Master's thesis in
Applied Computer Science

CoolingGen

A parametric 3D-modeling software for turbine blade cooling geometries using NURBS

September 19, 2022

Institute for Numerical and Applied Mathematics
at the Georg-August-University Göttingen

Institute for Propulsion Technology at the
German Aerospace Center in Göttingen

Bachelor's and master's theses at the Center for
Computational Sciences at the
Georg-August-University Göttingen

Julian Lüken
`julian.lueken@dlr.de`

Georg-August-University Göttingen
Institute of Computer Science

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@cs.uni-goettingen.de

www.informatik.uni-goettingen.de

I hereby declare that this thesis has been written by myself and no other resources than those mentioned have been used.

[illegible]

Göttingen, September 19, 2022

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Zusammenfassung

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Contents

1 Introduction

1.1 Motivation

1.2 State of the Art

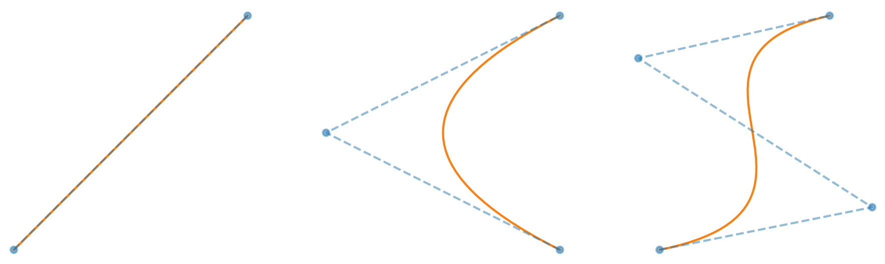
1.3 Problem Statement

2 Methods

As stated before, Non-Uniform Rational B-splines (NURBS) curves and surfaces are used to model geometries in CoolingGen. In this chapter we will first introduce Beziér curves. We will then generalize Beziér curves to obtain B-spline curves and surfaces. Applying an embedding map and a projection map to B-spline curves and surfaces, we acquire NURBS curves and surfaces, which are a generalization of B-spline curves and surfaces. Furthermore, CoolingGen uses special geometric algorithms such as point inversion, curve intersection and curve fillet fitting, which we will also present.

2.1 Bézier Curves

Bézier curves are named after the French engineer Pierre Bézier, who famously utilized them in the 1960s to design car bodies for the automobile manufacturer Renault [Béz68]. Today, they are used in a wide variety of vector graphics applications (i.e. in font representation on computers). At first glance, the definition of the Bézier curve might seem cumbersome, but given the mathematical foundation and a few graphical representations, it becomes apparent why they are such a powerful tool in computer-aided design.



(a) Degree 1. (b) Degree 2. (c) Degree 3.

Figure 2.1: Beziér curves of different degrees (orange) and their control points (blue).

2.1.1 Definition

Definition 2.1.1. The *Bernstein basis polynomials* of degree n on the interval $[t_0, t_1]$ are defined as

$$b_{n,k,[t_0,t_1]}(t) := \frac{\binom{n}{k}(t_1 - t)^{n-k}(t - t_0)^k}{(t_1 - t_0)^n}, \tag{2.1}$$

for $k \in \{0, \dots, n\}$.

Definition 2.1.2. A *Bézier curve* of degree n is a parametric curve $B_{P,[t_0,t_1]} : [t_0, t_1] \to \mathbb{R}^d$ that has a representation

$$B_{P,[t_0,t_1]}(t) = \sum_{k=0}^{n} b_{n,k,[t_0,t_1]}(t) P_k = \sum_{k=0}^{n} \frac{\binom{n}{k}(t_1 - t)^{n-k}(t - t_0)^k}{(t_1 - t_0)^n} P_k. \tag{2.2}$$

We call the elements of the set $P = \{P_0, P_1, \ldots, P_n\} \subset \mathbb{R}^d$ the *control points* of B_P.

Remark. Let $t_0 = 0$ and $t_1 = 1$. Then 2.2 simplifies to

$$b_{n,k}(t) := b_{n,k,[0,1]}(t) = \binom{n}{k}(1-t)^{n-k}t^k \tag{2.3}$$

and 2.1 simplifies to

$$B_P(t) := B_{P,[0,1]}(t) = \sum_{k=0}^{n} \binom{n}{k}(1-t)^{n-k}t^k P_k. \tag{2.4}$$

This case is the only case considered in this thesis.

2.1.2 De Casteljau's Algorithm

The computation of Equation 2.4 is usually performed using de Casteljau's algorithm. This is because the algorithm yields a simple implementation and lower complexity than straightforwardly computing Equation 2.4. The algorithm was proposed by Paul de Faget de Casteljau for the automobile manufacturer Citroën in the 1960s.

Algorithm 1 de Casteljau's algorithm

```
1:  Input
2:      $P = \{P_0, P_1, ..., P_n\}$        set of control points
3:      $t$                                 real number
4:  Output
5:      $P_0^{(n)} = B_P(t)$                the point on the Beziér curve w.r.t. to $t$
6:  procedure deCasteljau($P, t$)
7:      $P^{(0)} \leftarrow P$
8:      for $j = 1, 2, ..., n$ do
9:          for $k = 0, 1, ..., n - j$ do
10:             $P_k^{(j)} \leftarrow (1-t) \cdot P_k^{(j-1)} + t \cdot P_{k+1}^{(j-1)}$
11:     return $P_0^{(n)}$
```

Theorem 2.1.3. Algorithm 1 computes $B_P(t)$.

Proof. By induction. Let $n = 1$. Then

$$P_0^{(1)} = (1-t) \cdot P_0 + t \cdot P_1.$$

By employing the induction hypothesis

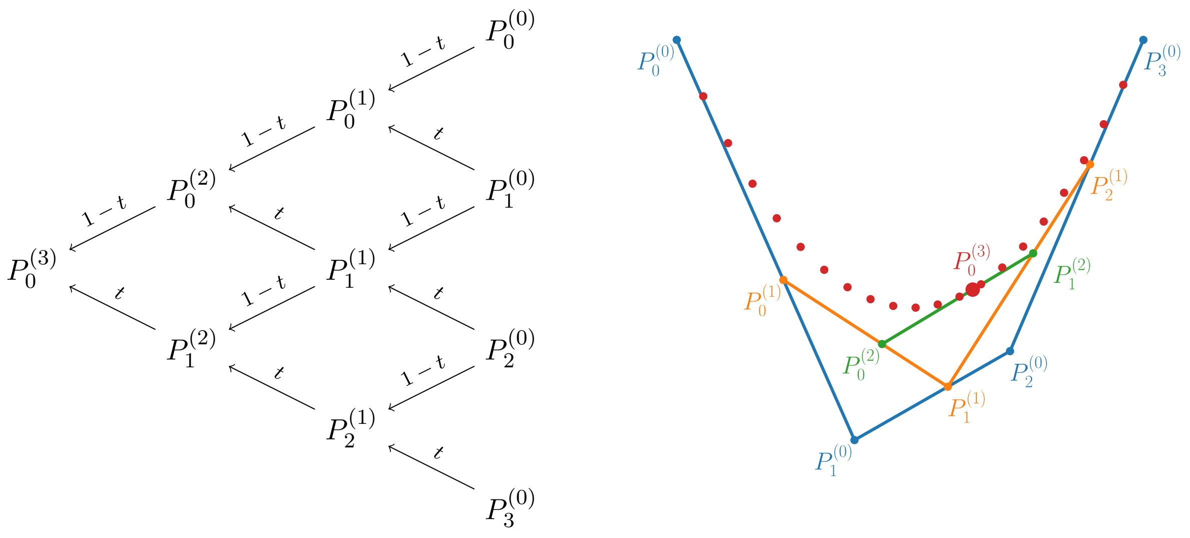
$$P_j^{(n)} = \sum_{k=j}^{n+j} \binom{n}{k}(1-t)^{n-k}t^k P_{j+k}$$

for some $n \in \mathbb{N}$, we can infer that

$$\begin{aligned}
P_0^{(n+1)} &= (1-t) \cdot P_0^{(n)} + t \cdot P_1^{(n)} \\
&= (1-t) \cdot \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_k + t \cdot \sum_{k=1}^{n+1} \binom{n}{k} (1-t)^{n-k} t^k P_{k+1} \\
&= \sum_{k=0}^{n+1} \binom{n+1}{k} (1-t)^{n+1-k} t^k P_k,
\end{aligned}$$

which is equal to $B_P(t)$ for degree $n+1$. $\square$

A visual representation of Algorithm 1 yields a triangular scheme. To compute one point on a Beziér curve B_P with degree n, one has to perform $\frac{n^2-n}{2}$ vector additions and $n^2 - n$ scalar multiplications.



