1 Bezier Curves

Bezier curve is a *parametric* curve, which often used for producing a smooth approximation of a given set of data points.

1.1 Analytical expression

An analytical expression for the Bezier curve is given by:

$$\vec{B}(t) = \sum_{i=0}^{n} b_i^n(t)\vec{P}_i,$$

where $\vec{P}_i$ is the i–th control point (we have $n+1$ control points). And

$$b_i^n(t) = \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{(n-i)} t^i$$

is the i–th Bernstein polynomial of degree n.

Additionally to the expression with the Bernstein polynomials, one can use a recursion formula (**de Casteljau Algorithm**) for the construction of the Bezier–curve, which we will not cover here.

1.2 Surfaces

Analogically to Beziercurves, but with $n \cdot m$ Points $\vec{P}_{i,j}$ and the analytical expression

$$\vec{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} b_i^n(u) b_j^m(v)\vec{P}_{i,j}$$

one can define a *Bezier surface*

> Bezier–curves and surfaces may be unstable! Minor changes in control points might lead to major global changes!

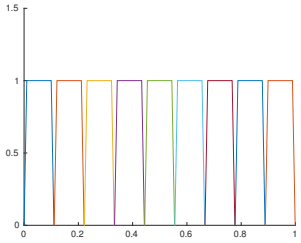
2 NURBS basis functions

Extending the idea, described in previous section, one could use NURBS basis functions instead of simple Bezier curves.

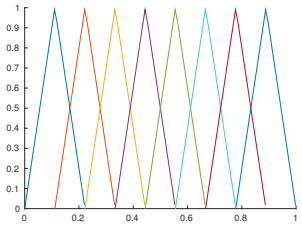
Unlike Bezier curves, for the B-spline basis a parameter domain is subdivided with, so-called, *knots*. In particular, given the parameter domain $[u_0, u_m]$ (in 1D), *knot vector* is given by $u_0 \leq u_1 \leq ... \leq u_m$. In most cases $u_0 = 0, u_m = 1$, so we get unit interval for our parameter values. Recall, that, N in NURBS stands for *non-uniform*. This means, that our knots $u_0, ..., u_m$ are not equidistant.

Given *knot vector* $[u_0, u_m]$ and a degree of B-spline p one can find i-th B-spline basis function recursively as follows:

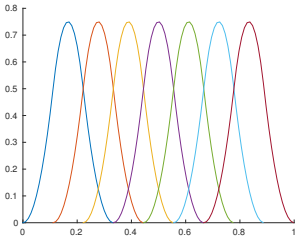
$$N_{i,0}(u) = \begin{cases} 1, & \text{if } u_i \leq u < u_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$



(a) B-spline basis for $p = 0$



(b) B-spline basis for $p = 1$



(c) B-spline basis for $p = 2$

Abbildung 1: B-spline basis functions

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2}$$

For $p = 0$ we get just step functions (see fig. 1a), for $p = 1$ we get familiar hat functions (see fig. 1b). Quadratic basis looks a bit more complicated, but also quite intuitive (fig. 1c).

Given these basis functions, Non-Uniform Rational B-Spline curve is given by:

$$C(u) = \frac{\sum_{i=1}^{k} N_{i,n} \omega_i P_i}{\sum_{i=1}^{k} N_{i,n} \omega_i}, \tag{3}$$

where k is number of points, $\{P_i\}$ are given control points.

B–splines are have the following properties, which are useful for our problem:

- degree n and number of control points $\vec{P}_{i\dots m}$ are independent!
- B–Splines only change locally (depends on the degree n) when a control point is changed.

Analogically, one can define B–spline surfaces.

3 Minimization problem

Now, once we defined all necessary tools, we proceed to the fitting problem.

The goal: Fit in a parametric curve to the set of given data points.

In our case our given set of points is a mesh, obtained on a previous step.

