Master's thesis in
Applied Computer Science

CoolingGen

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

October 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.

Göttingen, October 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

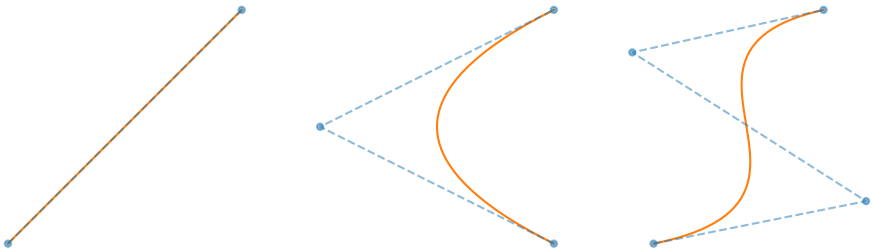
1.3 Problem Statement

2 Geometric 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 Bézier curves. We will then generalize Bézier 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, ray marching, curve intersection, offset curve creation and fillet creation, 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: Bézier 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 Equation (2.2) simplifies to

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

and Equation (2.1) simplifies to

$$B_P(t) := B_{P,[0,1]}(t) = \sum_{k=0}^{n} \binom{n}{k}(1-t)^{n-k}t^k P_k. \tag{2.3}$$

This case is the only case considered in this thesis.

In a Bézier curve of a given degree, the control points completely determine the shape of the curve. This behavior can be observed in Figure 2.1. However, changing one control point affects the whole curve, since for n control points the degree of the Bézier curve is always $n-1$. This behavior also becomes apparent using de Casteljau's algorithm for the computation of points on the Bézier curve.

2.1.2 De Casteljau's Algorithm

The computation of Equation (2.3) is usually performed using de Casteljau's algorithm. This is because the algorithm yields a simple implementation and lower complexity than straightforwardly computing Equation (2.3). The algorithm was proposed by Paul de Faget de Casteljau for the automobile manufacturer Citroën in the 1960s.

Algorithm 1 de Casteljau's algorithm

Input
  $P = \{P_0, P_1, ..., P_n\}$ set of control points
  t real number
Output
  $P_0^{(n)} = B_P(t)$ the point on the Bézier curve w.r.t. to t
procedure deCasteljau(P, t)
  $P^{(0)} \leftarrow P$
  for $j = 1, 2, ..., n$ **do**
    for $k = 0, 1, ..., n - j$ **do**
      $P_k^{(j)} \leftarrow (1-t) \cdot P_k^{(j-1)} + t \cdot P_{k+1}^{(j-1)}$
  return $P_0^{(n)}$

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

Proof. By induction. Let $n = 1$ and $t \in [0, 1]$ be fixed. Then

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

By employing the induction hypothesis

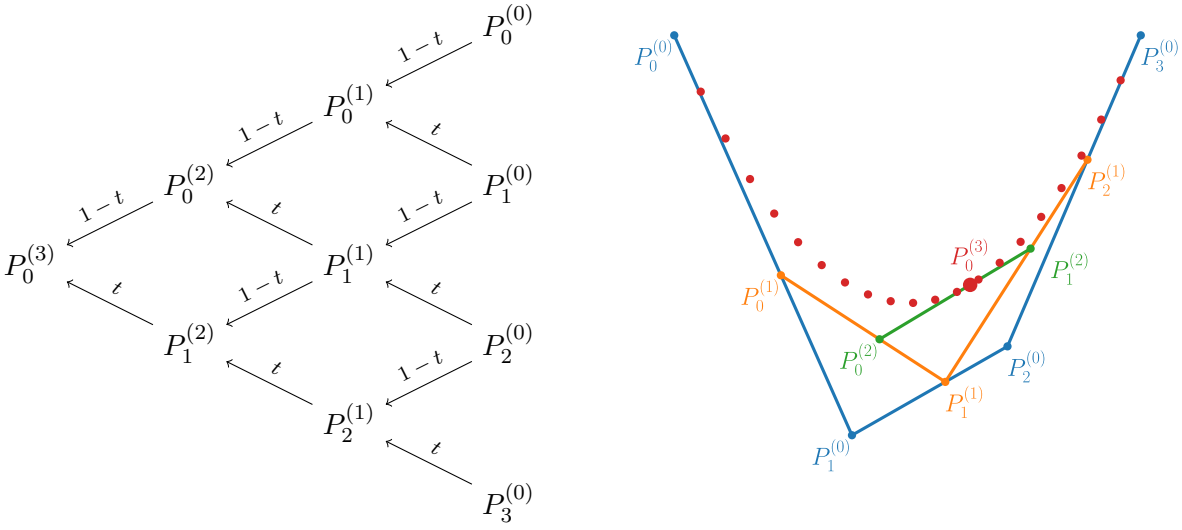
$$P_j^{(n)} = \sum_{k=j}^{n+j} \binom{n}{k} (1-t)^{n-k} t^k P_{j+k}$$

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 $n+1$. □

A visual representation of Algorithm 1 yields a triangular scheme. To compute one point on a Bézier 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 Bézier 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, Bézier curves have a lot of covenient properties. 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 the whole Bézier curve. The following theorems further illustrate why this is a good concept.

