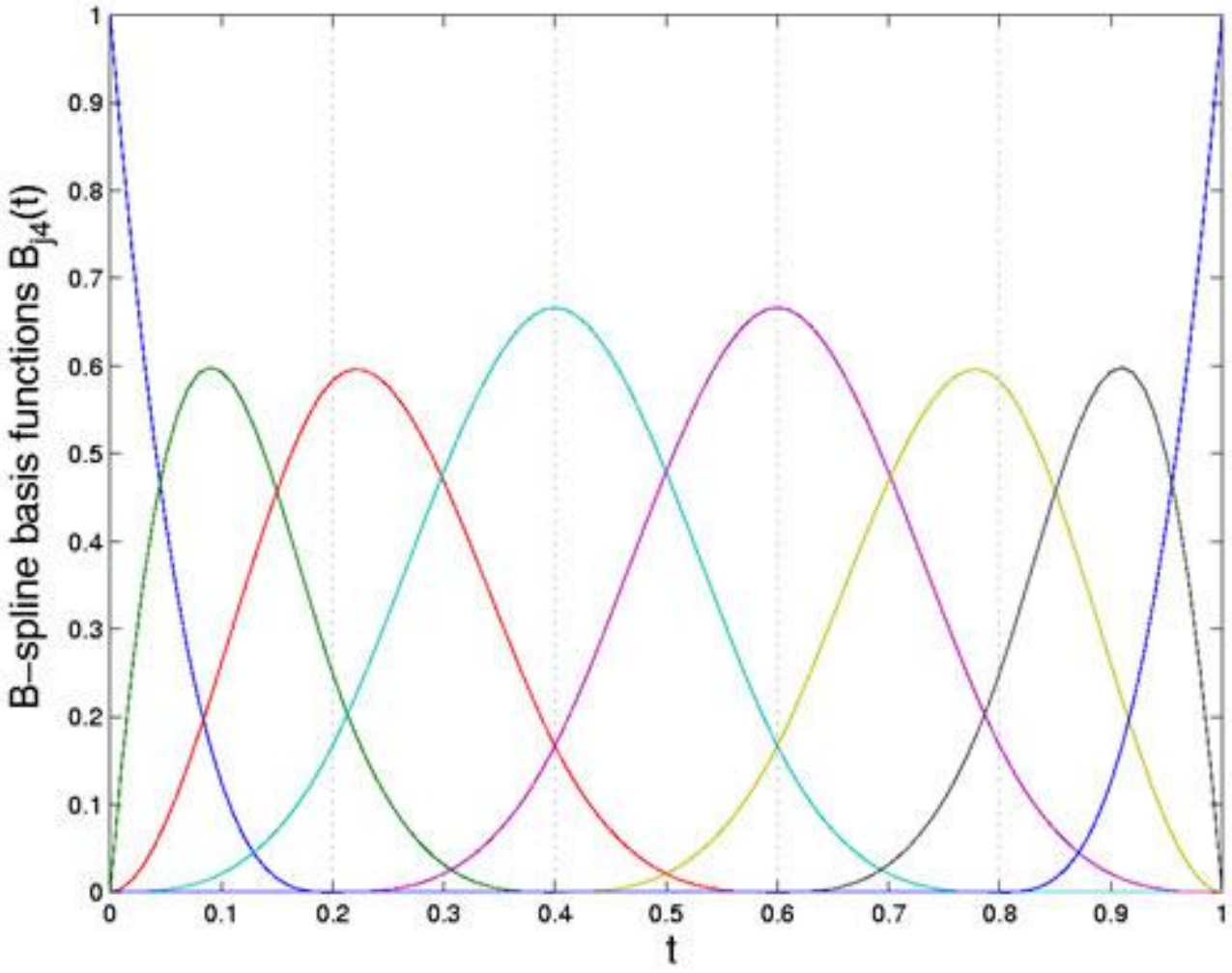
1.6 Spline Representations

Why multiple knot values?

As noted before, the multiplicity of a knot value reduces the continuity accordingly. Hence, it allows you to create design features, such as cusps.

In addition, often times multiplicity m is desired at both ends of the knot vector. To see why, we should take a closer look at the resulting blending functions:

1.6 Spline Representations



1.6 Spline Representations

Effect of multiplicity of m at both ends of knot vector

Since the blending functions at the ends of the interval equals to one, the resulting B-spline curve interpolates the two end points of the control polygon. Otherwise the B-spline would stop before reaching the end points of the control polygon.

This feature is usually only desired for “open” B-splines. With periodic B-splines, mostly this multiplicity at the end of the knot vector is not used to achieve C^{m-2} continuity at the end points of the control polygon

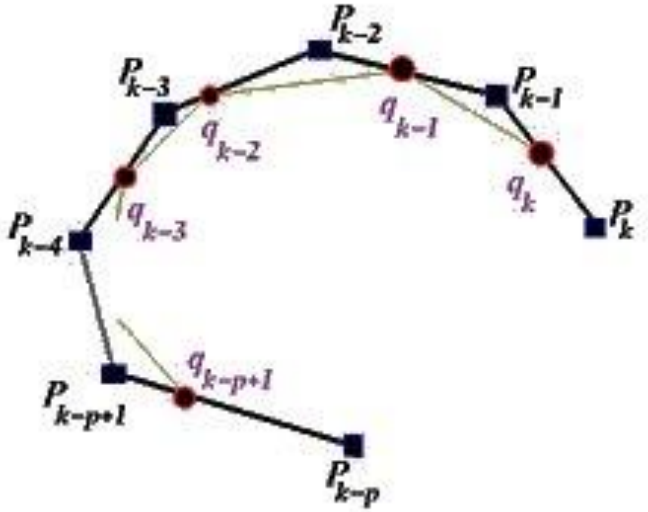
1.6 Spline Representations

Knot insertion

Sometimes it is useful to be able to insert an additional knot into the knot vector without changing the shape of the curve.

First, we have to identify in which segment the new knot t is located in, i.e. $t \in [u_k, u_{k+1})$. This means that the point $X(t)$ lies in the convex hull defined by the points $d_k, \ldots, d_{k-m}$. Consequently, we have to find new control points that replace $d_{k-1}, \ldots, d_{k-m+1}$ without changing the shape of the curve.

1.6 Spline Representations



The new points can be computed easily using the formula:

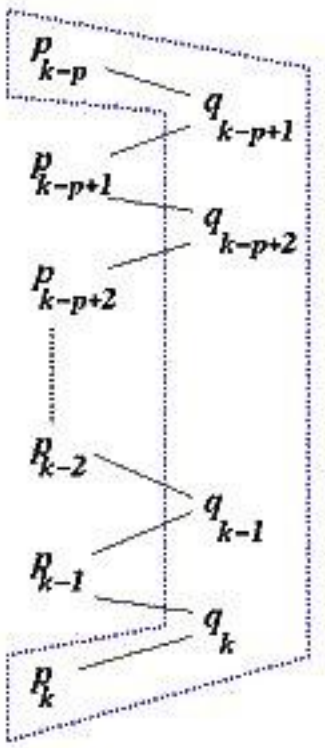
$$q_i = (1 - a_i) d_{i-1} + a_i d_i$$

where $$a_i = \frac{t - u_i}{u_{i+m-1} - u_i}$$ for $k - m + 1 \le i \le k$

Note: $P_i = d_i$

1.6 Spline Representations

The new B-spline then has the knot vector $u_0, \ldots, u_k, t, u_{k+1}, \ldots, u_{n+m}$ and the control polygon consists of the points $d_0, \ldots, d_{k-m}, q_{k-m+1}, \ldots, q_k, d_k, \ldots, d_n$ resulting in the following replacement scheme:



Note: $p = m$; $p_i = d_i$

1.6 Spline Representations

Example:

Let $(u_0,\ldots,u_{11})$ be a knot vector defined as $(0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1)$. We would like to insert an additional knot at $t=0.5$ which lies in the interval $[u_5,u_6)$. Hence, the affected control points are d_5, d_4, d_3, and d_2. Then, the new control points are determined:

$$a_5 = \frac{t - u_5}{u_8 - u_5} = \frac{0.5 - 0.4}{1 - 0.4} = \frac{1}{6}$$

$$a_4 = \frac{t - u_4}{u_7 - u_4} = \frac{0.5 - 0.2}{0.8 - 0.2} = \frac{1}{2}$$

$$a_3 = \frac{t - u_3}{u_6 - u_3} = \frac{0.5 - 0}{0.6 - 0} = \frac{5}{6}$$

1.6 Spline Representations

And the new control points are:

$$q_5 = \left(1 - \frac{1}{6} d_4\right) + \frac{1}{6} d_5$$

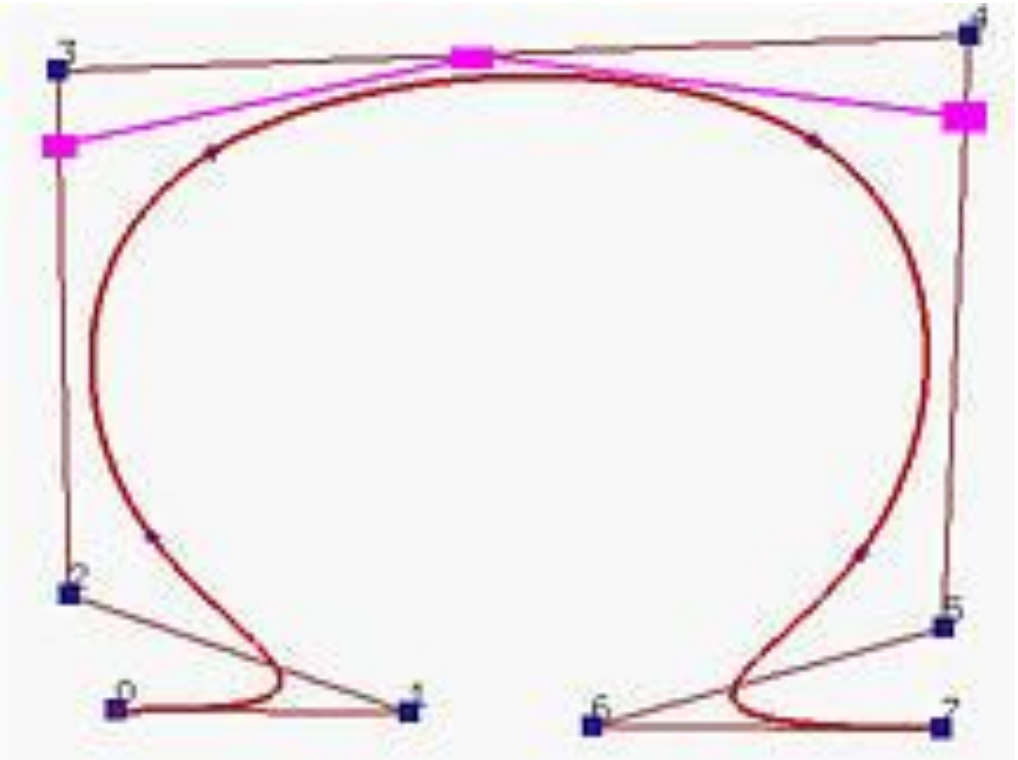
$$q_4 = \left(1 - \frac{1}{2} d_3\right) + \frac{1}{2} d_4$$

$$q_3 = \left(1 - \frac{5}{6} d_2\right) + \frac{5}{6} d_3$$

The new control polygon consist of the points d_0, d_1, d_2, q_3, q_4, q_5, d_5, d_6, and d_7 with knot vector $(0, 0, 0, 0, 0.2, 0.4, 0.5, 0.6, 0.8, 1, 1, 1, 1)$.

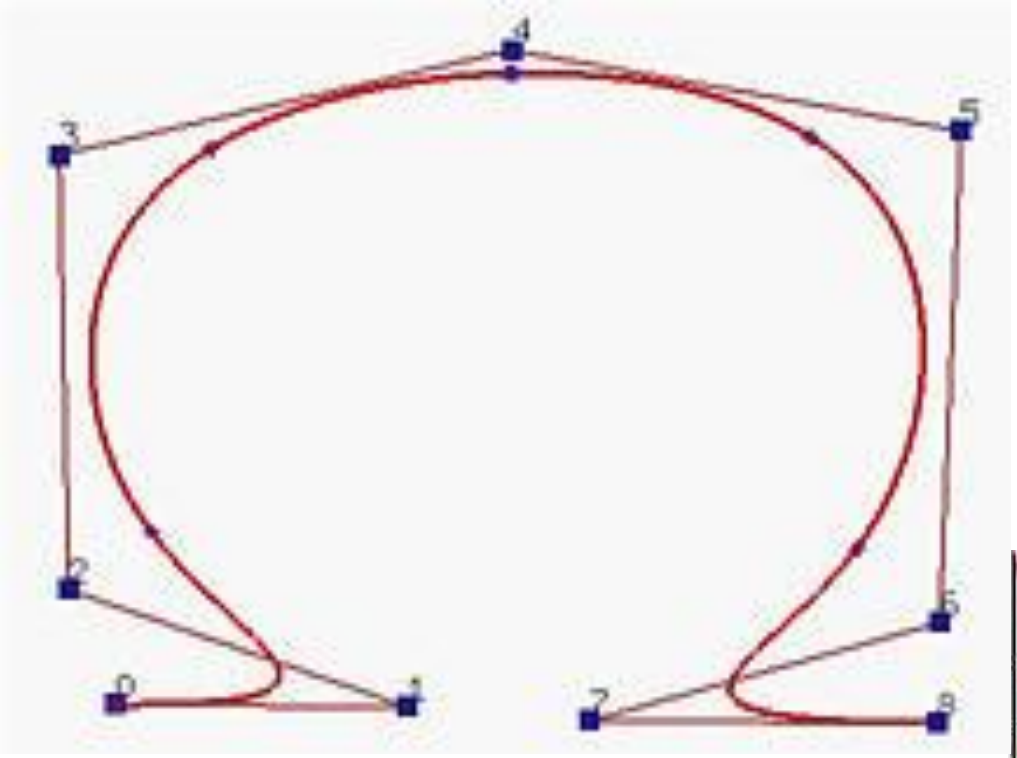
1.6 Spline Representations

For example, the B-spline could look like this before insertion of the new knot:



1.6 Spline Representations

After inserting the new knot:



1.6 Spline Representations

Displaying B-spline curves

Usually, B-splines curves are approximated with line segments for display purposes. Obviously, the number of points that are used for the approximation has great influence on the quality of the resulting visualization of the curve. We could use the control polygon as an approximation; however, this is often times too coarse. Sub-division techniques can lead to better results. Therefore, the control polygon is sub-divided by adding additional points in order to get a better representation. Fortunately, we can exploit some properties of B-splines for sub-dividing the control polygon.

