The Bezier Function

1 Polynom
2 de Casteljau
3 The formula
4 Segments
5 OpenGL

1 Polynom

We will find an expression for a polynom og degree 3.

A polynom of degree 3

We will use a situation quite similar to the one we had when we developed the Hermit curve.

We want to find the coefficients in an expression of third degree:

$B(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$

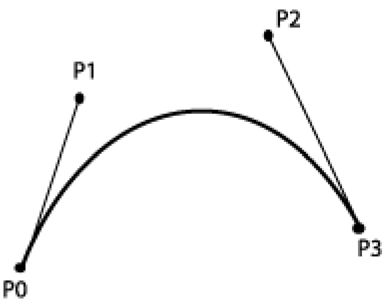
We want geometrical guidelines that state that the curve should start and stop in two points: P0 and P3.

Consequently:

$$B(0) = P_1$$
$$B(1) = P_3$$

We want a polynomial of third degree and need two more guidelines.

We use two points that indirect state the derivative in the endpoints:



We decide that:

$$B'(0) = 3 \cdot (P_1 - P_0)$$
$$B'(1) = 3 \cdot (P_3 - P_2)$$

The derivative is: $B'(t) = 3 \cdot a \cdot t^2 + 2 \cdot b \cdot t + c$

We have to find a,b,c and d in this set of equations:

$$\begin{aligned} B(0) = d &= P_0 \\ B(1) = a + b + c + d &= P_3 \\ B'(0) = c &= 3 \cdot (P_1 - P_0) \\ B'(1) = 3 \cdot a + 2 \cdot b + c &= 3 \cdot (P_3 - P_2) \end{aligned}$$

We solve and find:

$$\begin{aligned} a &= -P_0 + 3 \cdot P_1 - 2 \cdot P_2 + P_3 \\ b &= 3 \cdot P_0 - 6 \cdot P_1 + 3 \cdot P_2 \\ c &= 3 \cdot P_1 - 3 \cdot P_0 \\ d &= P_0 \end{aligned}$$

and we set up the expression for B(t):

$$B(t) = -P_0 t^3 + 3 \cdot P_1 t^3 - 3 \cdot P_2 t^3 + P_3 t^3 + 3 \cdot P_0 t^2 - 6 \cdot P_1 t^2 + 3 \cdot P_2 t^2 + 3 \cdot P_1 t - 3 \cdot P_0 t + P_0$$

We rearrange and we get:

$$B(t) = -P_0(-t^3 + 3 \cdot t^2 - 3 \cdot t + 1) + P_1(3 \cdot t^3 - 6 \cdot t^2 + 3 \cdot t) + P_2(-3 \cdot t^3 + 3 \cdot t^2) + P_3(t^3)$$

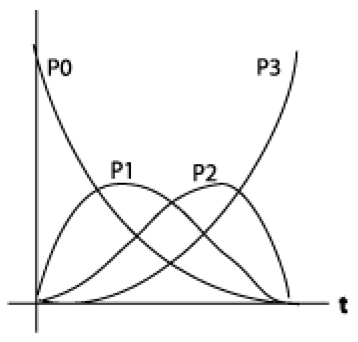
This has given us a form that resembles the one we had for the Hermit curve and we can set up the expression in a matrix form:

$$B(t) = T \cdot M_B \cdot G_B$$

where

$$T = [t^3, t^2, t, 1]\ , \quad M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \text{ and } \quad G_B = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

The blending functions are described like this:



$$P_0 = -t^3 + 3 \cdot t^2 - 3 \cdot t + 1$$

$$P_1 = 3 \cdot t^3 - 6 \cdot t^2 - 3 \cdot t$$

$$P_2 = -3 \cdot t^3 + 3 \cdot t^2$$

$$P_3 = t^3$$

Note that we can rewrite:

$$B(t) = -P_0(-t^3 + 3 \cdot t^2 - 3 \cdot t + 1) + P_1(3 \cdot t^3 - 6 \cdot t^2 + 3 \cdot t) + P_2(-3 \cdot t^3 + 3 \cdot t^2) + P_3(t^3)$$

to

$$B(t) = (1 - t)^3 \cdot P_0 + 3 \cdot t \cdot (1 - t)^3 \cdot P_1 + 3 \cdot t^2 \cdot (1 - t) \cdot P_2 + t^3 \cdot P_3$$

which is the normal description of the Bezier curve.