3.1 Minimization Problem: Bezier curve

First we want to find a Bezier–curve $\vec{B}_n(t)$ of degree n which is approximating a given spline $\vec{s}_m(t)$ defined by m points in a optimal way. For this purpose we want to minimize the L2–error. This leads to the minimization problem:

$$\text{find} \min_{\vec{B}_n \in \mathbb{B}_n} ||\vec{B}_n - \vec{s}_m||_{L_2} \tag{4}$$

The function space $\mathbb{B}_n$ is composed of functions of the following form:

$$\vec{B}_n(t) = \sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{5}$$

Therefore a control point for the Bezier–curve has the following form:

$$\vec{P}_i = \begin{pmatrix} a_{2i} \\ a_{2i+1} \end{pmatrix}. \tag{6}$$

Minimizing the L2–norm is equal to minimizing the functional:

$$F(\vec{B}_n) = \int_{t=0}^{t=1} \left(\vec{B}_n - \vec{s}_m\right)^2 \mathrm{d}t \tag{7}$$

Varying B_n yields

$$\delta F(B_n) = 2 \int_{t=0}^{t=1} \left(\vec{B}_n - \vec{s}_m\right)^T \delta\vec{B}_n \,\mathrm{d}t \tag{8}$$

using the definition of $\vec{B}_n$ and $\delta\vec{B}_n$

$$\vec{B}_n(t) = \sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{9}$$

$$\delta\vec{B}_n(t) = \sum_{i=0}^{n} \delta a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + \delta a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{10}$$

leads to

$$\delta F(B_n) = 2 \int_{t=0}^{t=1} \left(\sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} - \vec{s}_m \right)^T \left(\sum_{j=0}^{n} \delta a_{2j} \begin{pmatrix} b_j^n(t) \\ 0 \end{pmatrix} + \delta a_{2j+1} \begin{pmatrix} 0 \\ b_j^n(t) \end{pmatrix} \right) \mathrm{d}t \stackrel{!}{=} 0 \tag{11}$$

rewritten

$$\int_{t=0}^{t=1} \left(\sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \right)^T \left(\sum_{j=0}^{n} \delta a_{2j} \begin{pmatrix} b_j^n(t) \\ 0 \end{pmatrix} + \delta a_{2j+1} \begin{pmatrix} 0 \\ b_j^n(t) \end{pmatrix} \right) \mathrm{d}t$$

$$= \int_{t=0}^{t=1} \vec{s}_m^T \left(\sum_{j=0}^{n} \delta a_{2j} \begin{pmatrix} b_j^n(t) \\ 0 \end{pmatrix} + \delta a_{2j+1} \begin{pmatrix} 0 \\ b_j^n(t) \end{pmatrix} \right) \mathrm{d}t \tag{12}$$

or using the notation

$$\phi_{2i} = \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} \tag{13}$$

$$\phi_{2i+1} = \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{14}$$

we get

$$\int_{t=0}^{t=1} \left(\sum_{i=0}^{2n+1} a_i \phi_i \right)^T \left(\sum_{j=0}^{2n+1} \delta a_j \phi_j \right) \mathrm{d}t = \int_{t=0}^{t=1} \vec{s}_m^T \left(\sum_{j=0}^{2n+1} \delta a_j \phi_j \right) \mathrm{d}t \tag{15}$$

or

$$\sum_{i=0}^{2n+1} \sum_{j=0}^{2n+1} a_i \delta a_j \int_{t=0}^{t=1} \phi_i^T \phi_j \, \mathrm{d}t = \sum_{j=0}^{2n+1} \delta a_j \int_{t=0}^{t=1} \vec{s}_m^T \phi_j \, \mathrm{d}t \tag{16}$$

this has to be true for any j and δa_j, therefore

$$\sum_{i=0}^{2n+1} a_i \int_{t=0}^{t=1} \phi_i^T \phi_j \, \mathrm{d}t = \int_{t=0}^{t=1} \vec{s}_m^T \phi_j \, \mathrm{d}t \ \forall j = 0 \cdots 2n+1 \tag{17}$$

using

$$A_{ij} = \int_{t=0}^{t=1} \phi_i^T \phi_j \, \mathrm{d}t \text{ and } \ b_j = \int_{t=0}^{t=1} \vec{s}_m^T \phi_j \, \mathrm{d}t \tag{18}$$

we get the linear equation system

$$Aa = b \tag{19}$$

which has to be solved for a.