1.6 Spline Representations

Displaying B-spline curves (continued)

First, we convert the B-spline into a Bézier spline. This can be achieved by inserting knots so that every knot has multiplicity m-1. Then, the blending functions of the B-spline resemble the Bernstein polynomials so that the control polygon has the Bézier points as vertices. This already represents a finer approximation since several points were added to the control polygon. If a finer representation is necessary, we can further add points by using the de Casteljau algorithm as we saw earlier.

1.6 Spline Representations

Displaying B-splines with OpenGL

OpenGL already provides a methodology for rendering B-splines:

```
GLUnurbsObj *curveName;
curveName = gluNewNurbsRenderer ();
gluBeginCurve (curveName);
gluNurbsCurve (curveName, nknots, *knotvector,
               stride, *controlPoints, order,
               GL_MAP1_VERTEX_3);
gluEndCurve ();
```

1.6 Spline Representations

Comment:

OpenGL automatically determines the necessary discretization of the B-spline, i.e. the number of line segments needed for the approximation. The approximation depends on the viewing distance. This means the closer the camera moves to the B-spline the more points are used and vice versa. Caution: if you use display lists to generate B-splines OpenGL cannot regenerate the approximation since it is already stored in the display list. Therefore, the approximation is no longer adapted automatically.

1.6 Spline Representations

Demonstration

Behavior of a uniform B-spline curve

http://www.cs.biu.ac.il/~zultia/applets/CAGD/UBS%20draw2/UBsDraw.htm

http://www.ibiblio.org/e-notes/Splines/Basis.htm

1.6 Spline Representations

Rational B-splines

Regular B-splines cannot be used to exactly represent conic sections. This includes circle, ellipse, and parabola. Since these types of curves are common in designs, we would like to be able to model those type of curves exactly, i.e. without any approximation errors. Therefore, we define **rational B-splines** similar to B-splines:

$$X(u) = \frac{\sum_{i=0}^{n} \omega_i d_i N_{i,m}(u)}{\sum_{i=0}^{n} \omega_i N_{i,m}(u)} \qquad u_{m-1} \le u \le u_{n+1}$$

1.6 Spline Representations

Comments:

1) The ω_i are weights that can be used to manipulate the resulting curve. For $\omega_i = 1$ we get a regular B-spline.
2) By increasing the weight ω_i we can pull the curve towards the control point d_i. Decreasing the weight pushes the curve away from d_i.
3) Using a nonuniform knot vector results in the so called **NURBS** curve (**n**on**u**niform **r**ational **B**-**s**pline).

1.6 Spline Representations

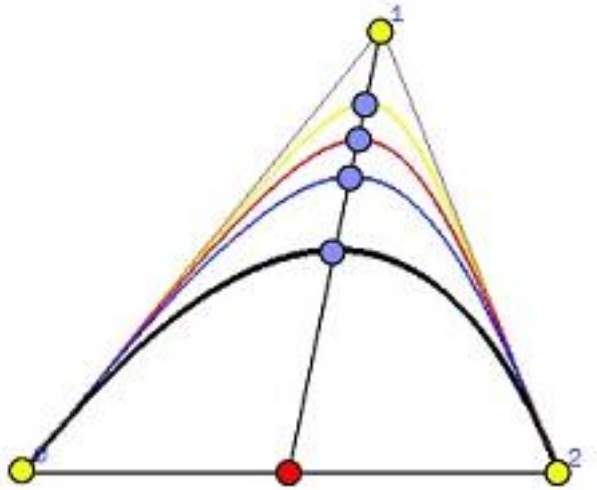
Conic sections

If we define a cubic rational B-spline as

$$X(u) = \frac{d_0 N_{0,3}(u) + \frac{r}{1-r} d_1 N_{1,3}(u) + d_2 N_{2,3}(u)}{N_{0,3}(u) + \frac{r}{1-r} N_{1,3}(u) + N_{2,3}(u)}$$

We can obtain various conic sections with the following values for the parameter r:

- r > ½: hyperbolic section
- r = ½: parabolic section
- r < ½: ellipse section
- r = 0: straight-line segment



1.6 Spline Representations

Displaying NURBS with OpenGL

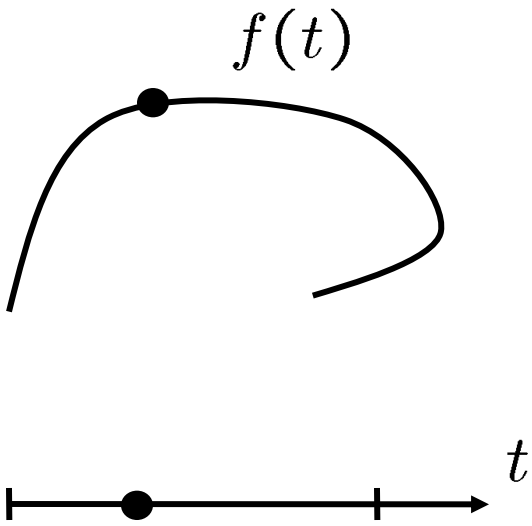
OpenGL already provides a methodology for rendering NURBS:

```
GLUnurbsObj *curveName;
curveName = gluNewNurbsRenderer ();
gluBeginCurve (curveName);
gluNurbsCurve (curveName, nknots, *knotvector,
               stride, *controlPoints, order,
               GL_MAP1_VERTEX_4);
gluEndCurve ();
```

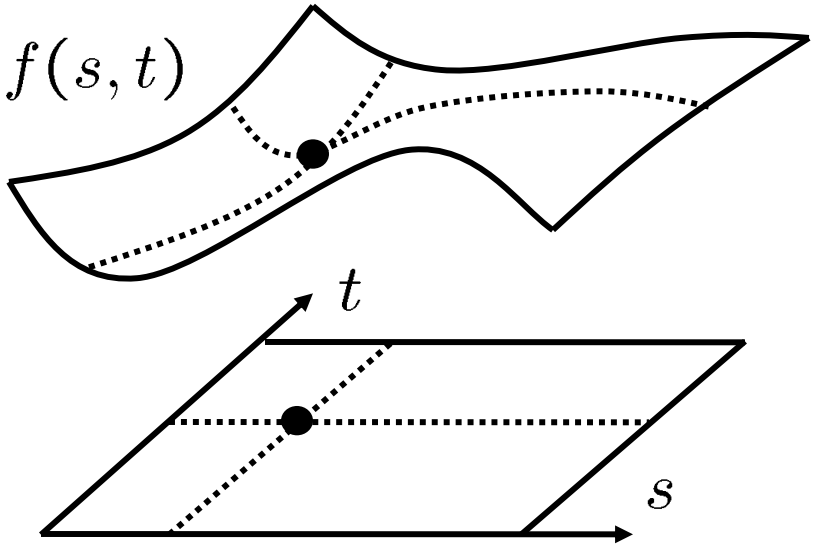
The vertices have the weights as their fourth component.

1.7 Surface Representations

Often surfaces based on the tensor product are used. Tensor product surfaces are parametric surfaces which are defined by two one-dimensional curve representations.



parametric curve



parametric surface

1.7 Surface Representations

From two arbitrary curve representa tions

$$g(s) = \sum_{i=0}^{m} a_i \phi_i(s) \qquad h(t) = \sum_{j=0}^{n} b_j \psi_j(t)$$

with two sets of basis functions $\{\phi_i\}_{i=0}^{m}$ and $\{\psi_j\}_{j=0}^{n}$

we can derive a tensor product surface :

$$f(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} \phi_i(s) \psi_j(t)$$

$$= \sum_{j=0}^{n} c_j(s) \psi_j(t) \quad \text{with} \quad c_j(s) = \sum_{i=0}^{m} c_{ij} \phi_i(s)$$

where c_{ij} are the matrix - shaped coefficien ts of the surface.

1.7 Surface Representations

To construct a 3-D surface we often base the surfaces on several curves. First, we evaluate the curves with respect to the parameter s:

$$c_j = \sum_{i=0}^{m} c_{ij} \phi_i(s)$$

The resulting coefficients $c_j(s)$ are then used to determine a curve on the surface using the second parameter t:

$$f(s,t) = \sum_{j=0}^{n} c_i(s) \psi_i(t)$$

Of course, we can also switch the parameters s and t and evaluate using the parameter t first.

1.7 Surface Representations

Lagrange surfaces

An interpolating surface based on a given set of coefficients c_{ij} can be constructed using the tensor product of the Lagrange polynomials:

$$f(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} L_i^m(s) L_j^n(t)$$

Here, the two set of curves

$$c_j(s) = \sum_{i=0}^{m} c_{ij} L_i^m(s) \quad \text{and} \quad \bar{c}_i(t) = \sum_{j=0}^{n} c_{ij} L_j^n(t)$$

are part of the surface. This results from the interpolating property of the Lagrange basis.

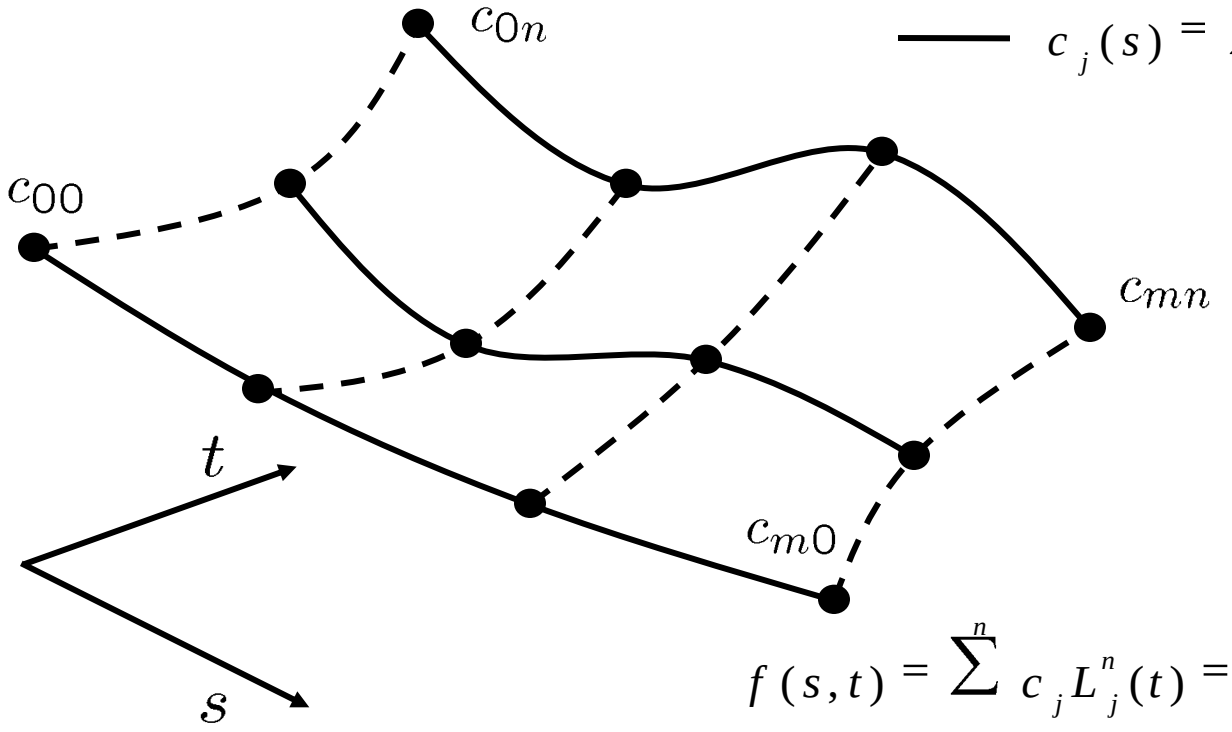
1.7 Surface Representations

Lagrange surface

Example (m=3, n=2):

$$\bar{c}_i(t) = \sum_{j=0}^{n} c_{ij} L^n_j(t)$$

$$c_j(s) = \sum_{i=0}^{m} c_{ij} L^m_i(s)$$



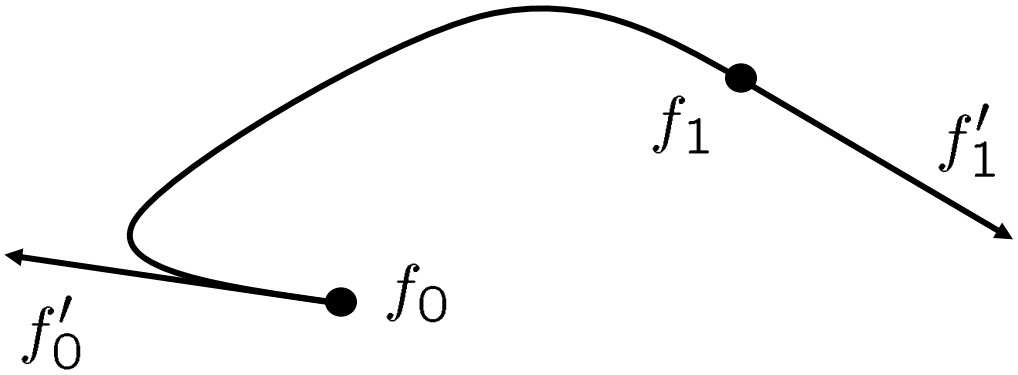
$$f(s,t) = \sum_{j=0}^{n} c_j L^n_j(t) = \sum_{i=0}^{m} \bar{c}_i L^m_i(s)$$