(a) Algorithmic visualization.

(b) Geometric visualization of $B_P(\frac{3}{5})$.

Figure 2.2: Visual representations of de Casteljau's algorithm.

Interestingly, the representation of the algorithm in Figure 2.2 also gives rise to an intuitive visualization of the geometric shape of the Beziér curve B_P. For all $i \in \{0, ..., n\}$ and all $j \in \{0, ..., n-i\}$, the point $P_j^{(i+1)}$ is the convex combination (always w.r.t. t) of $P_j^{(i)}$ and $P_{j+1}^{(i)}$. Thus $P_j^{(i+1)}$ always lies on the line segment between $P_j^{(i)}$ and $P_{j+1}^{(i)}$, as can be observed in the example in Figure 2.2.

2.1.3 Properties

Other than being remarkably intuitive, Beziér curves have a lot of properties which make them convenient. In computer-aided design software, most graphical user interfaces rely on the principle of letting the user interactively drag and drop the control points with a mouse, granting them control over the shape of Beziér curve. The following theorems further illustrate why this is a good idea.

Theorem 2.1.4. $B_P(0) = P_0$ and $B_P(1) = P_n$.

Proof. Explicit computation yields

$$B_P(0) = \sum_{k=0}^{n} \binom{n}{k} t^k P_k = \binom{n}{0} P_0 = P_0$$

and

$$B_P(1) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} P_k = \binom{n}{n} P_n = P_n.$$

□

Theorem 2.1.5. Let $T \in \mathbb{R}^{3\times 3}$. Then $B_{TP}(t) = TB_P(t)$ where $TP := \{TP_0, TP_1, ..., TP_n\}$.

Proof. For all $t \in [0,1]$ we can directly compute

$$TB_P(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k T P_k = B_{TP}(t).$$

□

Theorem 2.1.6. $B_P(t)$ lies in the convex hull of P for all $t \in [0,1]$.

Proof. By the algorithm of de Casteljau (1), we know that $P_j^{(i)} = (1-t) \cdot P_j^{(i-1)} + t \cdot P_{j+1}^{(i-1)}$ for all $t \in [0,1]$. Therefore, $P^{(i)}$ lie in the convex hull of $P^{(i-1)}$. But then $B_P(t) = P_0^{(n)}$ always lies in the convex hull of $P^{(0)} = P$ by induction. □

Simple as their appearance may be, Beziér curves fall short of representing some of the most common geometric shapes. Given a finite number of control points, we can never make $B_P(t)$ a circular arc, although a circle has a very simple parametric form. One of their greatest perks, the ability to describe a shape with just a handful of control points, is their greatest shortcoming at the same time. This is most likely the reason why Beziér curves are not the state of the art in technical engineering applications. However, Beziér curves certainly do provide an intuition for Non-Uniform Rational B-Splines or NURBS, which is their prevailing counterpart.

2.2 Non-Uniform Rational B-splines (NURBS)

NURBS are a state of the art tool for curve and surface modelling. There is a somewhat common joke that describes the acronym NURBS as *"Nobody Understands Rational B-Splines"* (source?). In this section, we invalidate this punch line. First of all, we discuss B-splines, then we construct B-spline curves and surfaces and then we apply simple transformations to the B-spline curves and surfaces to acquire NURBS curves and surfaces.

2.2.1 Definition

Similarly to how Beziér curves are defined on the Bernstein polynomial basis, NURBS are defined on basis functions called basis splines or, more commonly, B-splines.

Definition 2.2.1. A *knot sequence* $(t_m)_{m=-\infty}^{\infty} \subset \mathbb{R}$ is a sequence with $t_m \leq t_{m+1}$ for all $m \in \mathbb{Z}$.

Definition 2.2.2. The *B-splines of degree* 0 on a knot sequence (t_m) are defined as

$$N_{1,k}^{(t_m)}(t) := \begin{cases} 1 & \text{if } t \in [t_k, t_{k+1}), \\ 0 & \text{else.} \end{cases} \tag{2.5}$$

The *B-splines of degree* $p-1$ with $p > 1$ are given by the *Cox-de-Boor recursion formula*

$$N_{p,k}^{(t_m)}(t) := \omega_{p-1,k}^{(t_m)}(t)\, N_{p-1,k}^{(t_m)}(t) + (1 - \omega_{p-1,k+1}^{(t_m)}(t))\, N_{p-1,k+1}^{(t_m)}(t), \tag{2.6}$$

where

$$\omega_{p,k}^{(t_m)}(t) := \begin{cases} \frac{t-t_k}{t_{k+p}-t_k} & \text{if } t_{k+p} \neq t, \\ 0 & \text{else.} \end{cases} \tag{2.7}$$

Remark. Instead of $N_{p,k}^{(t_m)}$ we write $N_{p,k}$ and explicitly refer to (t_m) when necessary. We restrict the domain of definition of $N_{p,k}$ to $[0,1]$ by setting $\lim_{m\to-\infty} t_m = 0$ and $\lim_{m\to\infty} t_m = 1$.

Definition 2.2.3. A *B-spline curve* of degree $p-1$ over a set of control points $P = \{P_0, P_1, ...P_n\} \subset \mathbb{R}^d$ and a knot sequence (t_m) is defined as

$$S_P(t) = \sum_{k=0}^{n} N_{p,k}(t) P_k.$$

Definition 2.2.4. A *NURBS curve* $C_P(t)$ of degree $p-1$ with the control points $P = \{P_0, P_1, ..., P_n\} \subset \mathbb{R}^d$, the control weights $w = (w_0, w_1, ...w_n) \subset \mathbb{R}$ and a knot sequence (t_m) is defined as

$$C_P(t) = \frac{\sum_{k=0}^{n} N_{p,k}(t) w_k P_k}{\sum_{k=0}^{n} N_{p,k}(t) w_k}. \tag{2.8}$$

Remark. Let $P \subset \mathbb{R}^d$. A NURBS curve can alternatively be understood as a projection of a B-spline curve on a transformed set of control points. For this purpose we define the embedding into the weighted vector space

$$\Phi_w : \mathbb{R}^d \to \mathbb{R}^{d+1}$$

that maps each control point $P_k = (p_1, ..., p_d) \in \mathbb{R}^d$ onto $(w_k p_1, ..., w_k p_d, w_k) \in \mathbb{R}^{d+1}$. We also have to define the projection map

$$\Phi^\dagger : \mathbb{R}^{d+1} \to \mathbb{R}^d$$

that maps each point on the B-spline curve $S_{\Phi(P)}(t) = (s_1, ..., s_d, s_{d+1})$ onto $(\frac{s_1}{s_{d+1}}, ..., \frac{s_d}{s_{d+1}})$. We can then define the NURBS curve as

$$C_P(t) = \Phi^\dagger(S_{\Phi_{w(P)}}(t))$$

which we will make use of later on.

Theorem 2.2.5. Let $w \equiv 1$. Then $S_P \equiv \Phi^\dagger(S_{\Phi_{w(P)}})$. In other words, NURBS curves are a generalization of B-spline curves.

Proof. Since in this case $\Phi_w((p_1, ..., p_d)) = (p_1, ..., p_d, 1)$, we have that $\Phi^\dagger(S_{\Phi_w(P)}(t)) = \Phi^\dagger(\Phi_w(S_P(t))) = S_P(t)$. $\square$

Now that we have defined B-spline curves and NURBS curves, we can define B-spline surfaces and NURBS surfaces in a similar manner. To do this, we require a grid of knots represented by two knot sequences $(u_m)_{m=-\infty}^{\infty}$ and $(v_m)_{m=-\infty}^{\infty}$ which satisfy the same conditions as (t_m) did before. Instead of a one-dimensional set of control points, the surface definition relies on a two-dimensional set of control points. Notice that this grants us control

Definition 2.2.6. A *B-spline surface* $S_P(u, v)$ of degree $(p-1, q-1)$ over a set of control points $P = \{P_{i,j} : (i, j) \in \{0, 1, ..., n_u\} \times \{0, 1, ..., n_v\}\} \subset \mathbb{R}^d$ on the knot grid $(u_m), (v_m)$ is defined as

$$S_P(u, v) = \sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) P_{k_u,k_v},$$

where $N_{p,k_u}(u) := N_{p,k_u}^{(u_m)}(u)$ and $N_{q,k_v}(v) := N_{q,k_v}^{(v_m)}(v)$.