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

Proof. Explicit computation for $t = 0$ and $t = 1$ 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

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

This property is called invariance with respect to linear transforms. □

Theorem 2.1.6. The Bézier curve $B_P(t)$ 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. □

Theorem 2.1.4 guarantees the control over the end points of the the curve, whereas Theorem 2.1.6 ensures that the Bézier curve lies close to the control points. Theorem 2.1.5 shows that transformations such as rotations and projections can be applied directly to the control points instead of the points on the curve.

Simple as their appearance may be, Bézier 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 a low number of of control points, is simultaneously one of their greatest shortcoming. This is most likely the reason why Bézier curves are not the state of the art in technical engineering applications. However, Bézier 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 Bézier 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

$$N_{1,k}^{(t_m)}(t) := \begin{cases} 1 & \text{if } t \in [t_k, t_{k+1}), \\ 0 & \text{else.} \end{cases} \tag{2.4}$$

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

$$N_{p,k}^{(t_m)}(t) := \omega_{p-1,k}^{(t_m)}(t)\, N_{p-1,k}^{(t_m)}(t) + (1 - \omega_{p-1,k+1}^{(t_m)}(t))\, N_{p-1,k+1}^{(t_m)}(t), \tag{2.5}$$

where

$$\omega_{p,k}^{(t_m)}(t) := \begin{cases} \frac{t-t_k}{t_{k+p}-t_k} & \text{if } t_{k+p} \neq t, \\ 0 & \text{else.} \end{cases} \tag{2.6}$$

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

$$C_P(t) = \frac{\sum_{k=0}^{n} N_{p,k}(t) w_k P_k}{\sum_{k=0}^{n} N_{p,k}(t) w_k}. \tag{2.7}$$

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

$$C_P(t) = \Phi^\dagger(S_{\Phi_w(P)}(t)).$$

Utilizing the notation with the embedding map Φ_w and the projection map $\Phi^\dagger$, we will calculate points on an arbitrary NURBS curve by calculating points on a corresponding B-spline curve. 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. 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.

The calculation of points on a B-spline curve can be done similarly to the calculation of a point on a Bézier curve. The exact method is presented in the next section.

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

$$\Phi^\dagger(S_{\Phi_w(P)}(t)) = \Phi^\dagger(\Phi_w(S_P(t))) = S_P(t).$$

□

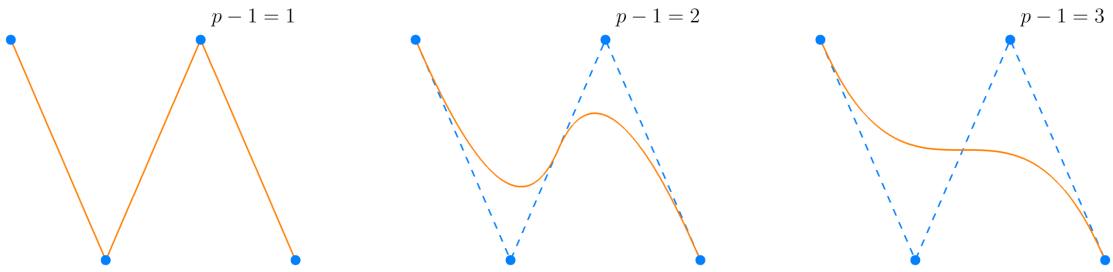


Figure 2.3: A set of control points and three NURBS curves of different degrees.

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.

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, which also yields the same calculation strategy.