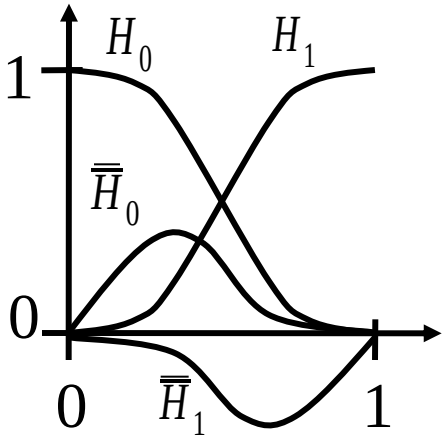
1.7 Surface Representations

Hermite surfaces

Using the cubic Hermite polynomials as basis polynomials, we can interpolate the points and first derivates of the surface at two nodes:

$$f(t) = f_0 H_o(t) + f_0' \overline{H}_0(t) + f_1 H_1(t) + f_1' \overline{H}_1(t)$$

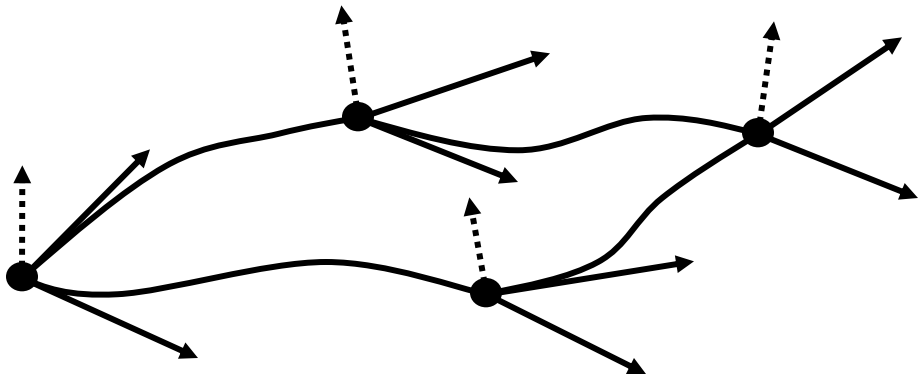




1.7 Surface Representations

Hermite surfaces

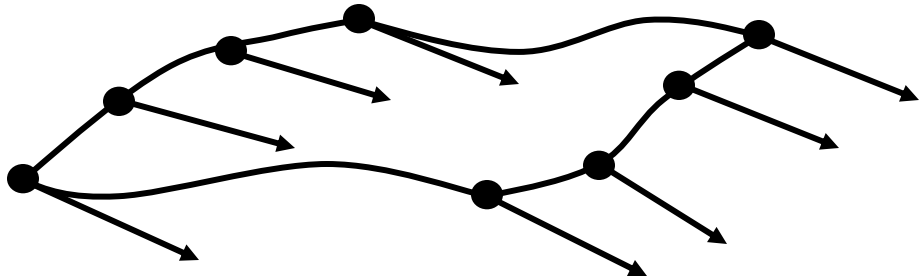
If the surface is defined using the tensor product of cubic Hermite polynomials, the resulting surface then interpolates the interpolation points at the corners, its partial derivatives $\frac{\partial f}{\partial s}, \frac{\partial \mathrm{f}}{\partial t}$ and twist vectors $\frac{\partial^2 f}{\partial s \partial t}$.



1.7 Surface Representations

Hermite-Lagrange tensor product

If only the derivatives in one direction are known, i.e. of one set of curves, a tensor product of Hermite and Lagrange polynomials can be used.



The interpolation of derivatives and maybe also the twist vectors is particularly useful if several patches are supposed to build a single smooth surface.

1.7 Surface Representations

Basis transformation

Similar to polynomial curves, we can transform the polynomial representation from one basis to another. This is especially useful if the data is supposed to be exchanged between different software systems.

Special basis polynomials, such as Bernstein polynomials or B-splines, allow a closer correlation between coefficients (control points) and the surface geometry. Hence, these basis polynomials are more versatile.

1.7 Surface Representations

Bézier tensor product

The tensor product of two Bézier curves results in a Bézier surface representation. Similar to the previous approaches, we define a set of curves X_j defined by the control points b_{ij}:

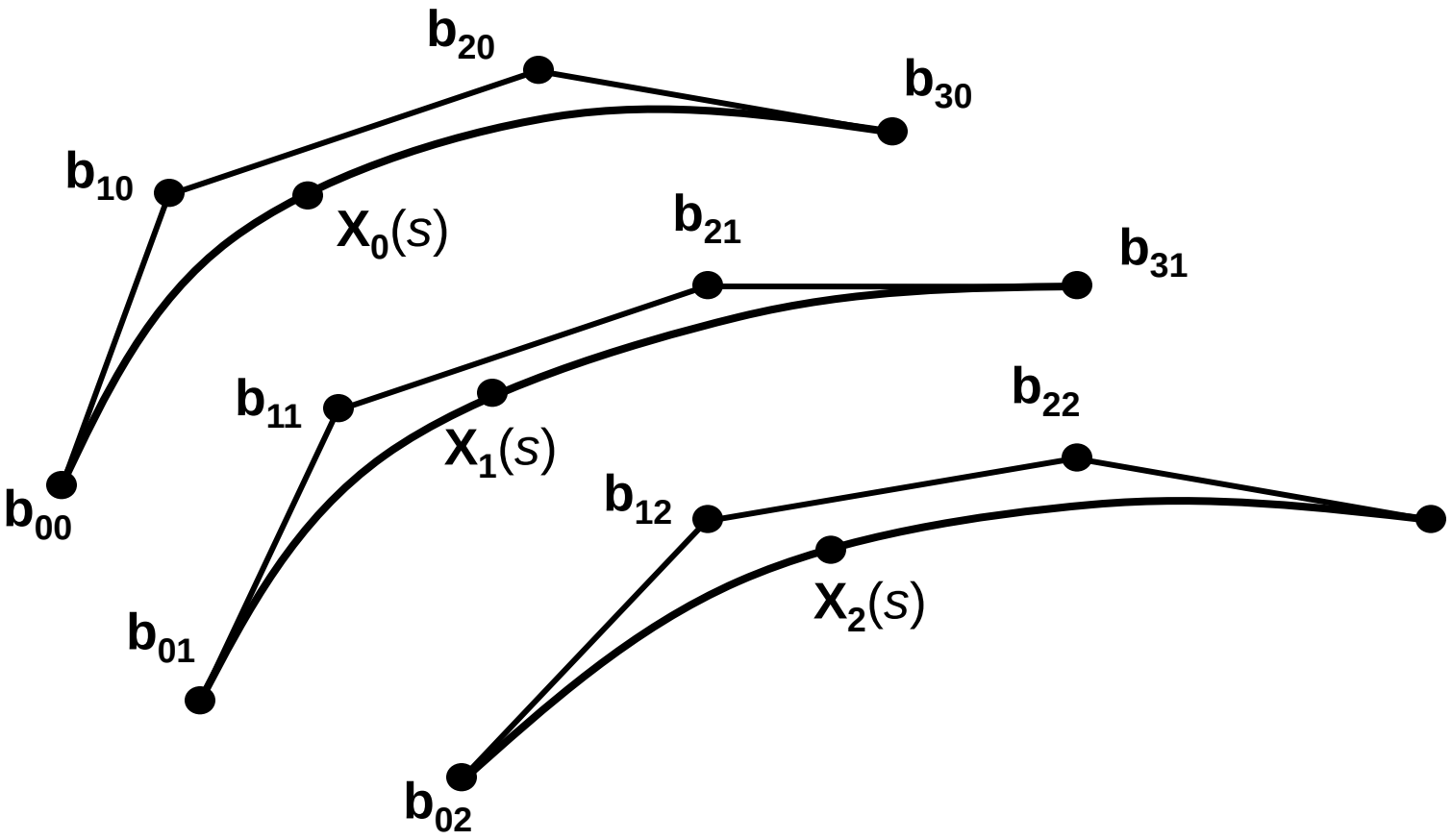
$$X_j(s) = \sum_{i=0}^{m} b_{ij} B_i^m(s) \qquad (j = 0, \ldots, n)$$

This set of curves then gives us control points $b_j = X_j(s)$ which we can use to define a Bézier curve with a second parameter t:

$$X(s,t) = \sum_{j=0}^{n} b_j B_j^n(t) = \sum_{j=0}^{n} \sum_{i=0}^{m} b_{ij} B_i^m(s) B_j^n(t)$$

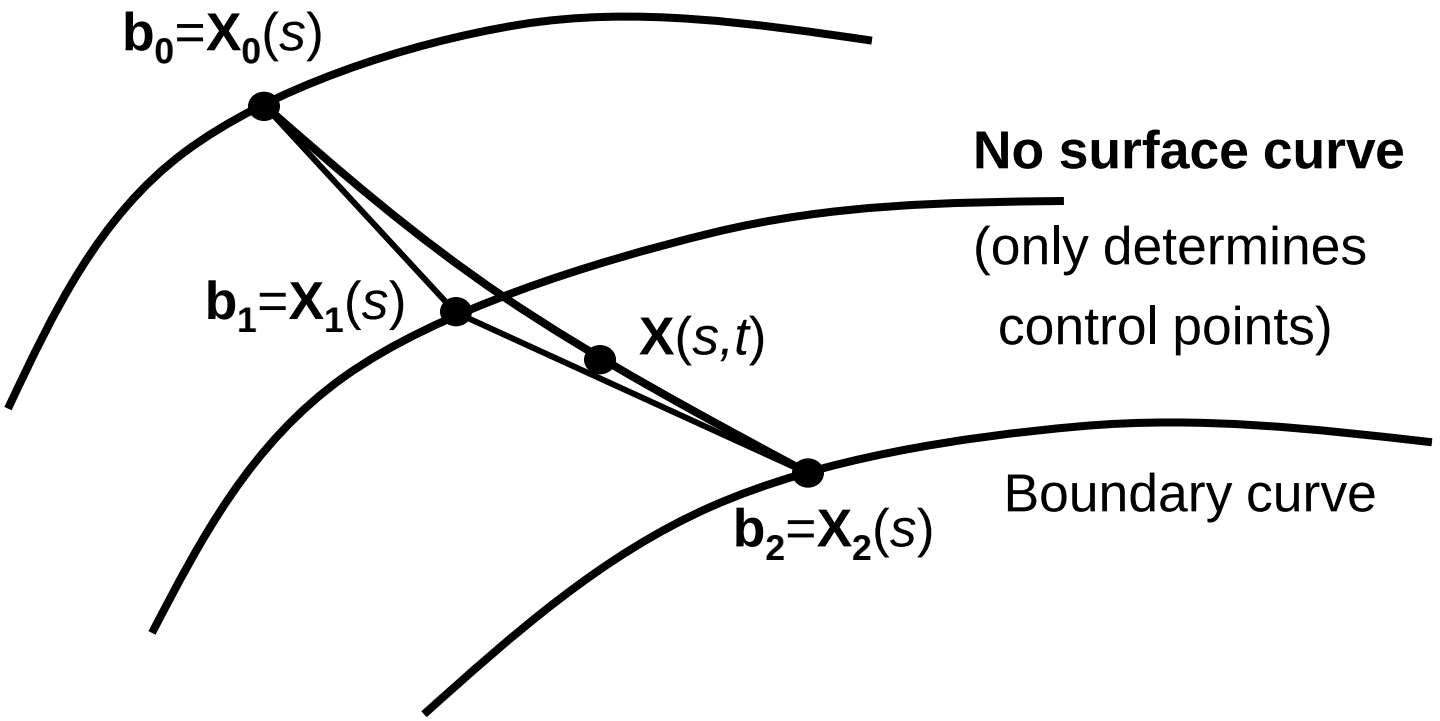
1.7 Surface Representations

Example (m=3, n=2):



1.7 Surface Representations

Example (m=3, n=2):



1.7 Surface Representations

Bézier tensor product

We can also construct a set of curves $X_i(t)$ first, and then compute the control points for the curve parameterized by s. Only the curves at the edge of surface are curves on the surfaces. All other curves of the set are not necessarily surface curves, i.e. part of the surface.

For $m=n$, the resulting surface is called bilinear, biquadratic, or bicubic, etc. The monomial basis of a bilinear surface consists of the polynomials $\{1, s, t, st\}$. For the quadratic case the monomial basis is $\{1, s, t, st, s^2, t^2, st^2, s^2t^2\}$.

1.7 Surface Representations

Bézier tensor product

Increasing the degree of the surface or subdividing the surface (along one direction) can be achieved by applying the respective algorithm to the row or column of the coefficient matrix.

For evaluating the Bézier tensor product efficiently, the de Casteljau algorithm can be applied once for each parameter. It does not make any difference if the algorithm is applied first to parameter t or s.