3.2 Minimization problem with least squares

Although the approach used above showed good results, it appears to be very computationally extensive. Analogically, one could reduce the original problem to the *linear regression problem*, which allows us to reduce computational costs. Also, to improve the locality of our solution, from now on we are going to use NURBS basis functions with weights $\omega_j = 1$ instead of Bezier curves.

Let X^0 be the $n \times 2$ matrix of the given set of points, N^p - the basis functions of degree p ($n \times (n+p)$ matrix, where n - number of points), P^0 - the control points ($(n+p) \times 2$ matrix).

The original problem can be written as:

$$X_i^0 = \sum_{j=1}^{n+p} P_j^0 N_{i,j}^p, \quad i \in \{1, .., n\} \tag{20}$$

Or, in short:

$$X^0 = N^p P^0 \tag{21}$$

The above system needs to be solved for the unknown P^0. One of the ways to solve it is to use SVD decomposition. So far, we used built-in MATLAB solver.

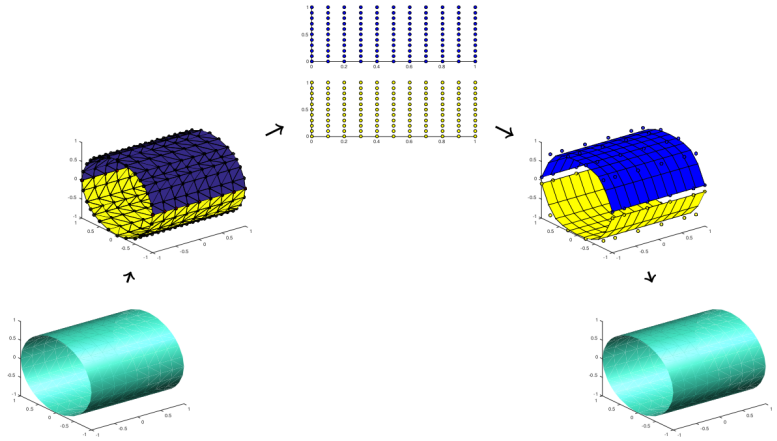


Abbildung 2: Fitting pipeline

3.3 Reparametrization

In our simple case we use coordinates of our points as a parameters, but even more advanced parametrization algorithms usually don't give an optimal parametrization. For handling this issue in (reference!!!) the following iterative algorithm was proposed:

1. *Fitting step:* For **fixed parametrizations** $\{t_i\}$, the optimal control points are found by solving a linear least-squares problem.
2. *Parameter correction step:* For **fixed control points**, optimal parametrizations $\{t_i\}$ are found by projecting the points onto our surface.

The first part of the algorithm is exactly the problem, we described in a previous section. The second part can be tackled by solving linearised system of equations, which provide us relatively cheap (we need to solve only a system of linear equations) and good (see fig.) method for the parameter correction.

4 Fitting pipeline

Since the geometry obtained after topology optimization can be arbitrary complex, we might not be able to find a food fit using only one patch. We seek a multi step algorithm, allowing us to break the overall big problem into smaller problems, which can be handled relatively easy. Based on the algorithm described in (reference to ...), our overall fitting pipeline looks as follows (see fig. 2):

- Patch selection (breaking our problem in small pieces which can be solved using least squares)
- Parametrization of obtained patches
- B–spline fitting using least squares
- Smooth connection of patches
- Conversion back to CAD

5 References

- Gerald Farin: NURBS
- Gerald Farin: Curver and Surfaces for CAGD