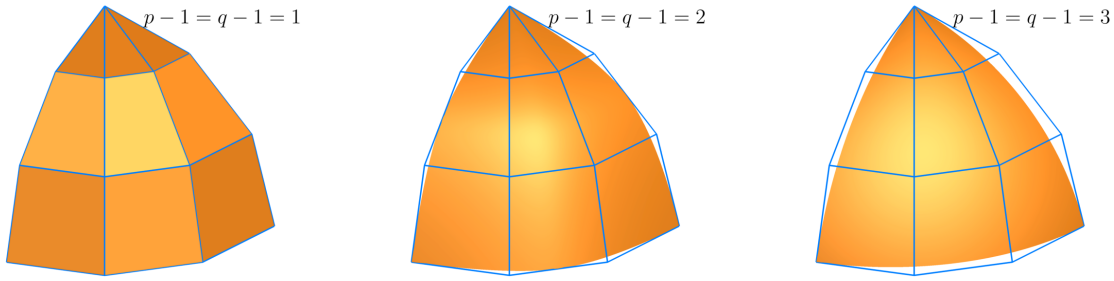


Figure 2.4: A set of control points and three NURBS surfaces of different degrees.

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 monotonicity 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 $P = \{P_0, P_1, ..., P_n\}$ be 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

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

Shifting the summation index in the second sum, we can summarize the two sums0 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

$$P_k^{(j)}(t) := \begin{cases} \omega_{p-j,k}\, P_k^{(j-1)}(t) \;+\; (1 - \omega_{p-j,k})\, P_{k-1}^{(j-1)}(t) & \text{if } j > 0, \\ P_k & \text{else,} \end{cases} \tag{2.8}$$

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 (2.8) as the key step of our algorithm to compute $S(t)$. It becomes apparent 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 Bézier curves. As zero-values for $\omega_{i,j}$ (or $1 - \omega_{i,j}$) can be completely omitted by counting how often the t occurs in the knot vector, we arrive at the following algorithm:

Algorithm 2 de Boor's algorithm for curves

Input

$P = \{P_0, P_1, ..., P_n\}$ set of control points of the B-spline curve

$\tau = (t_0, t_1, ..., t_{n+p})$ knot vector of the B-spline curve

$p - 1$ degree of the B-spline curve

$t \in [t_0, t_{n+p})$ real number

Output

$S_P(t)$ the point on the B-spline curve w.r.t. to t

procedure deBoorCurve(P, p, τ, t)

  Find l such that $t \in [t_l, t_{l+1})$

  Let m be the multiplicity of t in the knot vector τ

  $P^{(0)} \leftarrow P$

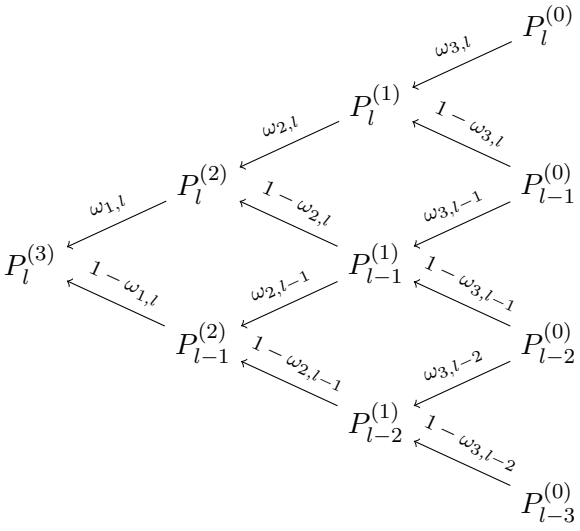
  for $j = 1, 2, ..., p - m - 1$ **do**

    for $k = l - p + j + 1, ..., l - m$ **do**

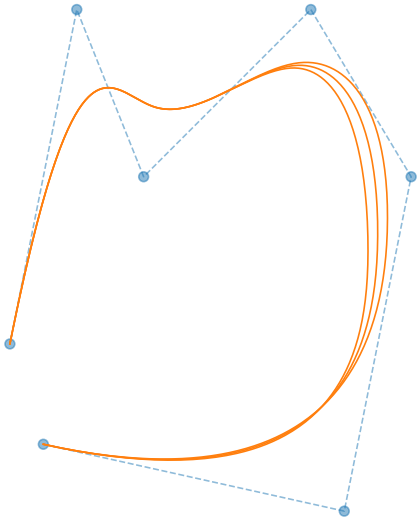
      $\omega_{p-j,k} \leftarrow \frac{t - t_k}{t_{k+p-j} - t_k}$

      $P_k^{(j)} \leftarrow (1 - \omega_{p-j,k}) \cdot P_{k-1}^{(j-1)} + \omega_{p-j,k} \cdot P_k^{(j-1)}$

  return $P_{l-m}^{(p-m-1)} = S_P(t)$



(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 2.5: 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 Bézier 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