1.7 Surface Representations

De Casteljau algorithm for surfaces

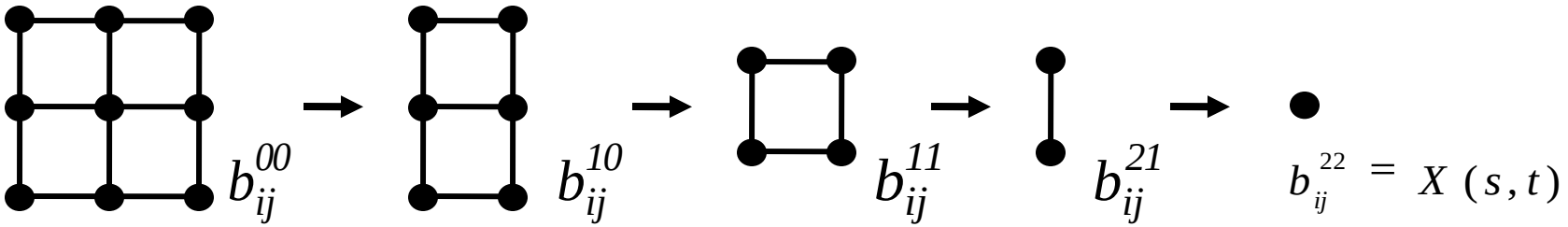
$$b_{ij}^{00} = b_{ij}$$

$$\left.\begin{aligned} b_{ij}^{k+1,l} &= (1-s)\,b_{ij}^{kl} + s\,b_{i+1,j}^{kl} \\ b_{ij}^{k,l+1} &= (1-t)\,b_{ij}^{kl} + t\,b_{i,j+1}^{kl} \end{aligned}\right\}$$

Arbitrary order until we get b_{00}^{mn}

$$X(s,t) = b_{00}^{mn}$$

Example (m=n=2):



1.7 Surface Representations

The **partial derivatives**

$$X_s(s,t) = \frac{\partial X}{\partial s} = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} B'_{i,m}(s) B_{j,n}(t)$$

$$X_t(s,t) = \frac{\partial X}{\partial t} = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} B_{i,m}(s) B'_{j,n}(t)$$

can also be determined using the de Casteljau algorithm :

$$X_s = m\left(b_{10}^{m-1,n} - b_{00}^{m-1,n}\right) \qquad X_t = n\left(b_{01}^{m,n-1} - b_{00}^{m,n-1}\right)$$

For rendering the surface, e.g. using an approximat ing triangulation, we also need the normal vector of the surface : $N = \frac{X_s \times X_t}{\left\| X_s \times X_t \right\|}$.

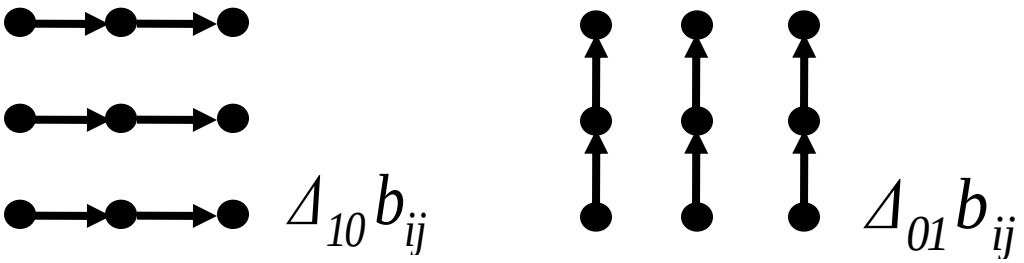
1.7 Surface Representations

Partial derivatives

Another option for computing the partial derivatives is to represent the derivatives as a Bézier surface on its own using the control points

$$m\Delta_{10} b_{ij} := m(b_{i+1,j} - b_{ij}) \quad \text{for} \quad X'_s$$

$$n\Delta_{01} b_{ij} := n(b_{i,j+1} - b_{ij}) \quad \text{for} \quad X'_t$$



1.7 Surface Representations

For a more generic intervall $[a,b]$ $[c,d]$ we get:

$$\frac{\partial^p}{\partial s^p}\frac{\partial^q}{\partial t^q}X(s,t) = A_{pq}\cdot\sum_{i=0}^{m}\sum_{j=0}^{n}\Delta^{pq}b_{ij}B_i^{m-p}\left(\frac{s-a}{b-a}\right)B_j^{n-q}\left(\frac{t-c}{d-c}\right)$$

where

$$A_{pq} = \frac{m!}{(m-p)!}\cdot\frac{1}{(b-a)^p}\cdot\frac{n!}{(n-q)!}\cdot\frac{1}{(d-c)^q}$$

$$\Delta^{pq}b_{ij} = \Delta^{p-1,q}b_{i+1,j} - \Delta^{p-1,q}b_{ij}$$

$$\Delta^{0q}b_{ij} = \Delta^{0,q-1}b_{i,j+1} - \Delta^{0,q-1}b_{ij}$$

$$\Delta^{00}b_{ij} = b_{ij}$$

1.7 Surface Representations

Properties of Bézier surfaces

1) The surface segment lies in the convex hull of the defining net of control points (due to the successive convex-combinations of the de Casteljau algorithm)
2) The Bézier point b_{ij} has the greatest influence on the segment at $(s,t)=(i/m, i/n)$ (pseudo-local control)
3) The corners of the net of control points and the corners of the Bézier segment are identical, i.e. the they are interpolated.
4) The control points on the edge of the net are the Bézier points of the edge of the Bézier segment.

1.7 Surface Representations

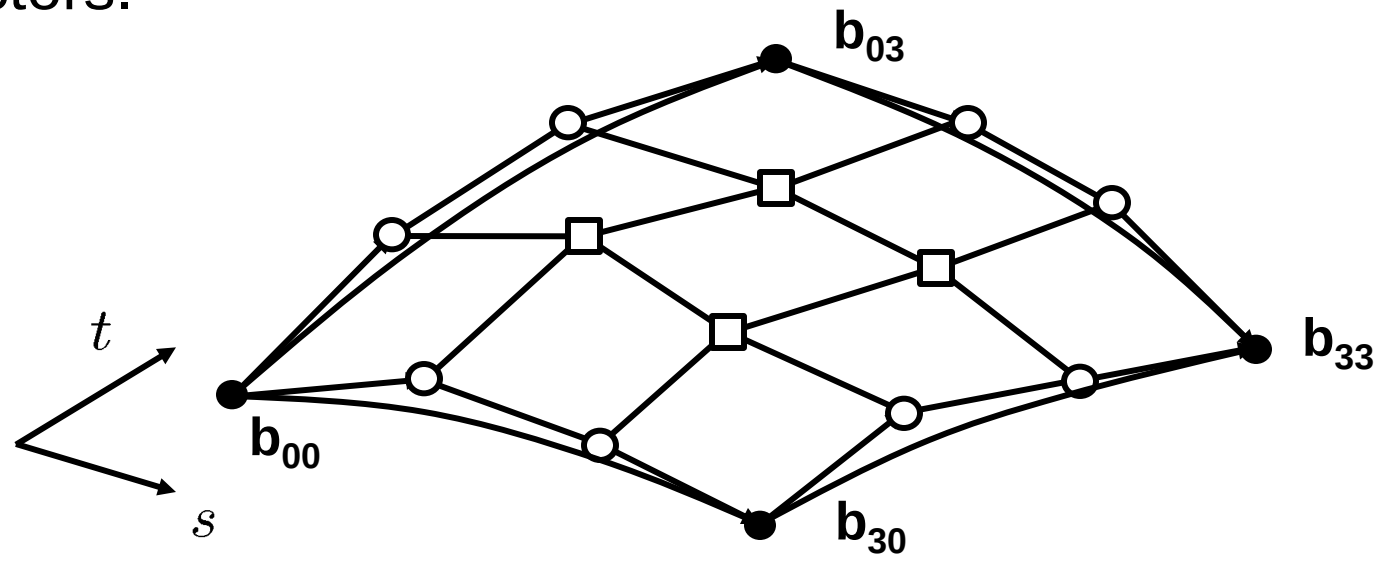
Surface curves of the type $X(s,t_0)$ or $X(s_0,t)$ (s_0, t_0 constant) are called **iso-parameter lines** and are polynomials of degree m or n, respectively. This is not true for diagonal surface curves $X(at+b,ct+d)$ (a,b,c,d constant) which generally are polynomials of degree $m+n$.

Using the de Casteljau algorithm, we can sub-divide an iso-parameter line in sub-segments. Continuing this sub-division process in both parameter direction results in a series of Bézier nets that converges towards the Bézier surface. This fact can be used, for example, for determining the intersection of two surfaces using this series as approximation.

1.7 Surface Representations

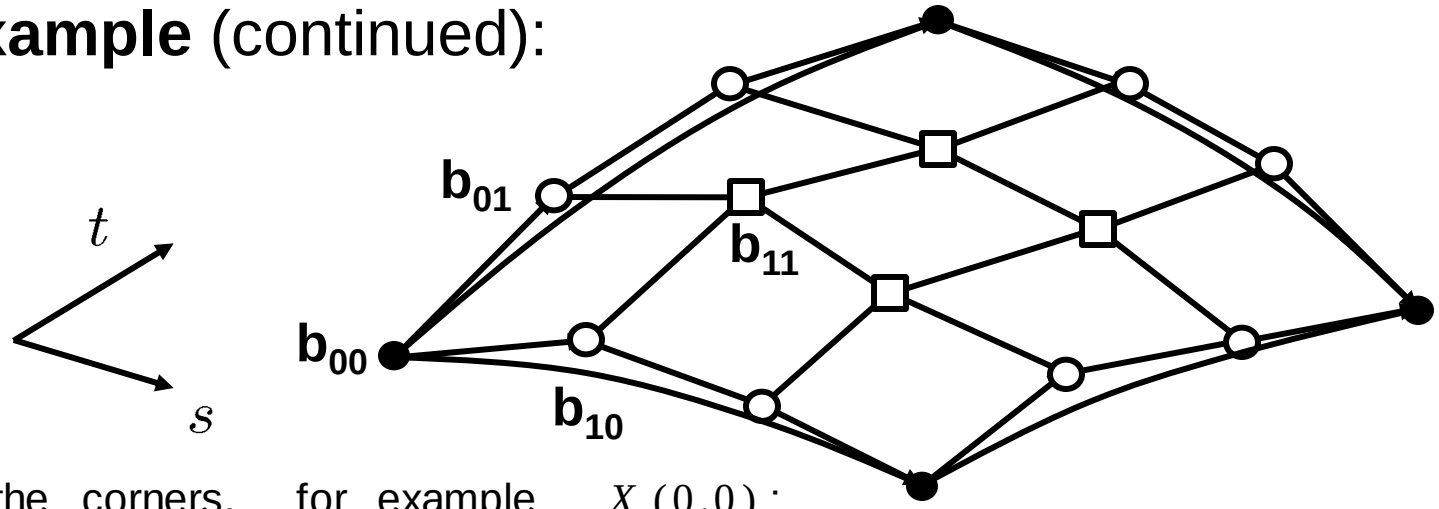
Example: bicubic Bézier surface

Similar to Hermite surface, the inner control points of the boundary curves define the partial derivatives in the corners . The inner control points determine the twist vectors.



1.7 Surface Representations

Example (continued):



In the corners, for example $X(0,0)$:

$$X_s(0,0) = \frac{\partial X}{\partial s} = 3(b_{10} - b_{00})$$

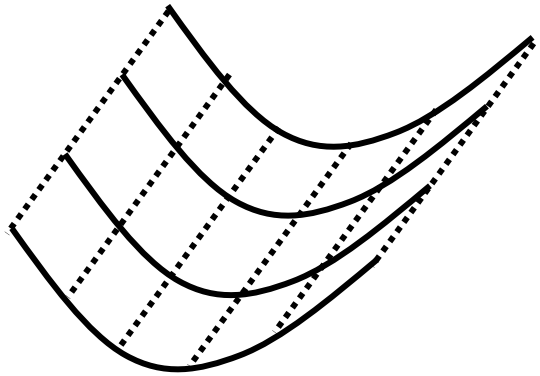
$$X_t(0,0) = \frac{\partial X}{\partial t} = 3(b_{01} - b_{00})$$

$$X_{st}(0,0) = \frac{\partial^2 X}{\partial s \partial t} = 9(b_{11} - b_{10} - b_{01} + b_{00}) \quad \text{(twist vector)}$$

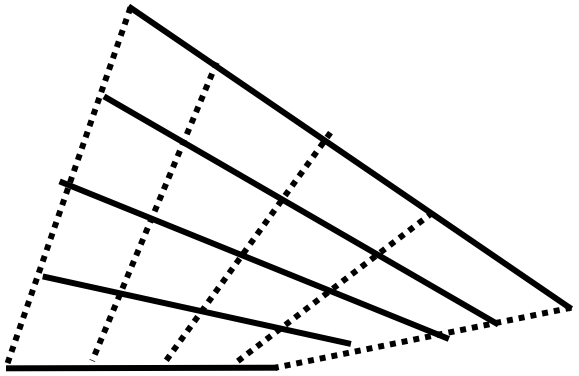
1.7 Surface Representations

The twist vector

The twist vector X_{st} describes the twist of the patch, e.g. how much two parallel straight edges are rotated against each other.