Definition 2.2.7. A *NURBS surface* $C_P(u, v)$ of degree $(p-1, q-1)$ over a set of control points $P = \{P_{i,j} : (i, j) \in \{0, 1, ..., n_u\} \times \{0, 1, ..., n_v\}\} \subset \mathbb{R}^d$, the control weights $w = \{w_{i,j} : (i, j) \in \{0, 1, ..., n_u\} \times \{0, 1, ..., n_v\}\} \subset \mathbb{R}$ and the knot grid $(u_m), (v_m)$ is defined as

$$C_P(u, v) = \frac{\sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) w_{k_u,k_v} P_{k_u,k_v}}{\sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) w_{k_u,k_v}},$$

where $N_{p,k_u}(u) := N_{p,k_u}^{(u_m)}(u)$ and $N_{q,k_v}(v) := N_{q,k_v}^{(v_m)}(v)$.

Remark. As with NURBS curve, we have the analogous result

$$C_P(u, v) = \Phi^\dagger(S_{\Phi_w(P)}(u, v))$$

for NURBS surfaces.

The notion of the knot sequence (t_m) is commonly computationally simplified to that of a knot vector τ, since τ only contains a finite number of elements. For our purposes, we let

$$\tau = (\underbrace{t_0, ..., t_{p-1}}_{=0}, t_p, ..., t_n, \underbrace{t_{n+1}, ..., t_{n+p}}_{=1}),$$

where $t_k = 0$ for $k \in \{0, ..., p-1\}$ and $t_k = 1$ for $k \in \{n+1, ..., n+p\}$. We still require the monotony property $t_k \leq t_{k+1}$ for all $k \in \{0, ..., n+p\}$.

2.2.2 De Boor's Algorithm

To efficiently calculate points on a B-spline object, Carl-Wilhelm Reinhold de Boor devised an efficient algorithm, the construction of which demonstrates its correctness. Together with the embedding Φ_w and the projection $\Phi^\dagger$ from the Remark for Definition 2.2.4, this algorithm can also be used to calculate points on a NURBS object.

Let $P = \{P_0, P_1, ..., P_n\}$ a set of control points, (t_m) a knot sequence and $p-1$ the degree of the B-spline curve $S(t)$. Then by the Cox-de-Boor recursion formula, we find

$$\begin{aligned} S(t) &= \sum_{k=0}^{n} N_{p,k}(t)\, P_k \\ &= \sum_{k=1}^{n} \omega_{p-1,k}(t)\, N_{p-1,k}(t)\, P_k \;+\; \sum_{k=0}^{n} (1-\omega_{p-1,k+1}(t))\; N_{p-1,k+1}(t)\, P_k. \end{aligned}$$

Changing the limits of the second term, we can summarize the two terms as

$$\begin{aligned} S(t) &= \sum_{k=1}^{n} N_{p-1,k}(t)\; \underbrace{\Big[\omega_{p-1,k}(t)\, P_k \;+\; (1-\omega_{p-1,k}(t))\; P_{k-1}\Big]}_{=:P_k^{(1)}(t)} \\ &= \sum_{k=1}^{n} N_{p-1,k}(t)\, P_k^{(1)}(t). \end{aligned}$$

Recursively defining

$$P_k^{(j)}(t) := \begin{cases} \omega_{p-j,k}\, P_k^{(j-1)}(t) \;+\; (1-\omega_{p-j,k})\, P_{k-1}^{(j-1)}(t) & \text{if } j > 0, \\ P_k & \text{else,} \end{cases} \tag{2.9}$$

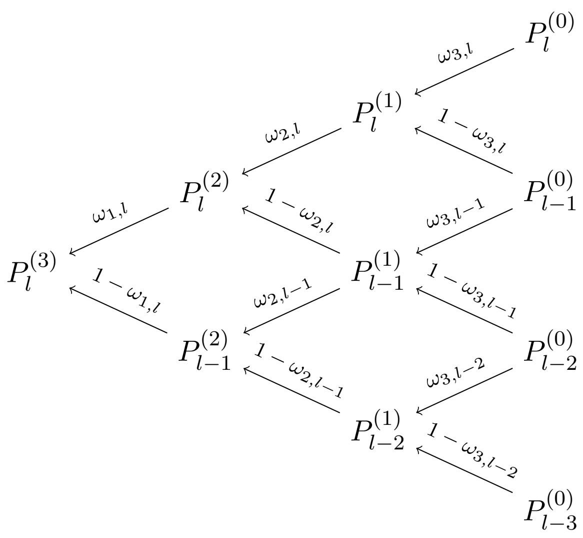
we can repeat this process up to $p-2$ more times, finding

$$S(t) = \sum_{k=p-1}^{n} N_{1,k}(t)\, P_k^{(p-1)}(t) = P_l^{(p-1)}(t)$$

for $t \in [t_l, t_{l+1})$. We can thus use the recursive definition in Equation 2.9 as the key step of our algorithm to compute $S(t)$. It is already clear from this place that the points calculated on a B-spline curve are in fact also a cumultated convex combination of control points, just as it is the case with Beziér curves. As zero-values for $\omega_{i,j}$ (or $1-\omega_{i,j}$) can be completely omitted by checking multiplicity, we arrive at the following algorithm:

Algorithm 2 de Boor's algorithm for curves

```
1: Input
2:     $P = \{P_0, P_1, ..., P_n\}$            set of control points of the B-spline curve
3:     $\tau = (t_0, t_1, ..., t_{n+p})$        knot vector of the B-spline curve
4:     $p - 1$                                  degree of the B-spline curve
5:     $t \in [t_0, t_{n+p})$                   real number
6: Output
7:     $S_P(t)$                                 the point on the B-spline curve w.r.t. to $t$
8: procedure deBoorCurve($P, p, \tau, t$)
9:     Find $l$ such that $t \in [t_l, t_{l+1})$
10:    Let $m$ be the multiplicity of $t$ in the knot vector $\tau$
11:    $P^{(0)} \leftarrow P$
12:    for $j = 1, 2, ..., p - m - 1$ do
13:        for $k = l - p + j + 1, ..., l - m$ do
14:            $\omega_{p-j,k} \leftarrow \frac{t - t_k}{t_{k+p-j} - t_k}$
15:            $P_k^{(j)} \leftarrow (1 - \omega_{p-j,k}) \cdot P_{k-1}^{(j-1)} + \omega_{p-j,k} \cdot P_k^{(j-1)}$
16:    return $P_{l-m}^{(p-m-1)} = C_P(t)$
```


$$\begin{tikzcd}[column sep=small]
 & & & P_l^{(0)} \arrow[dl, "\omega_{3,l}"'] \\
 & & P_l^{(1)} \arrow[dl, "\omega_{2,l}"'] & \\
 & P_l^{(2)} \arrow[dl, "\omega_{1,l}"'] & & P_{l-1}^{(0)} \arrow[uul, "1-\omega_{3,l}"] \arrow[dl, "\omega_{3,l-1}"'] \\
P_l^{(3)} & & P_{l-1}^{(1)} \arrow[uul, "1-\omega_{2,l}"] \arrow[dl, "\omega_{2,l-1}"'] & \\
 & P_{l-1}^{(2)} \arrow[uul, "1-\omega_{1,l}"] & & P_{l-2}^{(0)} \arrow[uul, "1-\omega_{3,l-1}"] \arrow[dl, "\omega_{3,l-2}"'] \\
 & & P_{l-2}^{(1)} \arrow[uul, "1-\omega_{2,l-1}"] & \\
 & & & P_{l-3}^{(0)} \arrow[uul, "1-\omega_{3,l-2}"]
\end{tikzcd}$$

(a) Visualization of de Boor's algorithm on a curve $S_P(t)$ with degree 3, where t does not appear in the knot vector τ (in other words, $m = 0$.

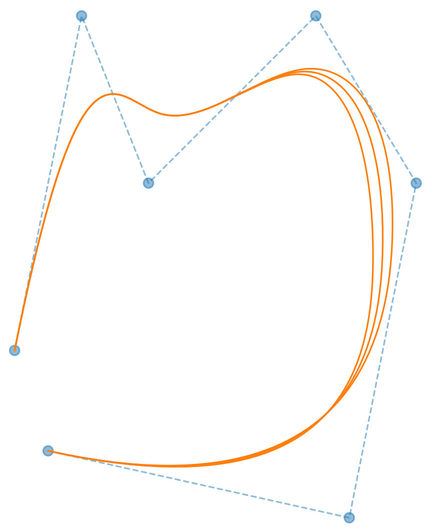

(b) Degree 3 NURBS curve $\Phi^\dagger(S_{\Phi_w(P)}(t))$ for different values of a single entry of the control weight vector w.