Input
- $P = \{P_{i,j}\}$ — set of control points with $n_1 \cdot n_2$ elements
- $\tau_u = (u_0, u_1, ..., u_{n_u+p})$ — first knot vector of the B-spline curve
- $\tau_v = (v_0, v_1, ..., v_{n_v+q})$ — second knot vector of the B-spline curve
- $p-1$ — first degree of the B-spline curve
- $q-1$ — second degree of the B-spline curve
- $u \in [u_0, u_{n_u+p})$ — first real number
- $v \in [v_0, v_{n_v+q})$ — second real number

Output
- $S_P(u,v)$ — the point on the B-spline curve w.r.t. to u, v

procedure deBoorSurface($P, p, q, \tau_u, \tau_v, u, v$)
  Find l such that $u \in [u_l, u_{l+1})$
  Let m be the multiplicity of u in the knot vector τ_u
  for $k = l-p+1, , ..., l-m-1$ **do**
    $Q_k \leftarrow$ deBoorCurve($P_{k,\cdot}, q, \tau_v, v$)
  return deBoorCurve(Q, p, τ_u, u)

Similarly to the Remark under Definition 2.2.4, we can use Algorithm 3 to calculate points on NURBS surfaces by employing the projection map Φ_w and the projection map $\Phi^\dagger$.

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 Bézier curves.

Theorem 2.2.8. Let S_P be a B-spline curve and let C_P be 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

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

for B-spline curves and

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

for NURBS curves. Similarly, we can compute

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

for B-spline surfaces and

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

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 (2.8), 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.

For a given degree p, B-spline and NURBS curves can have any number $n > p + 1$ of control points. This makes them a much more flexible tool than their Bézier counterpart. Yet Bézier, B-splines and NURBS curves still share a lot of nice properties, which makes them, once implemented, easy to use.

2.3 Common Methods on Curves and Surfaces

In this section, the most important tools of CoolingGen are presented. The modification of NURBS objects through their control points may be simple, but common and useful operations that are imperative for CAD are, unfortunately, not as simple. Such operations include the calculation of intersection points of NURBS objects, the projection of an arbitrary point in space onto a NURBS object and the construction of so called offset curves and fillets.

This comes down to the fact that many of the problem statements are (generally) non-convex minimization problems. However, these problems can often be greatly simplified by asserting properties of underlying geometric patterns that arise in specific procedures.

2.3.1 Point Projection

Let $\gamma(t) \subset \mathbb{R}^d$ a parametric 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 $t^* \in T^*$. We call T^* the set of projection parameters for the point Q onto the curve γ.

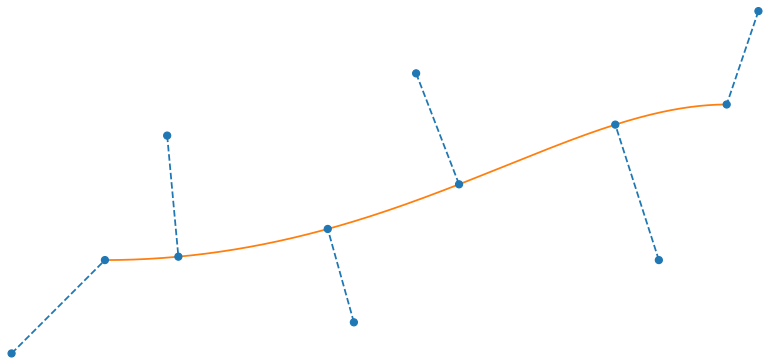


Figure 2.6: Point projection. Each blue point on the orange curve is the projection of the point connected to it with a dotted line.

Definition 2.3.1. The distance between a parametric curve $\gamma(t) \subset \mathbb{R}^d$ with $t \in [0,1]$ and a point $Q \in \mathbb{R}^d$ is given by

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

Remark. If $D(Q,\gamma) = 0$, then by the point-separating property of the Euclidean norm, Q lies on the curve γ. This special case, in which the point Q lies on the curve $\gamma(t)$, will be handled later. In general, the projection of a point is not necessarily unique. If we consider, for example, the curve $\gamma(t) = (\sin 2\pi t, \cos 2\pi t)$ with $t \in [0,1]$ and $Q = (0,0)$, the set of projection parameters T^* is equal to the whole domain of definition $[0,1]$.