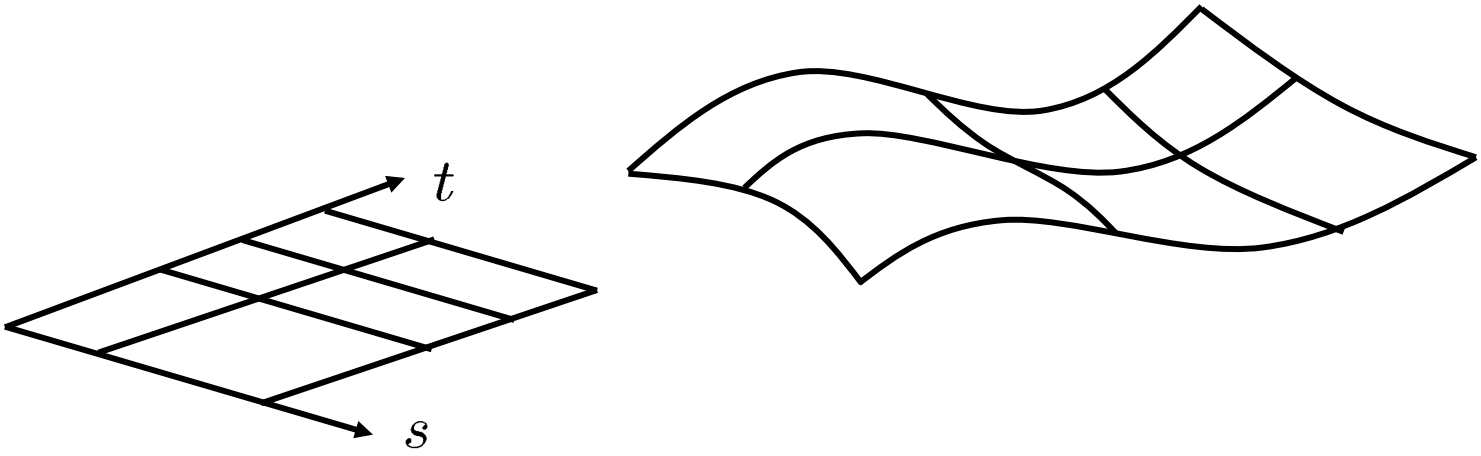
Surface with zero twist



Surface with constant twist

1.7 Surface Representations

Spline surfaces



Spline surfaces are composed of several patches (surface segments). At the boundaries certain conditions for a smooth and continuous transition need to be observed.

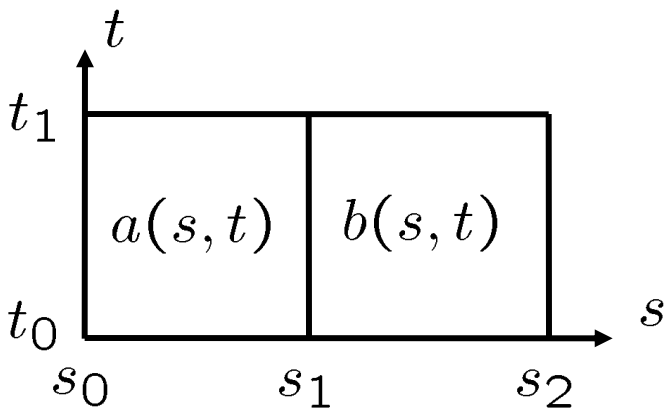
1.7 Surface Representations

Spline surfaces

Let us consider two patches

a with $(s,t) \in [s_0,s_1] \quad [t_0,t_1]$ and

b with $(s,t) \in [s_0,s_1] \quad [t_0,t_1]$.



For C^k continuity, the first k partial derivatives of a and b along the common boundary at s_1 have to be equal:

C^0: $a(s_1, t) = b(s_1, t), t \in [t_0, t_1]$

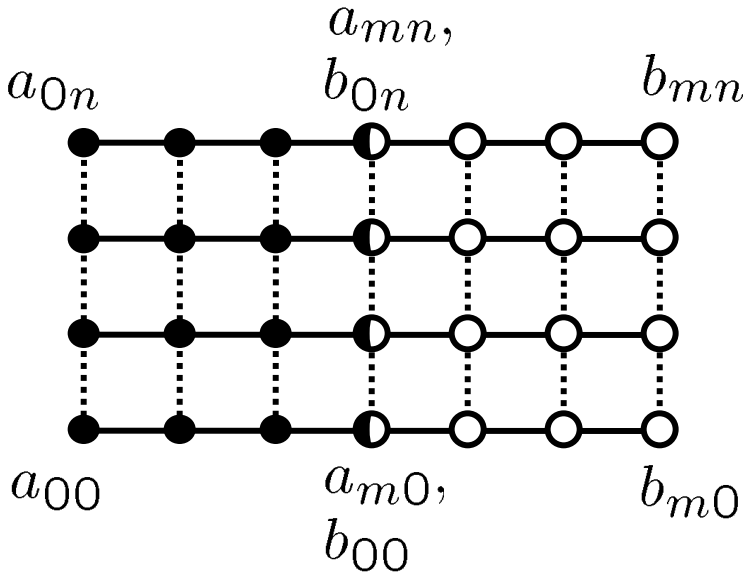
C^1: $a_s(s_1, t) = b_s(s_1, t)$, $t \in [t_0, t_1]$

C^2: $a_{ss}(s_1, t) = b_{ss}(s_1, t)$, $t \in [t_0, t_1]$

1.7 Surface Representations

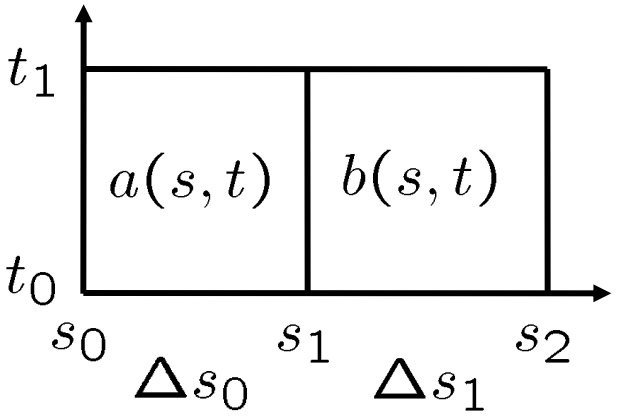
Spline surfaces

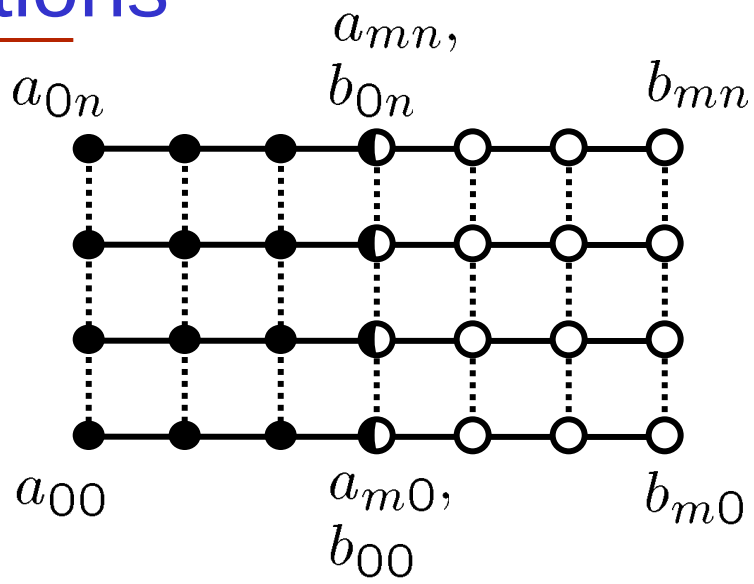
If Bézier surfaces are used as patches a and b then the condition for a C^k continuous transition is identical with the conditions for the individual curves when looking at the individual cells of the Bézier net.



1.7 Surface Representations

Spline surfaces





$$a_{m,i} = b_{0,i} \quad (i = 0, \cdots, n).$$

For a C^0 continuous transition it is required that $a_{m,i}=b_{0,i}$ $(i=0,\ldots,n)$.

For a C^1 continuous transition we get the additional condition: $\frac{1}{\Delta s_0}(a_{m,i} - a_{m-1,i}) = \frac{1}{\Delta s_1}(b_{1,i} - b_{0,i})$

1.7 Surface Representations

Spline surfaces

For C2 continuity the following additional equation needs to be fulfilled:

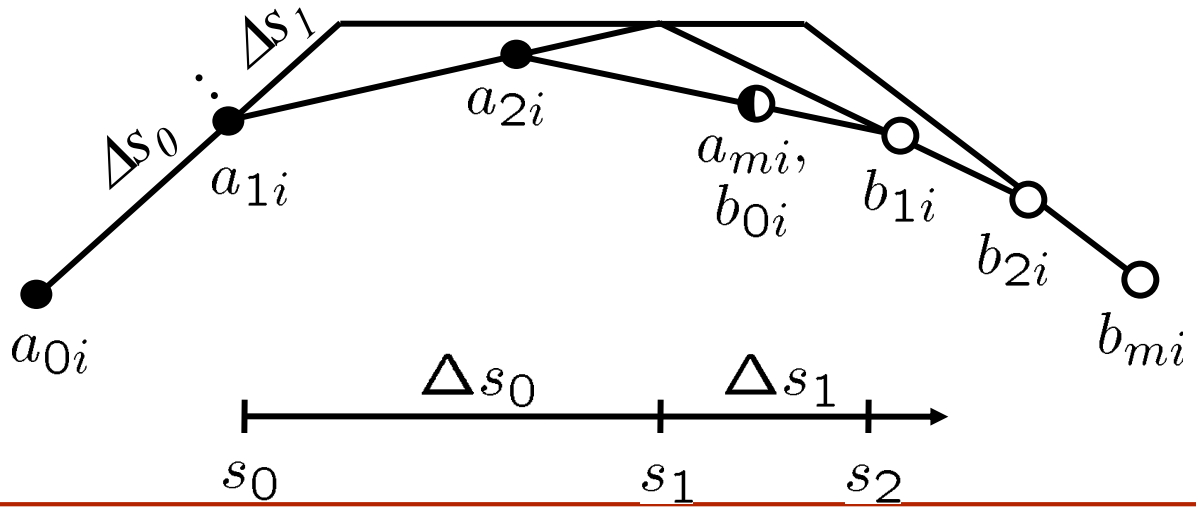
$$\frac{1}{\Delta s_0^2}(a_{m,i} - 2a_{m-1,i} + a_{m-2,i}) = \frac{1}{\Delta s_1^2}(b_{2,i} - 2b_{1,i} + b_{0,i})$$

If one patch, for example a, is given then the neighboring control points of the next patch b can be derived from these equations. The rows of the control net of the patch a can be interpreted as control polygons of a Bézier curve so that the next points of the control net can be determined accordingly.

1.7 Surface Representations

Spline surfaces

The new control points can be found, for example, by evaluating using the de Casteljau algorithm at s_2, i.e. outside of the interval Δs_0. In analogy to sub-dividing a Bézier curve we can extend it. For a C^k continuous transition we need the new points $b_{0,i}, \ldots, b_{k,i}$.

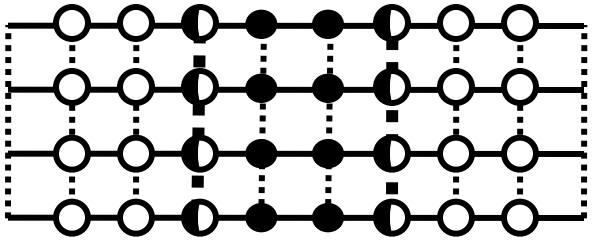


1.7 Surface Representations

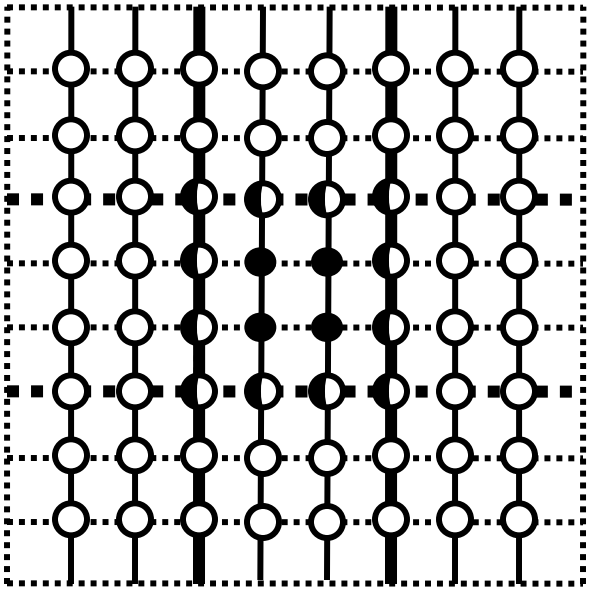
Example:

Determination of the eight neighboring points to achieve C^2 continuity of a bicubic Bézier patch.

Step 1: rows



Step 2: columns



1.7 Surface Representations

Interpolating using Bézier surfaces

Let p_{ij} be a set of given points that are to be interpolated at the parameter values (s_i, t_j). (the parameterization is given by two vectors and not an arbitrary matrix of nodes.)