Figure 2.3: Calculating points on a NURBS curve.

To calculate one point on the B-spline curve S_P of degree p, we require at most $\frac{p^2-p}{2}$ vector additions and $p^2 - p$ scalar multiplications, which is quite similar to what we found for Beziér curves. However, we also need to calculate $\omega_{j,k}$ at every step, which in total sums up to $p^2 - p$ real additions and $\frac{p^2-p}{2}$ real multiplications.

By writing the definition of the B-spline surface as

$$S_P(u,v) = \sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) P_{k_u,k_v} = \sum_{k_u=0}^{n_u} N_{p,k_u}(u) \underbrace{\left(\sum_{k_v=0}^{n_v} N_{q,k_v}(v) P_{k_u,k_v} \right)}_{:= q_{k_u}(v)},$$

we can observe that Algorithm 2 can be utilized to calculate points on B-splines surfaces. To do this, we run the de Boor's algorithm to calculate $q_{k_u}(v)$.

Algorithm 3 de Boor's algorithm for surfaces

```
 1: Input
 2:     $P = \{P_{i,j}\}$                        set of control points with $n_1 \cdot n_2$ elements
 3:     $\tau_u = (u_0, u_1, ..., u_{n_u+p})$     first knot vector of the B-spline curve
 4:     $\tau_v = (v_0, v_1, ..., v_{n_v+q})$     second knot vector of the B-spline curve
 5:     $p - 1$                                  first degree of the B-spline curve
 6:     $q - 1$                                  second degree of the B-spline curve
 7:     $u \in [u_0, u_{n_u+p})$                 first real number
 8:     $v \in [v_0, v_{n_v+q})$                 second real number
 9: Output
10:     $S_P(u,v)$                               the point on the B-spline curve w.r.t. to $u, v$
11: procedure DEBOORSURFACE($P, p, q, \tau_u, \tau_v, u, v$)
12:     Find $l$ such that $u \in [u_l, u_{l+1})$
13:     Let $m$ be the multiplicity of $u$ in the knot vector $\tau_u$
14:     for $k = l - p + 1, , ..., l - m - 1$ do
15:         $Q_k \leftarrow$ deBoorCurve($P_{k,\cdot}, q, \tau_v, v$)
16:     return deBoorCurve($Q, p, \tau_u, u$)
```

Utilizing the notation with the embedding map Φ_w and the projection map $\Phi^\dagger$, we can calculate points on NURBS objects with these two algorithms. To do this, we employ the following steps on a set of control points P and the weights w:

1. Calculate $P_w = \Phi_w(P)$.
2. Use de Boor's algorithm to calculate points on S_{P_w}.
3. Project points onto $\mathbb{R}^d$ by applying $\Phi^\dagger$ to S_{P_w} to find C_P.

2.2.3 Properties

In this section, we will see that B-spline curves and NURBS curves share some (although not all) useful properties with Beziér curves.

Theorem 2.2.8. Let S_P a B-spline curve and C_P a NURBS curve. Then $S_P(0) = P_0$ and $S_P(1) = P_n$ and therefore $C_P(0) = P_0$ and $C_P(1) = P_n$.

Proof. By using Algorithm 2, we have $m = p$ and return P_l without calculation, which is P_0 in the case of $t = 0$ and P_n in the case of $t = 1$. Then by $C_P(t) = \Phi^\dagger(S_{\Phi_w(P)}(t))$, we have $C_P(0) = \Phi^\dagger\Phi_w(P_0) = P_0$ and $C_P(1) = \Phi^\dagger\Phi_w(P_n) = P_n$. □

Theorem 2.2.9. Let $T \in \mathbb{R}^{d\times d}$. Let S_P be a B-spline curve or surface and C_P a NURBS curve or surface. Then $S_{TP} = TS_P$ and $C_{TP} = TC_P$.

Proof. By linearity of T we can compute

$$S_{TP}(t) = \sum_{k=0}^{n} N_{p,k}(t) T P_k = T\left(\sum_{k=0}^{n} N_{p,k}(t) P_k\right) = TS_P(t)$$

for B-spline curves and

$$C_{TP}(t) = \frac{\sum_{k=0}^{n} N_{p,k}(t) w_k T P_k}{\sum_{k=0}^{n} N_{p,k}(t) w_k} = T\left(\frac{\sum_{k=0}^{n} N_{p,k}(t) w_k P_k}{\sum_{k=0}^{n} N_{p,k}(t) w_k}\right) = TC_P(t)$$

for NURBS curves. Similarly, we can compute

$$S_{TP}(u,v) = \sum_{k_u=0}^{n_u}\sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) T P_{k_u,k_v} = TS_P(u,v)$$

for B-spline surfaces and

$$C_{TP}(u,v) = \frac{\sum_{k_u=0}^{n_u}\sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) w_{k_u,k_v} T P_{k_u,k_v}}{\sum_{k_u=0}^{n_u}\sum_{k_v=0}^{n_v} N_{p,k_u}(u) N_{q,k_v}(v) w_{k_u,k_v}} = TC_P(u,v)$$

for NURBS surfaces. □

Theorem 2.2.10. A B-spline curve or surface S_P lies in the convex hull of its control points P.

Proof. Using Equation 2.9, we can see that every iteration of Algorithm 2 produces a set of points which all lie in the convex hull of the last iteration. Therefore, $S_P(t) = P_l^{(p-1)}$ lies in the convex hull of $P^{(0)}$ for all l. □

Remark. This theorem does not hold for NURBS curves and surfaces C_P.

2.3 Methods on NURBS Objects

2.3.1 Point Inversion

Let $\gamma(t) \subset \mathbb{R}^d$ a curve, $t \in [0,1]$, and let $Q \in \mathbb{R}^d$ a point. We want to find an element in the set

$$T^* := \arg\min_{t\in[0,1]} D(\gamma(t), Q),$$

where $D(\cdot,\cdot)$ is the Euclidean distance between two points. Notice that

$$\min_{t\in[0,1]} D(\gamma(t), Q) = D(\gamma(t^*), Q),$$

is the distance between the curve γ and the point Q, and is constant for all $t^* \in T^*$.

Definition 2.3.1. The distance between a curve $\gamma \subset \mathbb{R}^d$ and a point $Q \in \mathbb{R}^d$ is given by

$$D(Q, \gamma) := D(\gamma, Q) := \min_t D(\gamma(t), Q).$$

Remark. If $D(Q, \gamma) = 0$, then by the point-separating property of the Euclidean norm, Q lies on the curve γ.

Theorem 2.3.2. Let $\gamma(t) \subset \mathbb{R}^d$ a curve, $t \in [0, 1]$ and let $Q \in \mathbb{R}^d$. If γ has no self-intersections and

$$\min_{t\in[0,1]} D(\gamma(t), Q) = 0$$

then T^* is a single element set.

Proof. If γ has no self-intersections, then no two parameters of γ will map onto the same point. Therefore, γ is injective. We know t^* exists such that $D(\gamma(t^*), Q) = 0$, which by the point-separating property of the Euclidean norm is equivalent to $Q = \gamma(t^*)$. By injectivity, this t^* is unique. □

Minimizing distance functions for general curves is, in general, a problem that is hard to solve. We therefore make some assumptions about the point-curve distance function $\phi(t) := D(\gamma(t), Q)$ that we are going to minimize in order to find one target parameter $t^* \in T^*$. If ϕ is convex and differentiable, then the problem statement has a well-known and easy to implement solution, namely Newton's method, which for an initial guess $t^{(0)}$ is given by the iteration

$$t^{(n+1)} = t^{(n)} - \frac{\phi(t^{(n)})}{\phi'(t^{(n)})}.$$

Here, $\phi'(t)$ is the derivative of $\phi(t)$ with respect to t, which for a small value of $\epsilon > 0$ can be approximated by the right-sided difference quotient

$$\phi'_r(t) = \frac{\phi(t + \epsilon) - \phi(t)}{\epsilon}.$$

However, to ensure differentiability of ϕ adjacent to t, it is recommended to compare the value of the right-sided difference quotient to

$$\phi'_l(t) = \frac{\phi(t) - \phi(t - \epsilon)}{\epsilon},$$

which is the left-sided difference-quotient. If $\phi'_r(t) = \phi'_l(t)$, then we can assume $\phi'_l(t) \approx \phi(t) \approx \phi'_r(t)$.