Minimizing distance functions for general curves is 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 two times differentiable, then this problem can be solved iteratively by Newton's method. For an initial guess $t^{(0)}$ close to a target parameter $t^* \in T^*$ we use 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 central difference quotient

$$\phi'(t) \approx \frac{\phi(t+\epsilon) - \phi(t-\epsilon)}{2\epsilon} \tag{2.9}$$

for $t \in [\epsilon, 1-\epsilon]$. The expression $\phi''(t)$ is the second derivative of $\phi(t)$ with respect to t, which can be approximated by the difference quotient

$$\phi''(t) \approx \frac{\phi(t+\epsilon) - 2\phi(t) + \phi(t-\epsilon)}{\epsilon^2} \tag{2.10}$$

for $t \in [\epsilon, 1-\epsilon]$. For $t_0 \in [0, \epsilon]$, we assign $\phi'(t_0) \approx \phi'(\epsilon)$ and $\phi''(t_0) \approx \phi''(\epsilon)$ and for $t_1[1-\epsilon, 1]$ we assign $\phi'(t_1) \approx \phi'(1-\epsilon)$ and $\phi''(t_1) \approx \phi''(1-\epsilon)$.

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

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

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

We let $l = m - \frac{1}{n}$ and $u = m + \frac{1}{n}$. 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

Input
  N number of samples for initial value search
procedure FINDPOINTINVERSIONINITVALUES(γ, Q)
  $m \leftarrow 0$
  for $k = 1, ..., N$ **do**
    $t \leftarrow \frac{k}{N}$
    if $D(Q, \gamma(m)) > D(Q, \gamma(t))$ **then** $m \leftarrow t$
  return $m - \frac{1}{N}, m + \frac{1}{N}, m$

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

Input
  M — number of iterations
  $\epsilon > 0$ — differentiation step (small)
procedure POINTINVERSION(γ, Q)
  $l, u, m \leftarrow$ findPointInversionInitValues(γ, Q)
  $t \leftarrow m$
  for $k = 1, 2, ..., M$ **do**
    $t_+ \leftarrow t + \epsilon$ and $t_- \leftarrow t - \epsilon$
    $\phi \leftarrow D(\gamma(t), Q)$
    $\phi_+ \leftarrow D(\gamma(t_+), Q)$ and $\phi_- \leftarrow D(\gamma(t_-), Q)$
    $\phi' \leftarrow \frac{\phi_+ - \phi_-}{2\epsilon}$
    $\phi'' \leftarrow \frac{\phi_+ - 2\phi + \phi_-}{\epsilon^2}$
    $t \leftarrow t - \frac{\phi'}{\phi''}$
    if $t < l$ **then** $t \leftarrow l$
    if $t > u$ **then** $t \leftarrow u$
  return t

If the curve $\gamma(t)$ does not satisfy the aforementioned assumptions, then we might still be able to find t^* by applying the start value search scheme iteratively on the domain of $\gamma(t)$ to find smaller intervals $[l, u]$ if $\gamma(t)$ is at least continuous. Continuity of curves can safely be assumed all throughout CoolingGen.

We now treat the special case that the minimum target distance is equal to zero, in which case Q lies on the curve $\gamma(t)$.

Theorem 2.3.2. Let $\gamma(t) \subset \mathbb{R}^d$ be 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. □

If 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 the input parameter that fits the point Q. This justifies the name *point inversion* for this special case.

Using these principles, we can also project (or invert) a point $Q \in \mathbb{R}^d$ on a parametric surface $\beta(u, v)$ with $u, v \in [0, 1]$ by generalizing the presented methods. In this case, the distance function we want to minimize is

$$\phi(u, v) := D(\beta(u, v), Q)$$

We once again use the start value search. We let $u_i = \frac{i}{n_u}, v_j = \frac{j}{n_v}$ for $i \in \{0, ..., n_u\}, j \in \{0, ..., n_v\}$ and $n_u, n_v \in \mathbb{N}$. Then we seek

$$m_u, m_v = \underset{(u,v)\in\{u_0,...,u_n\}\times\{v_0,...,v_n\}}{\arg\min} \phi(u, v),$$

where $\phi(u, v) := D(\beta(u, v), Q)$. We use Newton's method only on the subset

$$M = [m_u - \frac{1}{n_u}, m_u + \frac{1}{n_u}] \times [m_v - \frac{1}{n_v}, m_v + \frac{1}{n_v}] \subset [0, 1]^2,$$

which in this case for each iteration yields a linear system of equations

$$H_\phi \left(u^{(k)}, v^{(k)}\right) \cdot \begin{pmatrix} u^{(k+1)} - u^{(k)} \\ v^{(k+1)} - v^{(k)} \end{pmatrix} = -\nabla\phi \left(u^{(k)}, v^{(k)}\right),$$

where $H_\phi(u, v)$ is the Hessian of ϕ and $\nabla\phi(u, v)$ is the gradient of ϕ. These differentials can once again be approximated by finite differences. If $H_\phi(u, v)$ and $\nabla\phi(u, v)$ do not exist for some (u, v) in the set M, then we can once again iteratively apply the start value search to find smaller subsets of M to calculate some (u^*, v^*) in the set of projection parameters for the point Q.