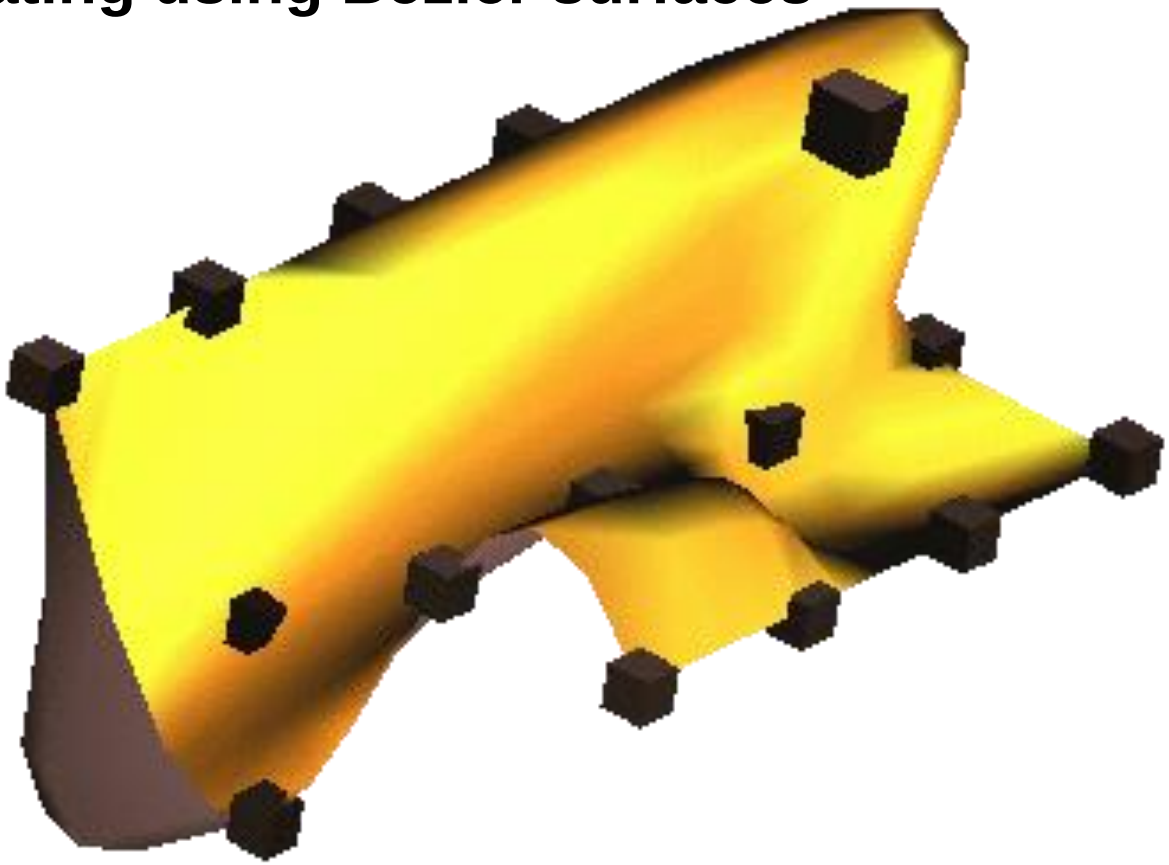
In order to determine an interpolating Bézier surface

$$f(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} B_i^m(s) B_j^n(t), \qquad f(s_i, t_j) = p_{ij}$$

we can use the algorithm for curves.

1.7 Surface Representations

Interpolating using Bézier surfaces



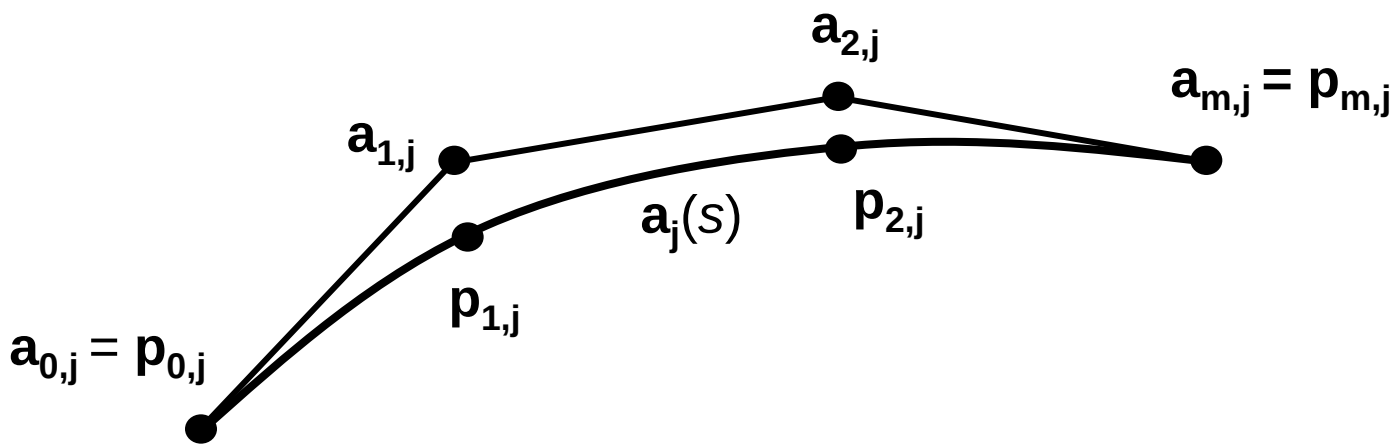
Bicubic interpolating surface

1.7 Surface Representations

Interpolating using Bézier surfaces

First, we generate a set of interpolating curves

$$a_j(s) = \sum_{i=0}^{m} a_{ij} B_i^m(s), \qquad a_j(s_i) = p_{ij} \qquad (j = 0, \ldots, n)$$



The inner Bézier points are determined using a system of linear equations (which uses the same matrix for each j).

1.7 Surface Representations

Interpolating using Bézier surfaces

Then, the points a_{ij} are interpolated row-wise in order to determine the Bézier points b_{ij} of the surface:

$$b_i(t) = \sum_{j=0}^{n} b_{ij} B_j^n(t), \qquad b_i(t_j) = a_{ij} \qquad (j = 0, \ldots, m)$$

The surface described by the Bézier points b_{ij} then interpolates the points p_{ij}.

Proof:

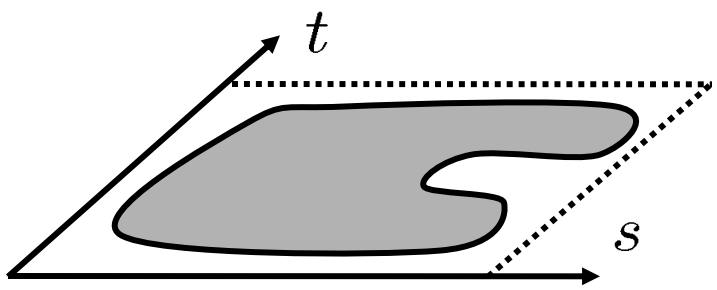
$$f(s_k, t_l) = \sum_{i=0}^{m} \left(\sum_{j=0}^{n} b_{ij} B_j^n(t_l) \right) B_i^m(s)$$

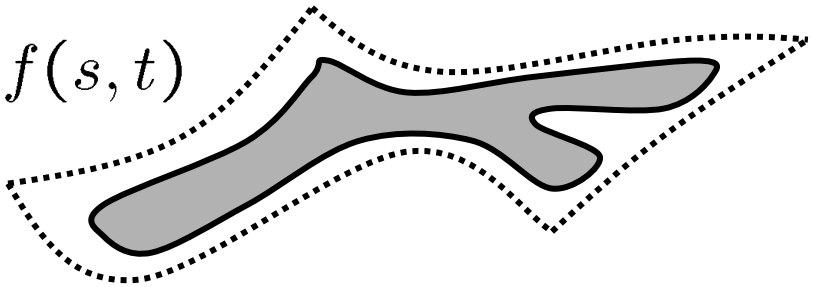
$$= \sum_{i=0}^{m} a_{il} B_i^m(s_k) = a_l(s_k) = p_{kl}$$

1.7 Surface Representations

Trimming

In some cases, it is necessary to bound the area on which a free-form surface is defined, e.g. by choosing a boundary curve (**trimming curve**) in order to limit (trim) the surface:





Trimmed free-form surface

1.6 Spline Representations

Displaying B-spline surfaces with OpenGL

OpenGL already provides a methodology for rendering B-spline surfaces:

```
GLUnurbsObj *surfName;
surfName = gluNewNurbsRenderer ();
gluBeginSurface (surfName);
gluNurbsSurface (surfName, nuknots,
                 *uknotvector, nvknots,
                 *vknotvector, ustride, vstride,
                 controlPoints, udegree,
                 vdegree, GL_MAP2_VERTEX_3);
gluEndSurface (surfName);
```

1.6 Spline Representations

Trimming a B-spline surface with OpenGL

OpenGL already provides a methodology for trimming B-spline surfaces:

```
GLUnurbsObj *surfName;
gluBeginTrim (surfName);
gluPw1Curve (surfName, npts, *curvepts, stride,
             GL_MAP1_TRIM_2);
gluEndTrim (surfName);
```

1.7 Surface Representations

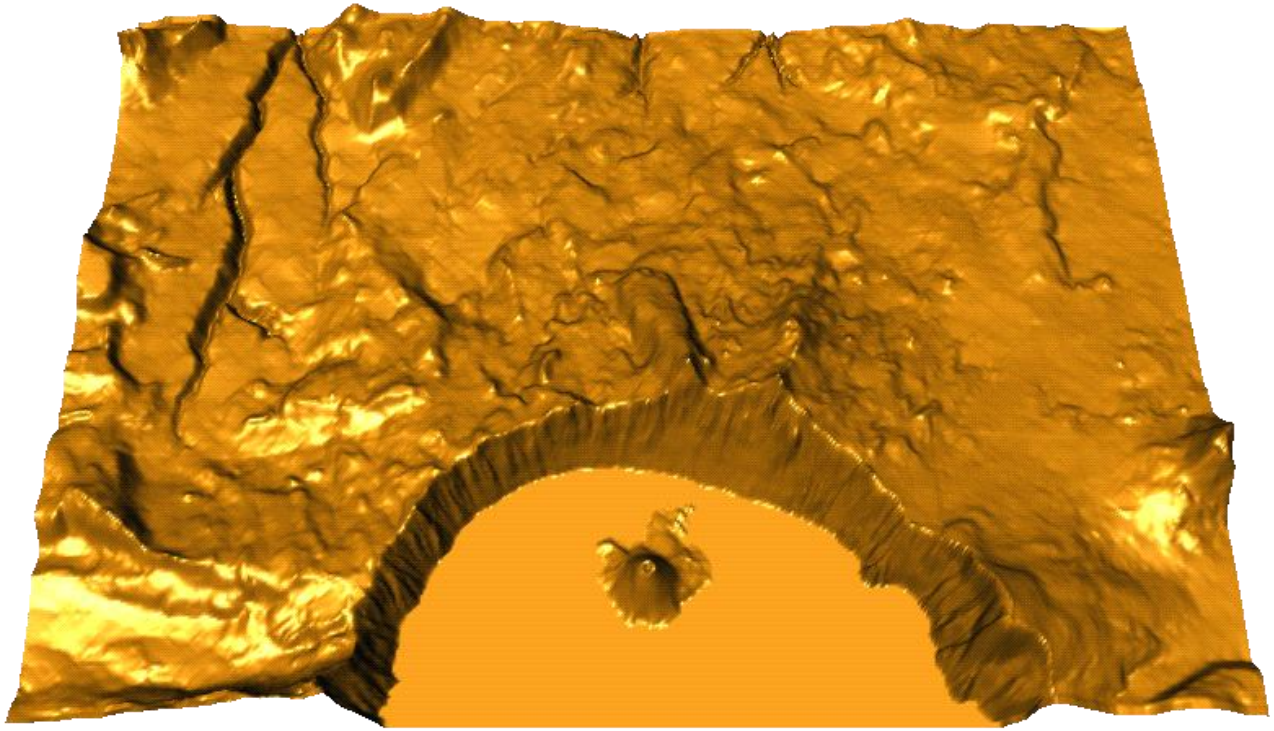
Scalar fields

Instead of using three-dimensional control points for defining a free-form surface, we can use scalar valued "points" (**ordinates**). The resulting one-dimensional surface function can then be visualized as a graph on top of the defining area. Surfaces that are defined in such a way are called **scalar fields** (in an analog way, 3-D functions can be interpreted as **vector fields**).

One-dimensional spline surfaces are often used for representing geographical height fields (terrain models):

1.7 Surface Representations

Scalar fields

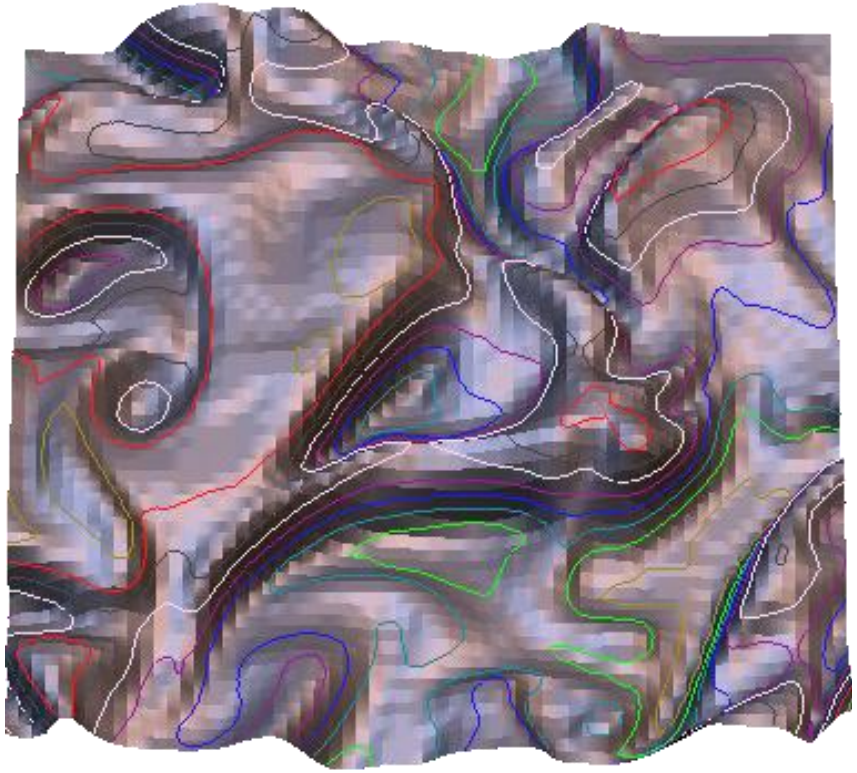


Crater lake terrain model. Source: U.S. Geological Survey

1.7 Surface Representations

Iso-curves

The height curves of a scalar field f, i.e. $f(x,y)$ constant, are called **iso-curves**. Iso-curves are algebraic or implicit curves, respectively.