Although in CoolingGen, we often cannot make the assumption of differentiability or convexity of ϕ. We therefore employ a pragmatic scheme and assume continuity, piecewise differentiability and piecewise convexity of ϕ and run Newton's method on a small subinterval $[l, u] \subset [0, 1]$.

To find $[l, u]$, we sample the curve γ at a set of points $\{t_0, t_1, ..., t_n\}$ with $t_i = i/n$ and seek

$$m = \underset{t \in \{t_0, t_1, ..., t_n\}}{\arg\min} \ \phi(t).$$

We let $l = m - \frac{i}{n}$ and $u = m + \frac{i}{n}$. Next, we can use Newton's method with the start value m to find

$$t^* = \underset{t \in [l,u]}{\arg\min} \ \phi(t).$$

This scheme will only converge to the global minimum of ϕ for large enough $n \in \mathbb{N}$.

We describe the procedure in pseudo-code. The start and boundary value search is described by the following algorithm.

Algorithm 4 Start value search

```
1:  Input
2:      $N$                  number of samples for initial value search
3:  procedure findPointInversionInitValues($\gamma, Q$)
4:      $m \leftarrow 0$
5:      for $k = 1, ..., N$ do
6:          $t \leftarrow \frac{k}{N}$
7:          if $D(Q, \gamma(m)) > D(Q, \gamma(t))$ then $m \leftarrow t$
8:      return $m - \frac{i}{N}, m + \frac{i}{N}, m$
```

Using the start and boundary values, we apply Newton's method for M iterations to find our target value, which is returned upon completion.

Algorithm 5 Point Inversion

```
1:  Input
2:      $M$                  number of iterations
3:  procedure pointInversion($\gamma, Q$)
4:      $l, u, m \leftarrow$ findPointInversionInitValues($\gamma, Q$)
5:      $t_0 \leftarrow m$
6:      for $k = 1, 2, ..., M$ do
7:          $t_1 \leftarrow t_0 + \epsilon$
8:          $\phi_0 \leftarrow D(\gamma(t_0), Q)$ and $\phi_1 \leftarrow D(\gamma(t_1), Q)$
9:          $\phi' \leftarrow \frac{\phi_1 - \phi_0}{\epsilon}$
10:         $t_0 \leftarrow t_0 - \frac{\phi_0}{\phi'}$
11:         if $t_0 < l$ then $t_0 \leftarrow l$
12:         if $t_0 > u$ then $t_0 \leftarrow u$
13:     return $t_0$
```

Given that the target distance is zero and γ is a NURBS curve, we can observe that the devised scheme is the inversion of de Boor's algorithm (Algorithm 2), that will return one input parameter of the curve that fits the point Q (in fact, the only input parameter if we also have injectivity of γ). This justifies the name *point inversion algorithm*.

If the target distance is not equal to zero or the curve is not injective, the algorithm still proves effective to find the distance $D(Q, \gamma)$, since the minimum distance is the same for all $t^* \in T^*$.

2.3.2 Ray-Curve Intersection in 2D

Definition 2.3.3. A *ray* with support vector $A \in \mathbb{R}^d$ and direction vector $B \in \mathbb{R}^d$ is given by the set of points

$$\hat{R}_{A,B} = \{A + tB \;:\; t \in \mathbb{R},\ t \geq 0\}.$$

The dependence of a point on the ray on the real parameter t motivates the description of this set as a map

$$R_{A,B} : \mathbb{R} \to \mathbb{R}^d, \quad t \mapsto A + tB.$$

which yields the equation $\hat{R}_{A,B} = R_{A,B}([0, \infty))$.

In this section, we want to find an algorithm that intersects a curve $\gamma(t)$ with $t \in [0, 1]$ with a ray $R_{A,B}(s)$ with $s \in [0, \infty)$. Generally, the intersection of these two sets can be written as

$$I = \gamma([0, 1]) \cap R_{A,B}([0, \infty)).$$

The algorithm presented will return only up to one element in that set, specifically the one corresponding to the lowest value of the ray parameter s. If however $I = \emptyset$, then no element will be returned.

Definition 2.3.4. Let $R_{A,B}$ a ray and $\gamma(t)$ a curve. Then the ray-curve-distance w.r.t $R_{A,B}$ and γ is defined as

$$D(R_{A,B}, \gamma) := D(\gamma, R_{A,B}) := \min_{s \in [0,\infty)} D(\gamma, R_{A,B}(s))$$

Definition 2.3.5. Let $R_{A,B}(s)$ a ray and $\gamma(t)$ a curve. If a solution to

$$\min_{s \in [0,\infty)} s \quad \text{subject to} \quad D(\gamma, R_{A,B}(s)) = 0$$

exists, then it is called *collision parameter* of $R_{A,B}$ w.r.t. γ.

We define the return value of our algorithm as the one element set given by

$$I_{\min} := \begin{cases} \{R_{A,B}(s^*)\} & \text{if } D(R_{A,B}(s^*), \gamma) \leq \epsilon, \\ \emptyset & \text{otherwise,} \end{cases}$$

where the absolute tolerance $\epsilon > 0$ is close to 0 and s^* is an approximation of the collision parameter of $R_{A,B}$ w.r.t. γ. If $I_{\min} \neq \emptyset$, then we refer to the element in $I_{\min}$ as *collision point*.

Remark. Why is the notion of the collision point of interest? To exemplify the prospect of our algorithm, imagine the following scenario: Let there be a reflective interface represented by a curve γ and a beam of light represented by a ray $R_{A,B}$. Then our model suggests the equivalence of the physical point of reflection and the collision point of the ray with respect to the curve.

We can find the approximation s^* of the collision parameter by using a common CAD scheme called *ray marching*, in which we traverse the ray $R_{A,B}$ by a small distance increments s until we achieve collision with the curve γ. How do we choose the optimal value for s? The idea is the following: Using the point inversion algorithm, we can calculate the *safe distance* s of the point A on the ray to the curve γ. Then we can easily construct a point P that fulfills $D(A, P) = s$ and lies on the ray $R_{A,B}$ by

$$P := A + s\frac{B}{||B||}.$$

Now there are only two possibilities: Either P lies within the ϵ-ball of the collision point, or the line segment $\overline{AP} \subset R_{A,B}([0, \infty))$ does not intersect with γ. In the first case, we can return $\{P\}$. In the second case, we can repeat the procedure on the set

$$R_{A,B}([0, \infty)) \setminus \overline{AP} = R_{A,B}([s, \infty)) = R_{P,B}([0, \infty)).$$

If after a certain amount of iterations N the algorithm does not converge to a point which lies within the ϵ-ball of the collision point, we return $\emptyset$.

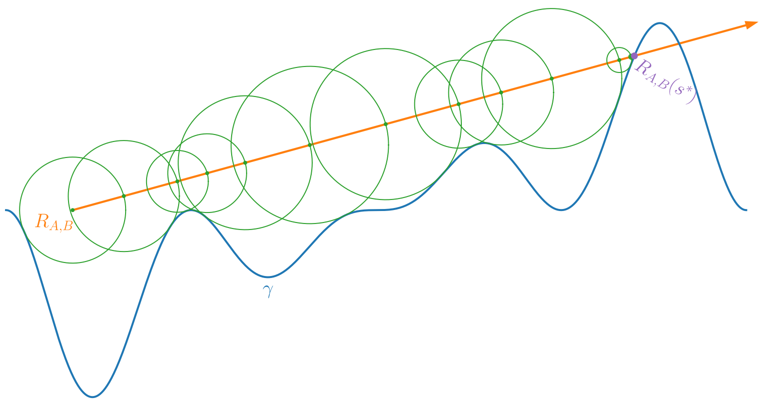


Figure 2.4: The ray marching algorithm.

Figure 2.4 describes ray marching visually. At each iteration, a new safe distance s (depicted as green circle radii) along the ray is calculated. The ray is traversed using s as step size (to achieve the points depicted in green) until the safe distance falls below the absolute tolerance ϵ. The resulting point, depicted in purple, is close to the collision point. The implementation of ray marching algorithm given a point inversion algorithm proves to be relatively simple.

Algorithm 6 Ray Marching

```
1: procedure INTERSECTCURVERAY($\gamma, A, B$)
2:     $B \leftarrow \frac{B}{||B||}, s \leftarrow 0, P \leftarrow A$
3:     for $k = 1, 2, ..., N$ do
4:         $s \leftarrow s + D(\gamma(\text{pointInversion}(\gamma, P)), P)$
5:         $P \leftarrow A + sB$
6:         if $s < \epsilon$ then
7:             return $\{P\}$
8:     return $\emptyset$
```

2.3.3 Curve-Curve Intersection in 2D

Let $\gamma_1(s)$ and $\gamma_2(t)$ be two continuous curves in $\mathbb{R}^2$ with $s, t \in [0, 1]$ which instersect at finitely many points at most. In this section, we seek for the points at which the curves γ_1 and γ_2 intersect. We will employ a pragmatic scheme that uses a piecewise linear approximation of the curve, on which finding intersections is comparably simple and only involves finding the intersection of line segments.