2.3.2 Intersection of a Ray and a Curve 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

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

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 $\overline{AP} \subset R_{A,B}([0,\infty))$ has at least ϵ distance to 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 $\emptyset$.

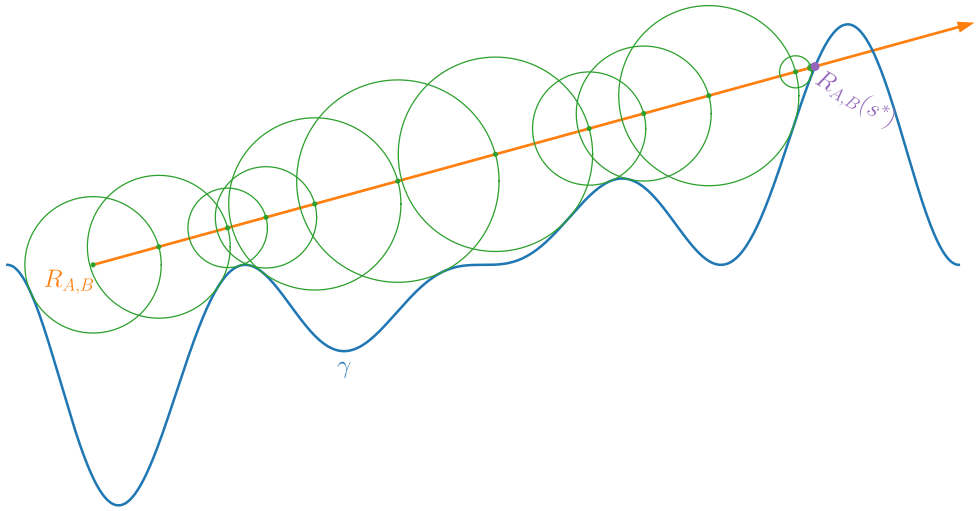


Figure 2.7: Ray marching.

Figure 2.7 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

procedure INTERSECTCURVERAY(γ, A, B)
  $B \leftarrow \frac{B}{||B||}, s \leftarrow 0, P \leftarrow A$
  for $k = 1, 2, ..., N$ **do**
    $s \leftarrow D(\gamma(\text{pointInversion}(\gamma, P)), P)$
    $P \leftarrow P + sB$
    if $s < \epsilon$ **then**
      return $\{P\}$
  return $\emptyset$

Using the point inversion algorithm for surfaces (instead of the point inversion algorithm for curves) to find the distance between a point and a surface, we can generalize this algorithm to find the intersection of a ray and a surface.

2.3.3 Intersecting Two Curves in 2D

In order to trim curves at intersection points, we require the pair of parameters at which two curves intersect. Let $\gamma_1(s)$ and $\gamma_2(t)$ be two continuous curves in $\mathbb{R}^2$ with $s, t \in [0, 1]$ which intersect at finitely many points at most. In this section, we seek for the set of parameters at which these curves intersect.

Definition 2.3.6. The distance between two curves γ_1, γ_2 is defined as

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

Similarly to the prior sections, our goal is to minimize a distance function $D(\gamma_1(s), \gamma_2(t))$. In this case, we will employ a pragmatic scheme that uses a piecewise linear approximation of the curve, in which finding intersections is comparably simple and involves finding the intersection of a number of line segments.

Definition 2.3.7. The intersection parameter set of two continuous curves γ_1 and γ_2 is defined as

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

Notice that $\gamma_1([0,1]) \cap \gamma_2([0,1]) = \emptyset$ if and only if $I(\gamma_1, \gamma_2) = \emptyset$.

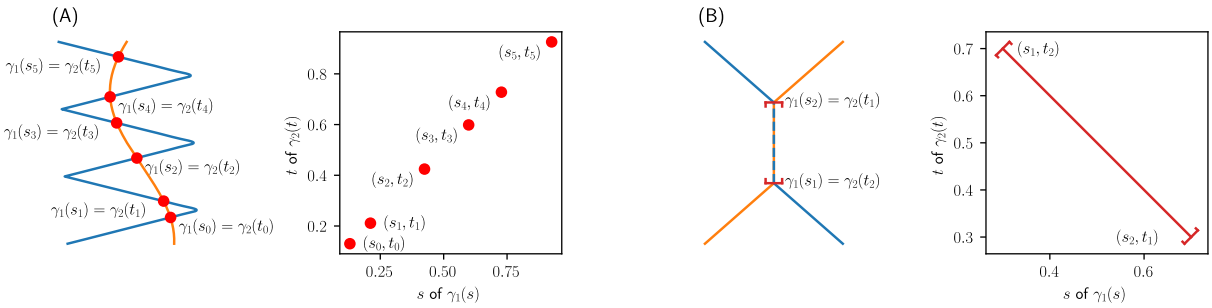


Figure 2.8: In (A) we have $|I(\gamma_1, \gamma_2)| < \infty$, whereas in (B) we have $|I(\gamma_1, \gamma_2)| = \infty$.