2 de Casteljau

We will use de Casteljaus algorithm to find the Bezier function

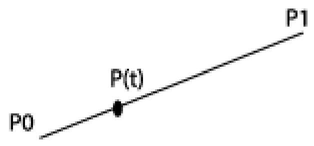
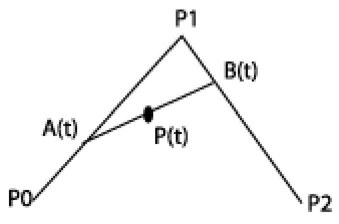
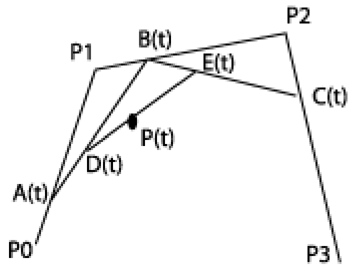
de Casteljau

The starting point is the parametrical form for a line:

$$P(0) = P_0$$
$$P(1) = P_1$$
$$P(t) = (1 - t) \cdot P_0 + t \cdot P_1$$

According to de Casteljaus algorithm we may consider this as special case of a Bezier curve, a linear Bezier curve.

The algorithm can be described after the following scheme, where we imagine that t runs from 0 to 1:

	$P(t) = (1 - t) \cdot P_0 + P_1$
	$P(t) = (1 - t) \cdot A(t) + t \cdot B(t)$ and $A(t) = (1 - t) \cdot P_0 + t \cdot P_1$ $B(t) = (1 - t) \cdot P_1 + t \cdot P_2$ giving $P(t) = (1 - t)^2 \cdot P_0 + 2 \cdot t \cdot (1 - t) \cdot P_1 + t^2 \cdot P_2$
	$P(t) = (1 - t) \cdot D(t) + t \cdot E(t)$ and $D(t) = (1 - t) \cdot A(t) + t \cdot B(t)$ $E(t) = (1 - t) \cdot B(t) + t \cdot C(t)$ and $A(t) = (1 - t) \cdot P_0 + t \cdot P_1$ $B(t) = (1 - t) \cdot P_1 + t \cdot P_2$ $C(t) = (1 - t) \cdot P_2 + t \cdot P_3$ giving: $P(t) = (1 - t)^3 \cdot P_0 + 3 \cdot t \cdot (1 - t)^2 \cdot P_1 + 3 \cdot t^2 \cdot (1 - t) \cdot P_2 + t^3 \cdot P_3$

The last expression is the same as the one we reached above, when we used a general polynomial of third degree and constraints for the endpoints and the derivative in the endpoints as a starting point.

It is easy to convince oneself that we can repeat the de Casteljau algorithm with further control points, P4, P5 etc.

3 The formula

The general form of the formula

The formula

The expression for the Bezier function is in general:

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

where n states the number of control points.

The description of the curve becomes:

$$B(t) = \sum_{i=0}^{n} B_i^n \cdot t^i \cdot (1-t)^{(n-i)} \cdot P_i$$

Coefficients, B_i^n are the same as the ones in Pascals triangle:

				1				
			1		1			
		1		2		1		
	1		3		3		1	
1		4		6		4		1

We have used n=4, and used the coefficients 1, 3, 3, 1.

4 Segments

How to connect curves to achieve "smoothness":

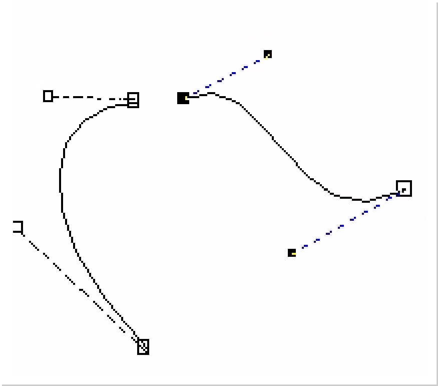
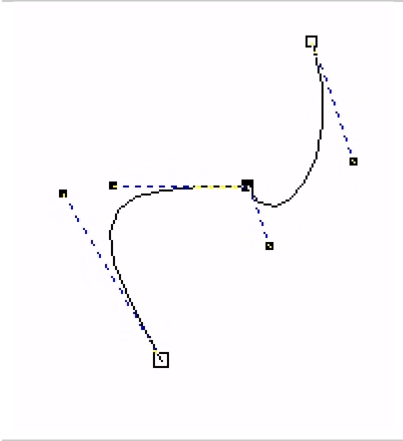
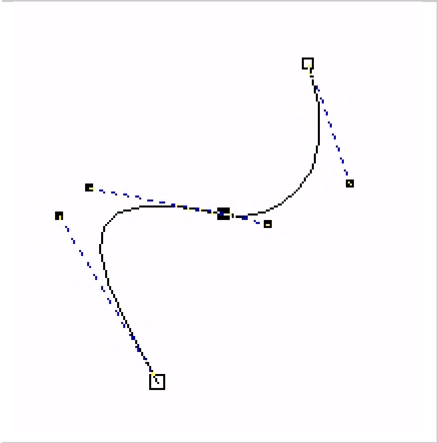
Segments