Definition 2.3.6. The intersection parameter set of two continuous curves γ_1 and γ_2 is defined as

$$I(\gamma_1, \gamma_2) := \{(s, t) \in [0, 1]^2 : \gamma_1(s) = \gamma_2(t)\}.$$

Our goal is to find the set $I(\gamma_1, \gamma_2)$ in case of $|I(\gamma_1, \gamma_2)| < \infty$. We only consider this case, because none other are of interest in the applications of CoolingGen.

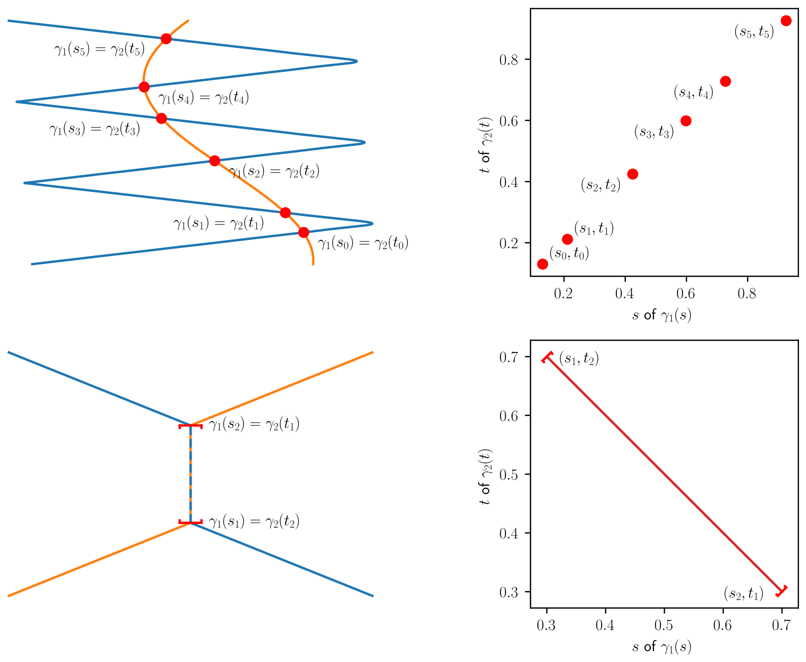


Figure 2.5: In the top row, we have $|I(\gamma_1, \gamma_2)| < \infty$, which is the case considered in this thesis. In the bottom row, $|I(\gamma_1, \gamma_2)| = \infty$. In the left column we can see the intersection points, whereas the right column represent the set $I(\gamma_1, \gamma_2)$.

Definition 2.3.7. The line segment between the points A_s and A_e is defined as the set

$$\overline{A_sA_e} := \{(1-t)A_s + tA_e : t \in [0,1]\}.$$

The dependence of a point on the line segment on the real parameter t motivates the description of this set as a map

$$L_A : \mathbb{R} \to \mathbb{R}^d, \quad t \mapsto (1-t)A_s + tA_e,$$

which yields the equation $\overline{A_sA_e} = L_A([0,1])$. Notice that given the definition of the direction vector $A_d := A_e - A_s$, we can write

$$L_A(t) = A_s + tA_d$$

The key idea of our algorithm is described by the following theorem.

Theorem 2.3.8. Let γ_1 and γ_2 be two continuous curves and let $(s^*, t^*) \in I(\gamma_1, \gamma_2)$ and $s^*, t^* \neq 0$. If the curve derivatives yield $\nabla\gamma_1(s^*) \neq \nabla\gamma_2(t^*)$, then there exist neighborhoods

$$[s^* - \epsilon, s^* + \epsilon] \quad \text{and} \quad [t^* - \epsilon, t^* + \epsilon]$$

around s^* and t^*, respectively, where $\epsilon > 0$. Then the line segments

$$\overline{\gamma_1(s^* - \epsilon)\gamma_1(s^* + \epsilon)} \quad \text{and} \quad \overline{\gamma_2(t^* - \epsilon)\gamma_2(t^* + \epsilon)}$$

intersect.

Remark. Suppose that the line segments $\overline{\gamma_1(s_1)\gamma_1(s_2)}$ and $\overline{\gamma_2(t_1)\gamma_2(t_2)}$ intersect. We cannot conclude that there is an intersection $(s,t) \in I(\gamma_1, \gamma_2)$ with $s \in [s_1, s_2]$ and $t \in [t_1, t_2]$. Using this theorem as core concept for our algorithm, we need to ensure that the distance between the curves is small inside of $[s_1, s_2] \times [t_1, t_2]$. If this is not the case, then no intersection lies inside of $[s_1, s_2] \times [t_1, t_2]$ and every line segment intersection found within should be discarded.

If for any intersection parameter pair $(s^*, t^*) \in I(\gamma_1, \gamma_2)$ we have $\nabla\gamma_1(s^*) = \nabla\gamma_2(t^*)$, then it is very likely that the intersection is not detected by the described line segment intersection scheme.

Theorem 2.3.9. Let $L_A([0,1])$ and $L_B([0,1])$ be two line segments in $\mathbb{R}^2$. If the lines $L_A((-\infty, \infty))$ and $L_B((-\infty, \infty))$ are not parallel, their single intersection pointt is given by $L_A(t_A) = L_B(t_B)$, where

$$t_A = \frac{\det\begin{pmatrix} D_y & D_x \\ B_{d_y} & B_{d_x} \end{pmatrix}}{\det\begin{pmatrix} B_{d_x} & B_{d_y} \\ A_{d_x} & A_{d_y} \end{pmatrix}} \quad \text{and} \quad t_B = \frac{\det\begin{pmatrix} D_y & D_x \\ A_{d_y} & A_{d_x} \end{pmatrix}}{\det\begin{pmatrix} B_{d_x} & B_{d_y} \\ A_{d_x} & A_{d_y} \end{pmatrix}}, \tag{2.10}$$

where $D = B_s - A_s$ and $t_A, t_B \in \mathbb{R}$. If $t_A \in [0,1] \wedge t_B \in [0,1]$, the line segments intersect in a single point.

In our algorithm we sample the curves γ_1 and γ_2 at $t_i = \frac{i}{N}$, where $i \in \{0, ..., N\}$. In the next step we check for an intersection of

$$\overline{\gamma_1(t_i)\gamma(t_{i+1})} \quad \text{and} \quad \overline{\gamma_2(t_j)\gamma_2(t_{j+1})}$$

for $i, j \in \{0, ..., N-1\}$. For all intersection pairs (i^*, j^*), we calculate the interval midpoint distance

$$D\left(\gamma_1\left(\frac{t_{i^*}+t_{i^*+1}}{2}\right), \gamma_2\left(\frac{t_{j^*}+t_{j^*+1}}{2}\right)\right)$$

and recursively repeat this procedure on the curves $\gamma_1([t_i, t_{i+1}])$ and $\gamma_2([t_j, t_{j+1}])$. If after a given recursion depth, the interval midpoint distance of some pair (i^*, j^*) is greater than a given absolute tolerance α, we discard (i^*, j^*).

The sampling into a piecewise linear interpolation of any curve will be handled by the following algorithm.

Algorithm 7 Calculating the piecewise linear interpolation

```
1: Input
2:     $\gamma$                      curve on $[0, 1]$
3:     $l, u$                    lower and upper bound inside of $[0, 1]$
4:     $N$                       number of line segments that approximate the curve
5: procedure PIECEWISELININTERP($\gamma, l, u, N$)
6:     $P \leftarrow$ empty vector with $N + 1$ entries
7:     for $i = 0, ..., N$ do
8:         $P_i \leftarrow \gamma\left((u-l)\frac{i}{N} + l\right)$
9:     return $P$
```

Calculating whether two line segments intersect is a constant time operation. For each recursion depth, this operation is executed N^2 times, granting the total performance of $O(N^2)$.