For the use with CoolingGen, our goal is to find the set $I(\gamma_1, \gamma_2)$ in case of $|I(\gamma_1, \gamma_2)| < \infty$ (see Figure 2.8).

Definition 2.3.8. The line segment between the points $A_s = (A_{s_x}, A_{s_y})^T$ and $A_e = (A_{e_x}, A_{e_y})^T$ is defined as the set

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

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

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

We write the derivative of a curve $\gamma(t)$ as $\nabla\gamma(t) := \gamma'(t)$. The key idea of our algorithm is described by the following theorem.

Theorem 2.3.9. Let γ_1 and γ_2 be two differentiable curves and let $(s^*, t^*) \in I(\gamma_1, \gamma_2) \cap (0,1)^2$. If the curve derivatives $\nabla\gamma_1(s^*)$ and $\nabla\gamma_2(t^*)$ are not parallel, then there exist neighborhoods

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

around s^* and t^*, respectively, where $\epsilon > 0$ such that 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 will need to ensure that the distance between the target curve segments is zero. In this numerical setting, it is sufficient to instead ensure

$$D(\gamma_1|_{[s_1,s_2]}, \gamma_2|_{[t_1,t_2]}) \leq \alpha$$

for a sufficiently small absolute tolerance $\alpha > 0$. 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. Here, the notation $\gamma|M$ for some parametric curve γ and a subset $M \in [0, 1]$ is the domain restriction of γ onto $[a, b]$.

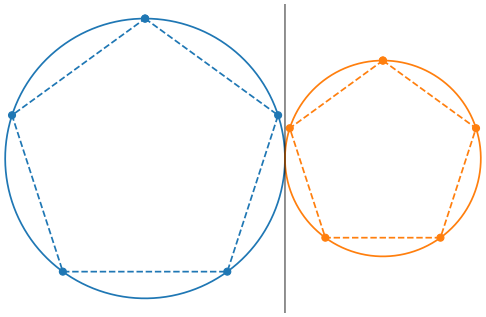


Figure 2.9: Two curves that intersect in a point tangentially and their linear approximations.

What Theorem 2.3.9 and it's Remark tell us is that parallel intersection points (for example in two circular curves that intersect in exactly one point, see Figure 2.9) are rarely detected by intersecting a piecewise linear approximation of the curve. In other words, 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 likely that the intersection is not detected by the described line segment intersection scheme.

Theorem 2.3.10. 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

$$t_A = \frac{\det \begin{pmatrix} \Delta S_y & \Delta S_x \\ B_{d_y} & B_{d_x} \end{pmatrix}}{\det \begin{pmatrix} B_{d_x} & B_{d_y} \\ A_{d_x} & A_{d_y} \end{pmatrix}} \quad \text{and} \quad t_B = \frac{\det \begin{pmatrix} \Delta S_y & \Delta S_x \\ A_{d_y} & A_{d_x} \end{pmatrix}}{\det \begin{pmatrix} B_{d_x} & B_{d_y} \\ A_{d_x} & A_{d_y} \end{pmatrix}},$$

where $\Delta S = B_s - A_s$ 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

Input
- γ — curve on $[0, 1]$
- l, u — lower and upper bound inside of $[0, 1]$
- N — number of line segments that approximate the curve

procedure PIECEWISELININTERP(γ, l, u, N)
- $P \leftarrow$ empty vector with $N + 1$ entries
- **for** $i = 0, ..., N$ **do**
  - $P_i \leftarrow \gamma\left((u - l)\frac{i}{N} + l\right)$
- **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 $O(N^2)$.