Iso-lines of a bilinear spline surface, Lawrence Livermore National Laboratory

1.7 Surface Representations

Iso-surfaces

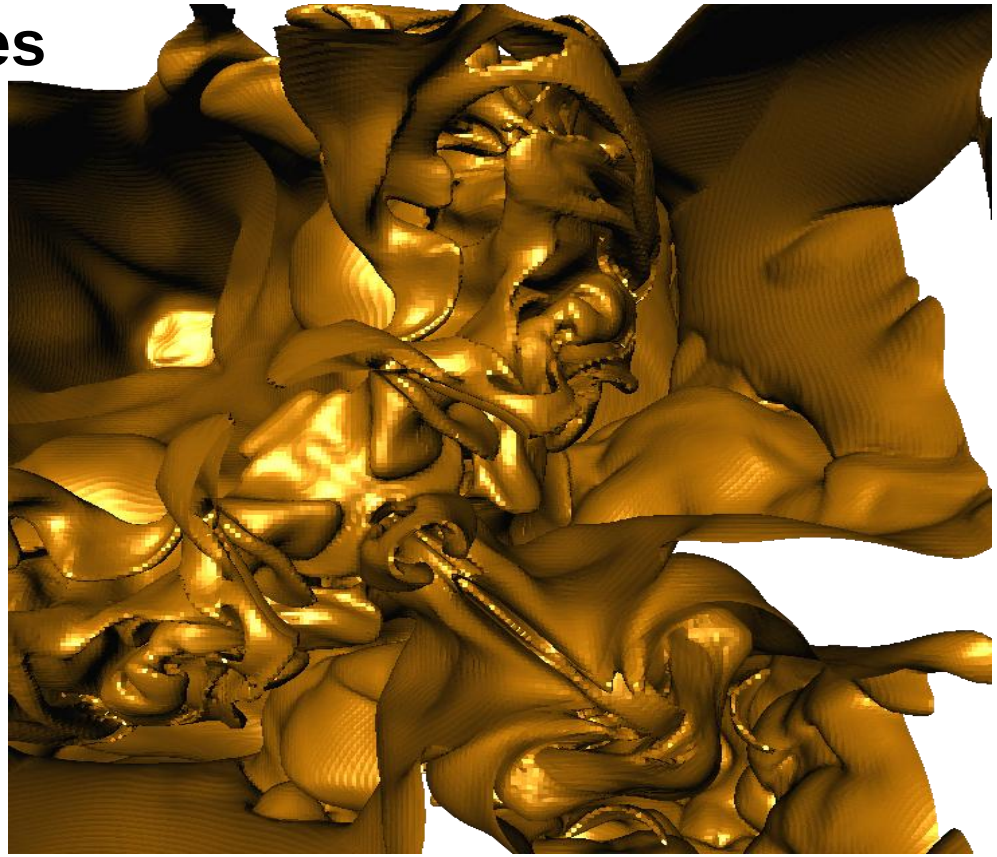
In analogy to iso-cruves we can define iso-surfaces by using three-dimensional scalar fields. For an iso-value c, the corresponding iso-surface is defined as the set of all points with $f(x,y,z)=c$.

Tri-variate functions (i.e. functions with three arguments, "**volumes**") can be defined as tensor product. Often times, tri-linear volumes are used:

$$f(s,t,u) = \sum_{i=0}^{l} \sum_{j=0}^{m} \sum_{k=0}^{n} d_{ijk} N_i^2(s) N_j^2(t) N_k^2(u)$$

1.7 Surface Representations

Iso-surfaces

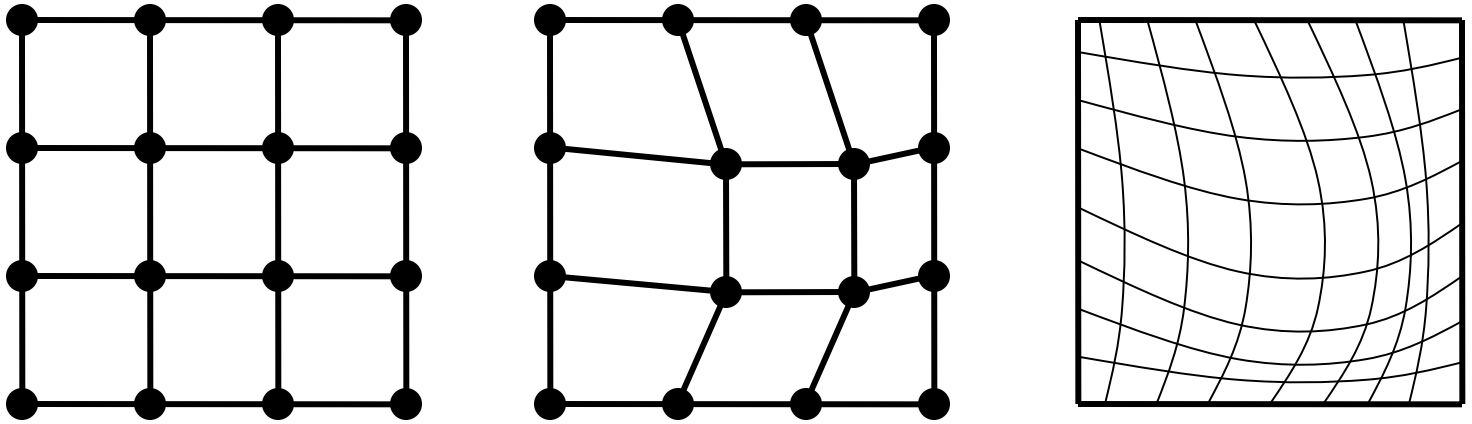


Iso-surface of a three-dimensional density function,
Lawrence Livermore National Laboratory

1.7 Surface Representations

Deformation

Deformation, e.g. of a 2-D image, can be by displacing control points of a regular grid structure. The surface defined by the displaced control net then maps the domain onto itself: $(x',y')=f(x,y)$.



1.7 Surface Representations

Morphing

Deformations are used, for example, to cross-fade from one image to another one (image morphing).

Let $m_1(x,y)$ and $m_2(x,y)$ be two images. We then look for a function

$$m(t,x,y),\ t \in [0,1]$$

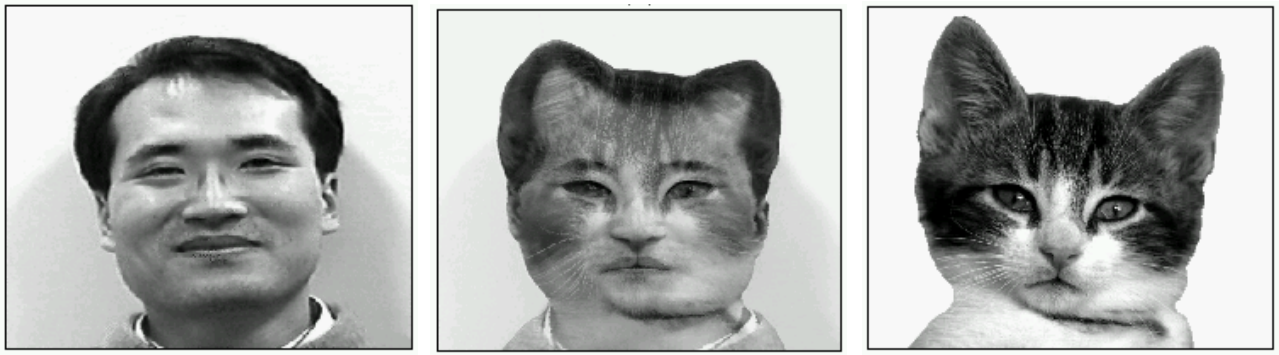
that fades continuously from m_1 to m_2 in such a way that $m(0,x,y)=m_1(x,y)$ and $m(1,x,y)=m_2(x,y)$.

The simple approach, $m=(1-t)\cdot m_1 + t\cdot m_2$, would not consider the contours of the image properly. Therefore, it is necessary to work with deformations.

1.7 Surface Representations

Morphing

First, a deformation $f_2(x,y)$ is defined manually which maps previously picked points within m_1 onto m_2. While moving the geometry, i.e. $f(t,x,y)=(1-t)\cdot(x,y)+t\cdot f_2(x,y)$, the color values of the images are “blended” accordingly.



[S. Lee, et al., Image morphing using deformation techniques, 1996]

1.7 Surface Representations

Morphing

In the same way, three-dimensional objects can be cross-faded, e.g. using B-spline volumes, which are called **free-form deformations** (**FFDs**).

3D Morphing, Stanford University

1.8 Space Partitioning

For representing an object using space subdivision techniques the object space is split up into several smaller elements. For each element, we store if this specific element is covered by the object.

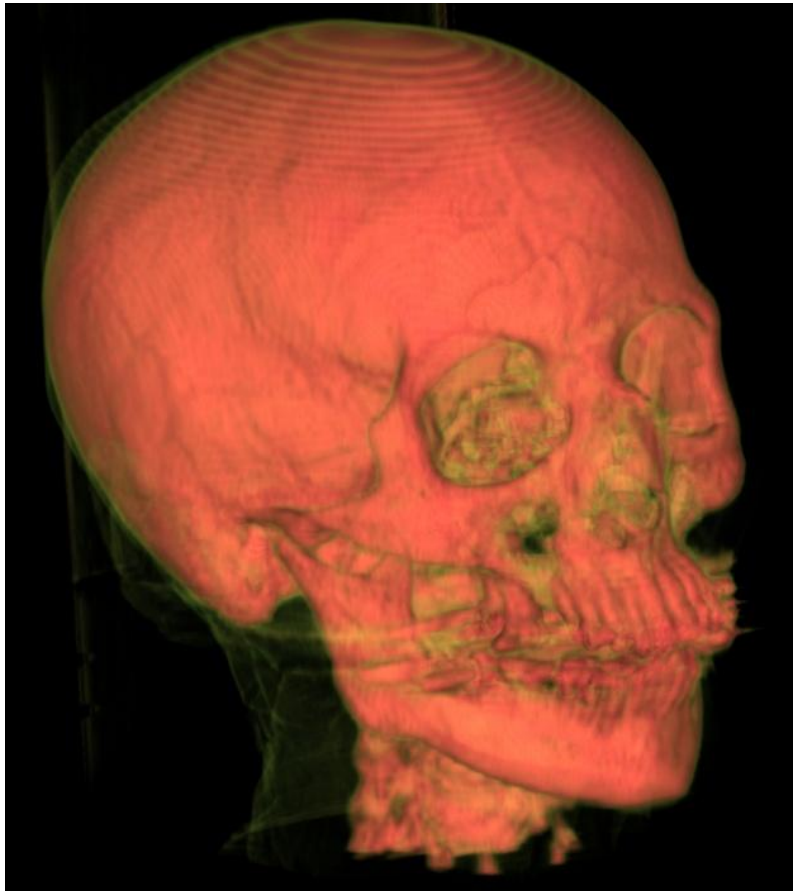
Standard approach:

- Space is divided by a regular equidistant grid, resulting in a grid where each cell has exact identical geometry.
- In 3-D space, we get cube-shaped cells, which are called **voxels** (volume element).

  → The name is in analogy to pixel (picture element) in 2-D

1.8 Space Partitioning

Example: volumetric image of a CT-scanned object



1.8 Space Partitioning

Advantages:

- It can be determined very easily if a given point is part of the object or not.
- It can be checked easily if two objects are connected or attached to each other.
- The representation of an object is unique.

Disadvantages:

- There cannot be any cells that are only partly filled.
- Objects can generally be represented approximately.
- For a resolution of n voxels in each dimension we need n^3 voxels to represent the object. Therefore, it requires a lot of memory → save space using octrees.

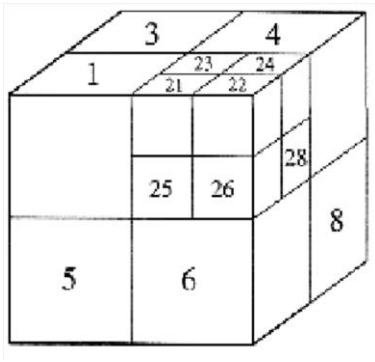
1.8 Space Partitioning

Octrees

An octree is a hierarchical data structure for storing an irregular, i.e. not equidistant, sub-division of a 3-D space.

Idea:

- The initial element is a cube which covers the entire object space. The element can have two states: covered or uncovered.
- In case an element is partly covered, it is sub-divided into eight equally sized sub-elements.
- The coverage of each element is checked recursively until a desired resolution is achieved.