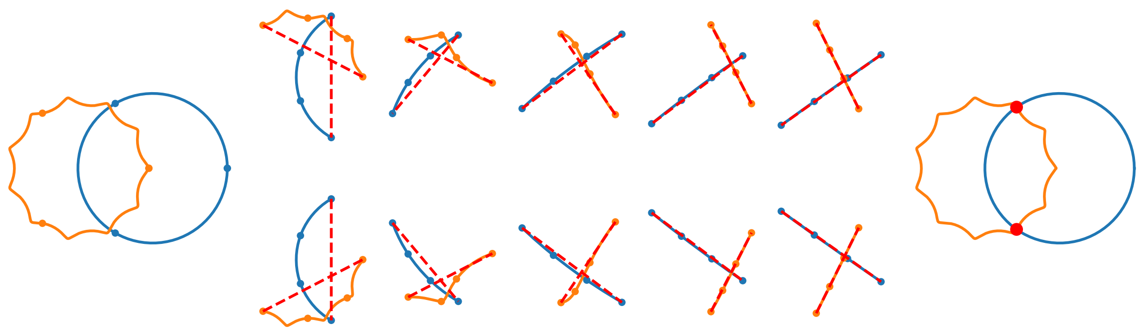


Figure 2.6: We find the intersection of the curves on the left. The two middle rows represent the different steps of the algorithm, where the curves are divided curve segments. If an intersection between end points of these segments occur, that segment is divided again into new segments until a points (s, t) is found such that $D(\gamma_1(s), \gamma_2(t)) \leq \epsilon$.

The algorithm in Figure 2.6 first calculates estimates of the intersection points and then refines those estimates until the target distance falls below the thresold ϵ, in which case it returns the point. If a maximum recursion depth $d_{\max}$ is reached before falling below target distance, no point is returned instead.

Algorithm 8 Curve Intersection

```
1: Input
2:     $\gamma_1, \gamma_2$                 curves on $[0,1]$
3:     $d_{\max}$                          maximum recursion depth
4:     $\epsilon$                          absolute tolerance value
5: Output
6:     $I(\gamma_1, \gamma_2)$             intersection parameter set
7: procedure INTERSECTCURVESRECURSION($I, \gamma_1, l_1, u_1, \gamma_2, l_2, u_2, d$)
8:     $m_1 \leftarrow \frac{l_1+u_1}{2}$ and $m_2 \leftarrow \frac{l_2+u_2}{2}$
9:     if $D(\gamma_1(m_1), \gamma_2(m_2)) \leq \epsilon$ then append $(m_1, m_2)$ to $I$ and return
10:    if $d \geq d_{\max}$ then do nothing and return
11:    $P^{(1)} \leftarrow$ piecewiseLinInterp$(\gamma_1, l_1, u_1, N)$ and $P^{(2)} \leftarrow$ piecewiseLinInterp$(\gamma_2, l_2, u_2, N)$
12:    for $i = 0, ..., N-1$ do
13:        for $j = 0, ..., N-1$ do
14:            if lineSegmentsIntersect$(P^{(1)}_i, P^{(1)}_{i+1}, P^{(2)}_j, P^{(2)}_{j+1})$ then
15:                $s_1 \leftarrow (l_1 - u_1)\frac{i}{N} + l_1$ and $s_2 \leftarrow (l_1 - u_1)\frac{i+1}{N} + l_1$
16:                $t_1 \leftarrow (l_2 - u_2)\frac{j}{N} + l_2$ and $t_2 \leftarrow (l_2 - u_2)\frac{j+1}{N} + l_2$
17:                intersectCurvesRecursion$(I, \gamma_1, s_1, s_2, \gamma_2, t_1, t_2, d+1)$
18:    return $I$
19: procedure INTERSECTCURVES($\gamma_1, \gamma_2$)
20:    $I \leftarrow \emptyset$
21:    intersectCurvesRecursion$(I, \gamma_1, 0, 1, \gamma_2, 0, 1, 0)$
22:    return $I$
```

As discussed before, this intersection algorithm will not identify every kind of intersection, however, the algorithm proves to be fast, robust and sufficient for CoolingGen.

2.3.4 Offset Curves in 2D

Offset curves are an important CAD tool and can be used to create fillets or simply to model parallel curves, i.e. for the creation of contours of a certain thickness.

Definition 2.3.10. Let $\gamma(t) = (x(t), y(t)) \in \mathbb{R}^2$ for $t \in [0,1]$ be a differentiable curve. Then $O^\gamma_d(t)$ is called the d-offset curve and is defined as

$$O^\gamma_d(t) := \gamma(t) + dN^\gamma(t),$$

where $N^\gamma(t)$ is called the normal of $\gamma(t)$ and defined as

$$N^\gamma(t) := \frac{\nabla\gamma(t)^\perp}{||\nabla\gamma(t)||}$$

and

$$\nabla\gamma(t)^{\perp} = \left(\frac{\mathrm{d}x}{\mathrm{d}t}(t), \frac{\mathrm{d}y}{\mathrm{d}t}(t)\right)^{\perp} = \left(-\frac{\mathrm{d}y}{\mathrm{d}t}(t), \frac{\mathrm{d}x}{\mathrm{d}t}(t)\right)$$

is orthogonal to the gradient of $\gamma(t)$.

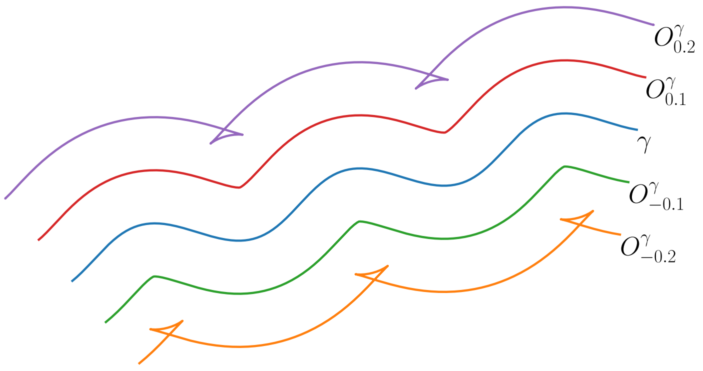


Figure 2.7: Offset curves for a given curve γ and for different values of d.

As can be observed in Figure 2.7, some O^{γ}_{d} tend to self-intersect even though γ does not self-intersect. We previously established that the non-existence of any self-intersection of a curve is equivalent to its injectivity. In CoolingGen, we are often interested in modelling parallel contours. However, self-intersections do not represent meaningful objects in this context. Specifying the offset value d, a user of CoolingGen would expect to find a curve $\overline{O}^{\gamma}_{d}$ that at least satisfies the property

$$\min_{s} D(\gamma(s), \overline{O}^{\gamma}_{d}) = d. \tag{2.11}$$

Informally, if we imagine a ball of radius d to roll along one side of the curve γ, then $O^{\gamma}_{d}(t)$ is injective as long as the ball only touches one point at a time.

Remark. Formally, we are able to define the *curvature* $\kappa^{\gamma}(t)$ of a curve at any given point, which is equal to the reciprocal of the radius of the tangential sphere of the curve at that point. This notion allows us to make the following observation: If there exists some t such that $\kappa^{\gamma}(t) > \frac{1}{d}$, then $O^{\gamma}_{d}(t)$ is not injective.

If we force injectivity of O^{γ}_{d} in a certain way, we will be able to guarantee the property in Equation 2.11. To do this, we need to find the self-intersections of O^{γ}_{d}.

Definition 2.3.11. The parameter set of self-intersections of a curve $\gamma(t)$ is defined as

$$I(\gamma) := \{(s,t) \in [0,1]^2 : \gamma(s) = \gamma(t) \ \wedge \ s < t\}.$$

To find these self-intersections, we can use a slightly modified version of Algorithm 8. In the first recursion step of intersectCurvesRecursion, instead of iterating j from 0 to $N-1$, we are only interested in j from $i+2$ to $N-1$, since for all $(s,t) \in I(O^{\gamma}_{d})$, we have $s < t$. Because $s < t$ each such parameter pair naturally represents an interval subset $[s,t]$ of the parameter space $[0,1]$.

Definition 2.3.12. A pair of self-intersections $(s_1, t_1), (s_2, t_2) \in I(\gamma)$ is called nested if $[s_1, t_1] \cap [s_2, t_2] \neq \emptyset$. We call $I(\gamma)$ untangled if $I(\gamma)$ does not contain a nested pair of self-intersections.