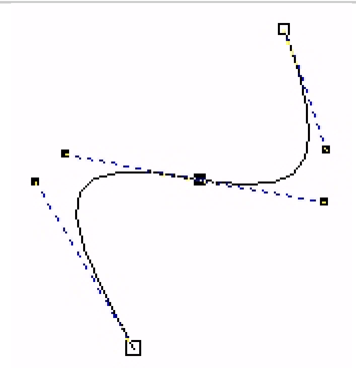
The general shape of the Bezier curve gives us the possibility to specify more control points. We can for example let n=5 and get:

$$B(t) = (1-t)^4 \cdot P_0 + 4 \cdot t \cdot (1-t)^3 \cdot P_1 + 6 \cdot t^2 \cdot (1-t)^2 + 4 \cdot t^3 \cdot (1-t) \cdot P_3 + t^4 \cdot P_4$$

We will not develop this reasoning to handle longer curves. We will in stead regard a curve as put together by several curve segments.

We want to look at how we can join curves and gain different degrees of smoothness. We will concentrate on what conditions that has to be fulfilled if two segments are going to hang together in a "smooth" way. We'll look at four degrees of geometric continuity. We'll regard Bezier curves and represent them as they are represented in drawing programs. The lines between the control points is represented as "handles" that we can use to change the shape of the curve.

	No continuity. The two curve segments don't have a common point.
	The two curve segments have a common point. The curves are connected, but with an obvious break in direction. The derivatives (for right and left segment) are different both in direction and size in this point.
	The two curve segments have a common point, and the joint is quite smooth. The derivatives in this point have the same direction, but not the same size.

	The two curve segments have a common point, and the joint is smooth. The derivatives in this point have the same direction and size.

5 OpenGL

We will see how we can realize a Bezier curve in OpenGL

Bezier curves is handled directly by OpenGL

```
// 4 control points in space
GLfloat ctrlpoints[4][3] = {
    {-4.0,-4.0,0.0},
    {-2.0,4.0,0.0},
    {2.0,-4.0,0.0},
    {4.0,4.0,0.0}    };
// we want to draw with t from 0.0 to 1.0
// and we give the dimensions of the data
glMap1f(GL_MAP1_VERTEX_3,0.0,1.0,3,4,& ctrlpoints[0][0]);
glEnable(GL_MAP1_VERTEX_3);
// draw a curve with 30 steps from t=0.0 to t=1.0
glBegin(GL_LINE_STRIP);
for (int i=0;i<=30;i++)
   glEvalCoord1f((GLfloat) i/30.0);
glEnd();
```

Example of a surface definition and drawing.

```
 // define ctrlpoints for trampoline
GLfloat ctrlpoints[4][4][3] = // [v][u][xyz]
{
 { {-1.5f,-1.5f,0.0f}, {-0.5f,-1.5f,0.0f},
   {0.5f,-1.5f,0.0f}, { 1.5f,-1.5f,0.0f}
 },
 { {-1.5f,-0.5f,0.0f}, {-0.5f,-0.5f,0.0f},
   {0.5f,-0.5f,0.0f}, {1.5f,-0.5f,0.0f}
 },
 {{-1.5f,0.5f,0.0f}, {-0.5f,0.5f,0.0f},
  {0.5f,0.5f,0.0f}, {1.5f,0.5f,0.0f}
 },
 { {-1.5f,1.5f,0.0f}, {-0.5f,1.5f,0.0f},
   {0.5f,1.5f,0.0f}, {1.5f,1.5f,0.0f}
 }
};
...
//  drawing
  glMap2f(GL_MAP2_VERTEX_3,0.0f,1.0f,3,4,0.0f,1.0f,12,4,
          & ctrlpoints[0][0][0]);
  glEnable(GL_MAP2_VERTEX_3);
  glMapGrid2f(20,0.0f,1.0f,20,0.0f, 1.0f);
  glEvalMesh2(GL_FILL, 0, 20, 0, 20);
```