1.8 Space Partitioning

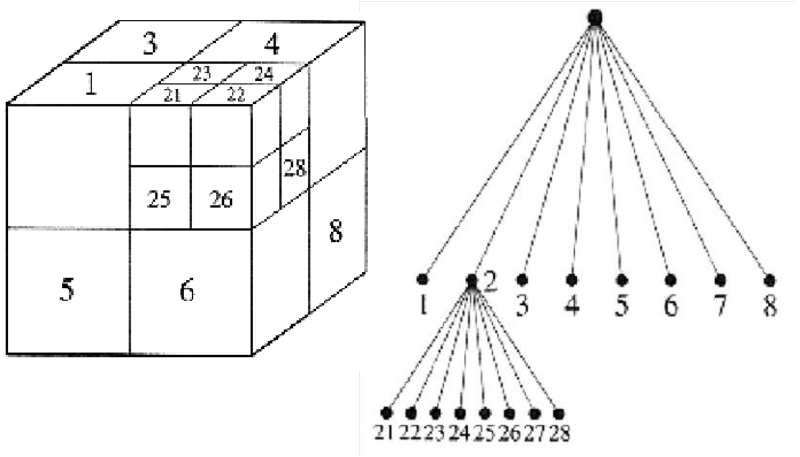
Octrees (continued)

In an octree, each node (element) that is not a leaf has eight successors (sub-elements).

The root of the tree represents the initial cube. For each sub-division a fixed numbering scheme is used for the sub-elements when inserting a new node as a child.

Each leaf stores the state of its corresponding (sub-) cube.

Each inner node represents a partly covered cube.



1.8 Space Partitioning

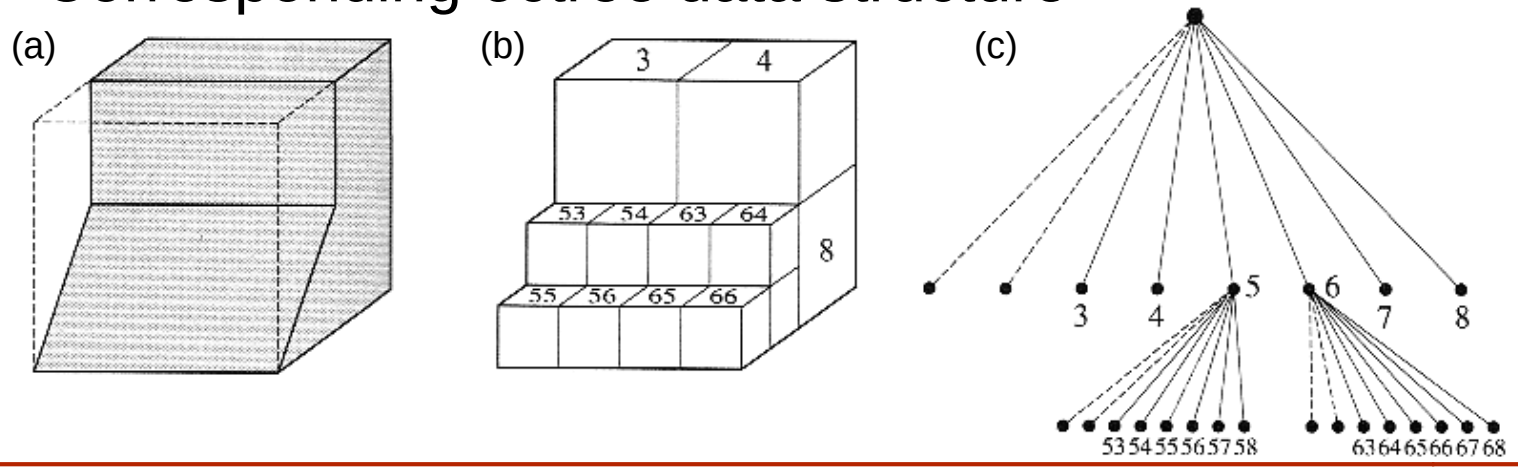
Octrees (continued)

Example: representation of a 3-D object using an octree

a) Object embedded into initial cube.

b) Representation of the object using a maximal sub-division of two.

c) Corresponding octree data structure



1.8 Space Partitioning

Octrees (continued)

Octrees can not only be used for representing 3-D objects. A very common use of octrees is the sub-division of a 3-D scene.

- Here, the individual objects are represented by standard data structures, e.g. polygons.
- The state of the cells of the octree is then extended to a data structure that stores a list of objects, e.g. polygons, which are contained by the cell.

This results in a significant performance increase for algorithms that work on the individual areas of the object space locally (e.g. ray tracing).

1.8 Space Partitioning

Quadtrees

The principle of sub-dividing the 3-D space can be generalized to an n-dimensional space.

For the case $n=2$ we get the sub-division of a 2-D plane resulting in a quadtree, where each inner node of the tree has exactly four children.

Historically, quadtrees are the older data structures. They were used initially in the late 60's of the last century. Octrees were derived from quadtrees and used since the late 70's, early 80's of the last century.

1.8 Space Partitioning

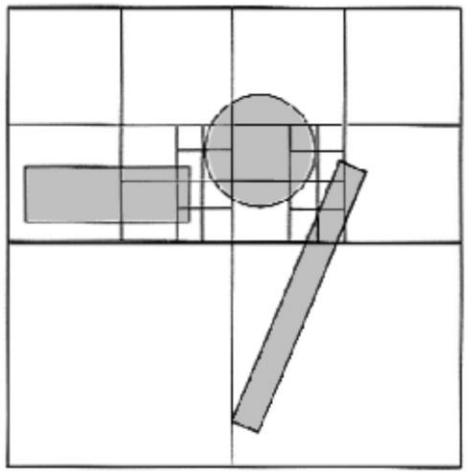
Quadtrees (continued)

Example: sub-division of a 2-D space using a quadtree.

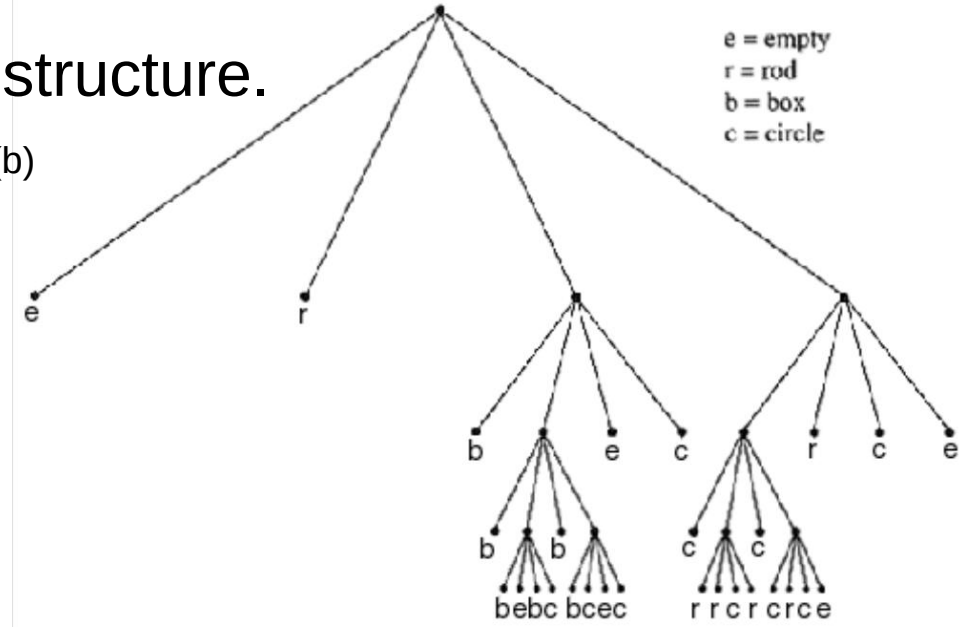
a) Sub-division of the 2-D space until each cell containes maximally one object.

b) Corresponding data structure.

(a)



(b)



1.8 Space Partitioning

Binary space-partitioning (BSP trees)

Octrees and quadtrees both sub-divide at each level equally in each dimension, i.e. at the center.

A BSP tree offers an alternative representation where an element can be sub-divided into **two** sub-elements at an arbitrary (hyper-)plane

- If one sub-element is defined as part of the inside while the other sub-element is defined as the outside, a convex polyhedron can be represented by using properly chosen planes limiting the volume.
- By uniting convex interior areas, arbitrary concave polyhedra with holes can be defined.

1.8 Space Partitioning

BSP trees (continued)

In the realm of computer graphics, BSP trees are often used for determining the visibility of an object.

Idea:

- BSP trees can – similar to octrees and quadtrees – be used for sub-dividing a 3-D scene (see next example). Here, the objects are not bound to a particular rasterization.
- The object space is to be sub-divided recursively in such a way, that each area contains at most one object.
- Using the locations of those areas relatively to the view point, the objects can be sorted according to the viewing distance (depth) easily, i.e. it can be determined which objects are completely invisible.

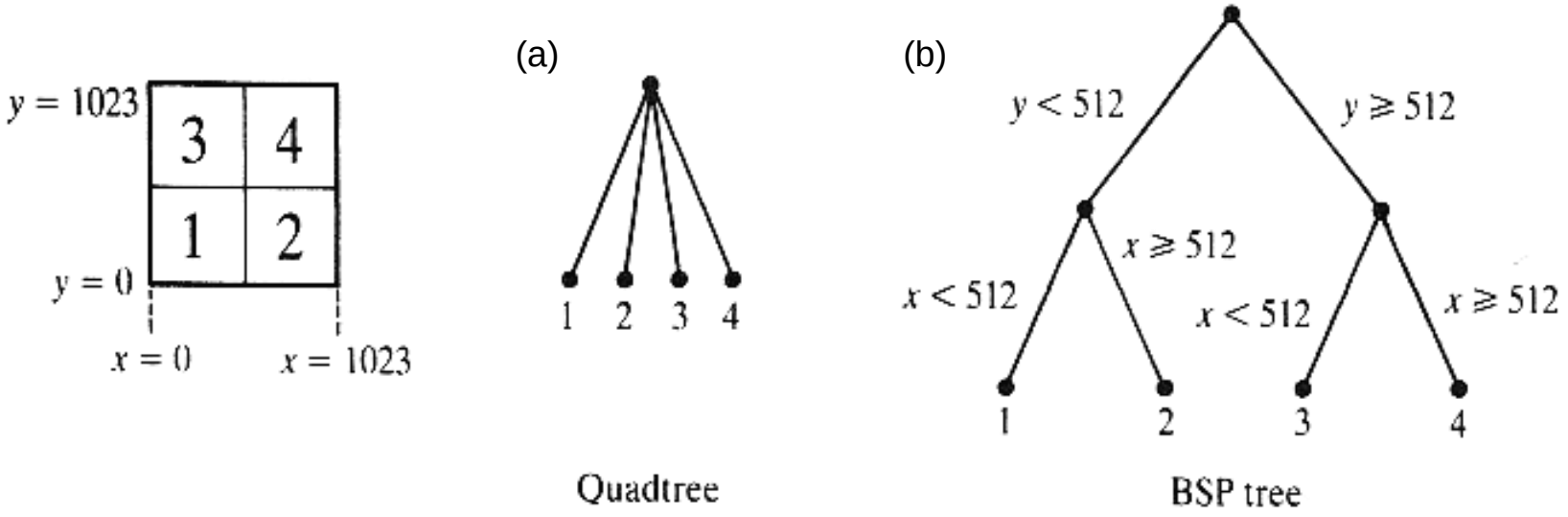
1.8 Space Partitioning

BSP trees (continued)

Example: sub-division of a 2-D scene

a) Using a quadtree

b) Using a BSP tree



1.8 Space Partitioning

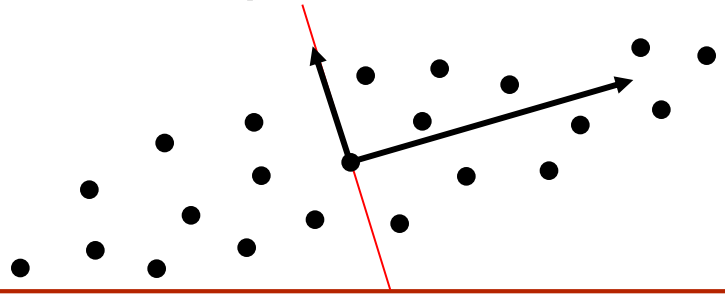
Principal Component Analysis (PCA)

An ideal choice of dividing planes for a BSP tree is offered by the **principal component analysis** (PCA). Let us assume that a complex scene is given by a point cloud

$$P_i \in IR^3 \quad (i=1,\ldots,n)$$

(for example object centers or vertices of polygons).

PCA defines an orthogonal coordinate system e_1, e_2, e_3 which orientation corresponds to the one of the point cloud.



1.8 Space Partitioning

Principal Component Analysis (continued)

Now, we choose the average of all points as the center of the coordinate system:

$$c = \frac{1}{n}\sum_{i=1}^{n} P_i$$

$$A = \begin{pmatrix} P_{1x} - c_x & P_{1y} - c_y & P_{1z} - c_z \\ P_{2x} - c_x & P_{2y} - c_y & P_{2z} - c_z \\ \vdots & \vdots & \vdots \\ P_{nx} - c_x & P_{ny} - c_y & P_{nz} - c_z \end{pmatrix}$$

$$B = \frac{1}{n-1} A^T A$$

$$b_{ij} = \frac{1}{n-1}\sum_{k=1}^{n} a_{ki} a_{kj}$$

B has the real eigenvalues λ_1, λ_2, λ_3 and eigenvectors e_1, e_2, e_3, i.e. $\lambda_i \cdot e_i = B \cdot e_i$. The eigenvectors in combination with the center c form the coordinate system we are looking for. The extent of the point cloud in direction of e_i is proportional to $\sqrt{\lambda_i}$.