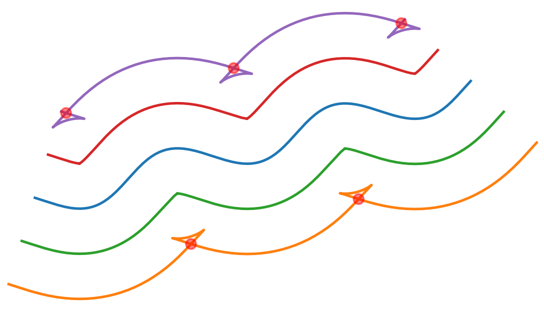
To meaningfully trim the curve, we need to ensure that the self-intersection parameter set is untangled. We can do this by iterating over the set of intersections $I(O_d^\gamma) = \{(s_i, t_i) : i \in \{1, ..., m\}\}$ returned by Algorithm 8 and checking for disjointness of pairs $[s_i, t_i]$ and $[s_j, t_j]$ for $i \neq j$, where $i, j \in \{1, ..., m\}$. In case these two sets are not disjoint, we can replace them with their union $[s_i, t_i] \cup [s_j, t_j]$. Using this procedure, we can calculate the untangled intersection set $\overline{I}(O_d^\gamma)$.

Definition 2.3.13. For an untangled self-intersection set $\overline{I}(\gamma)$, we call the set

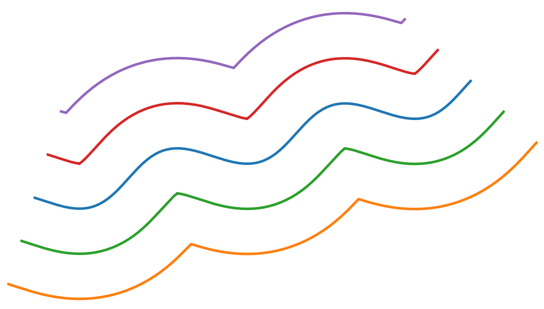
$$T(\gamma) := \bigcup_{(s,t) \in \overline{I}(\gamma)} [s, t]$$

the trim range of γ.

We can now choose to only further use and display the curve on $[0, 1] \setminus T(O_d^\gamma)$ or on $T(O_d^\gamma)$.



(a) Non-injective offset curves.



(b) Trimmed offset curves on $[0, 1] \setminus T(O_d^\gamma)$.

Figure 2.8: Trimming two offset curves at intervals of self intersections for a given curve γ and for different values of d.

2.3.5 Fillet Creation

Given two curves γ_1 and γ_2 defined on $t \in [0, 1]$ that intersect at their end points, we can construct a circle of a given radius that shares one tangential intersection point with each curve. A circle like this is called a fillet circle. More formally, we can make the following definitions

Definition 2.3.14. Let γ_1 and γ_2 be two differentiable curves. If curve $\phi_r^{\gamma_1,\gamma_2}$ satisfies

$$\phi_r^{\gamma_1,\gamma_2}([0, 1]) = \{(x, y) : (x - x_0)^2 + (y - y_0)^2 = r^2\}$$

for a center point (x_0, y_0) and a radius r, then it is a circle. If furthermore there exist intersection points

$$\gamma_1(s_1) = \phi_r^{\gamma_1,\gamma_2}(\sigma_1) \text{ and } \gamma_2(s_2) = \phi_r^{\gamma_1,\gamma_2}(\sigma_2)$$

for some $s_1, s_2, \sigma_1, \sigma_2 \in [0, 1]$ and these intersection points are tangential in the sense that

$$\nabla\gamma_1(s_1) = \nabla\phi_r^{\gamma_1,\gamma_2}(\sigma_1) \text{ and } \nabla\gamma_2(s_2) = \nabla\phi_r^{\gamma_1,\gamma_2}(\sigma_2),$$

then $\phi_r^{\gamma_1,\gamma_2}$ is a fillet circle of radius r of γ_1 and γ_2.

Remark. A fillet circle $\phi_r^{\gamma_1,\gamma_2}$ does not necessarily exist for two given curves γ_1, γ_2. If a fillet curve $\phi_{r^*}^{\gamma_1,\gamma_2}$ exists for some radius r^*, it does not exist for all radii $r \in \mathbb{R}$.

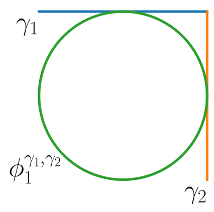


Figure 2.9: A fillet curve of radius 1 and the respective curves γ_1 and γ_2.

If it exists, we will find a procedure that yields a fillet circle for a given radius r and a pair of curves γ_1, γ_2. We only need to find the center point (x_0, y_0). Then we can use the parametric definition of a circle to write

$$\phi_r^{\gamma_1,\gamma_2}(t) = (r\cos(2\pi t) + x_0, r\sin(2\pi t) + y_0).$$

For example, as shown in Figure 2.9, given the curves $\gamma_1(t) = (2t-1, 1)$ and $\gamma_2 = (1, 2t-1)$ and a radius $r = 1$, the center point (x_0, y_0) of the circle is equal to $(0, 0)$.

We can use offset curve and intersection methods to find the center point (x_0, y_0) in the general case.

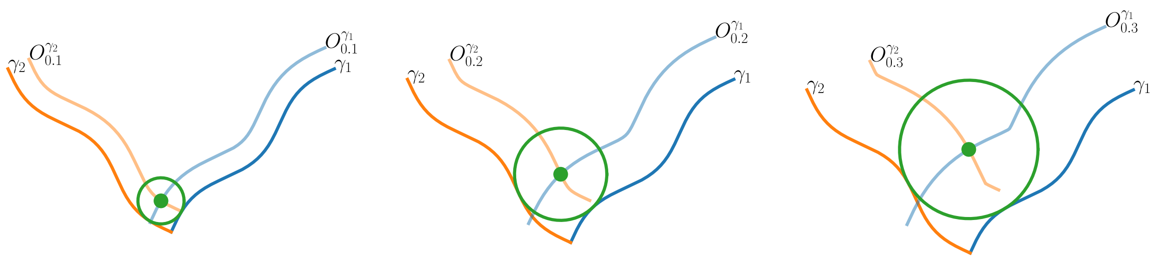


Figure 2.10: Construction of the fillet circle of different radii.

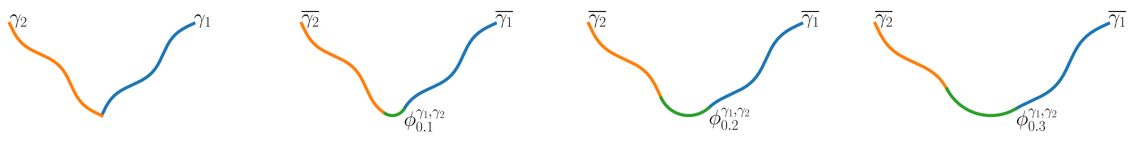


Figure 2.11: Trimmed curves and fillets

2.4 Jet Engine Design Specifics

2.4.1 Fundamental Terms

2.4.2 The S2M Net

3 Results

3.1 Cooling Geometries And Their Parametrizations

3.1.1 Chambers

3.1.2 Turnarounds

3.1.3 Slots

3.1.4 Film Cooling Holes

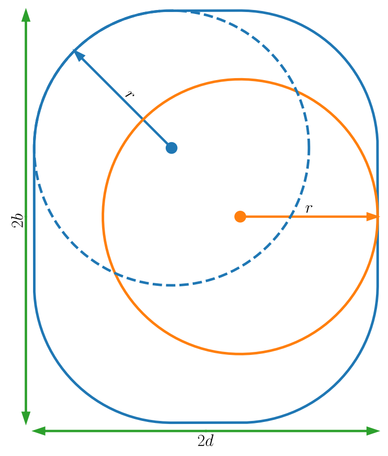


Figure 3.1: yeah

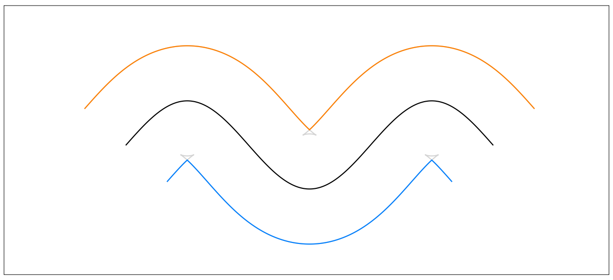


Figure 3.2: yeah

3.1.5 Impingement Inserts

3.2 Export for CENTAUR

3.3 Export for Open CASCADE

4 Discussion

4.1 Future Work

4.2 Conclusion

[Pie97]

5 References

[Béz68] Pierre E. Bézier. “How Renault Uses Numerical Control for Car Body Design and Tooling”. In: *SAE Technical Paper Series*. SAE International, Feb. 1968. DOI: 10.4271/680010.

[Pie97] Les A. Piegl. *The NURBS book*. Springer, 1997, p. 646. ISBN: 3540615458.