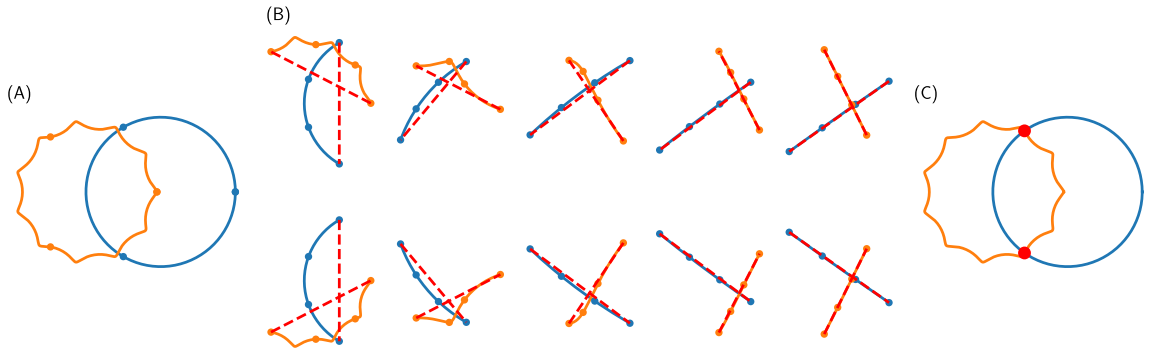


Figure 2.10: Demonstration of the curve intersection algorithm. Figure (A) shows the input curves, Figure (B) the different steps and Figure (C) the result. The dots on the curve represent the samples on the curve. The red lines represent the intersection of two line segments.

The algorithm in Figure 2.10 first calculates estimates of the intersection points and then refines those estimates until the target distance falls below α, 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. Using Algorithm 7 and writing the procedure from Theorem 2.3.10 as function `lineSegmentsIntersect`, which returns `true` if and only if the input line segments intersect, we can write the curve intersection algorithm as follows.

Algorithm 8 Curve Intersection

Input
  γ_1, γ_2 — curves on $[0,1]$
  $d_{\max}$ — maximum recursion depth
  α — absolute tolerance value
Output
  $I(\gamma_1, \gamma_2)$ — intersection parameter set
procedure INTERSECTCURVESRECURSION($I, \gamma_1, l_1, u_1, \gamma_2, l_2, u_2, d$)
  $m_1 \leftarrow \frac{l_1+u_1}{2}$ and $m_2 \leftarrow \frac{l_2+u_2}{2}$
  if $D(\gamma_1(m_1), \gamma_2(m_2)) \leq \alpha$ **then** append (m_1, m_2) to I and **return**
  if $d \geq d_{\max}$ **then** do nothing and **return**
  $P^{(1)} \leftarrow$ piecewiseLinInterp(γ_1, l_1, u_1, N) and $P^{(2)} \leftarrow$ piecewiseLinInterp(γ_2, l_2, u_2, N)
  for $i = 0, ..., N-1$ **do**
    for $j = 0, ..., N-1$ **do**
      if lineSegmentsIntersect$(P_i^{(1)}, P_{i+1}^{(1)}, P_j^{(2)}, P_{j+1}^{(2)})$ **then**
        $s_1 \leftarrow (l_1 - u_1)\frac{i}{N} + l_1$ and $s_2 \leftarrow (l_1 - u_1)\frac{i+1}{N} + l_1$
        $t_1 \leftarrow (l_2 - u_2)\frac{j}{N} + l_2$ and $t_2 \leftarrow (l_2 - u_2)\frac{j+1}{N} + l_2$
        intersectCurvesRecursion$(I, \gamma_1, s_1, s_2, \gamma_2, t_1, t_2, d+1)$
  return I
procedure INTERSECTCURVES(γ_1, γ_2)
  $I \leftarrow \emptyset$
  intersectCurvesRecursion$(I, \gamma_1, 0, 1, \gamma_2, 0, 1, 0)$
  return I

Using slight modifications, one can also make use of this algorithm in self-intersection detection. The details of self-intersection are worked out in the next section.

As discussed before, this intersection algorithm might not identify an intersection parameter pair (s,t) where $\nabla\gamma_1(s) = \nabla\gamma_2(t)$. It does however prove fast and robust in the case of CoolingGen, where such intersections are currently not of interest.

2.3.4 Offset Curves and Self-Intersections in 2D

Offset curves are an important CAD tool and can be used to model contours of a certain thickness, which is why they are often referred to as parallel curves. Offset curves are paramount in the creation of fillets, which we will elaborate later. Among the variety of methods in CoolingGen, the creation of offset curves proves to be the most useful one, which perhaps lies in the very nature of cooling geometries, most of which are comprised of offset surfaces inside a given turbine blade surface.

Definition 2.3.11. Let $\gamma(t) = (x(t), y(t)) \in \mathbb{R}^2$ for $t \in [0,1]$ be a differentiable curve. Then $O_d^\gamma(t)$ is called the d-offset curve and is defined as

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

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 $\gamma(t)$.

Remark. Notice that this definition makes the convention of an oriented normal vector, because it will be useful later. The unit vector $-N^\gamma(t)$ is also orthogonal to $\nabla\gamma(t)$.

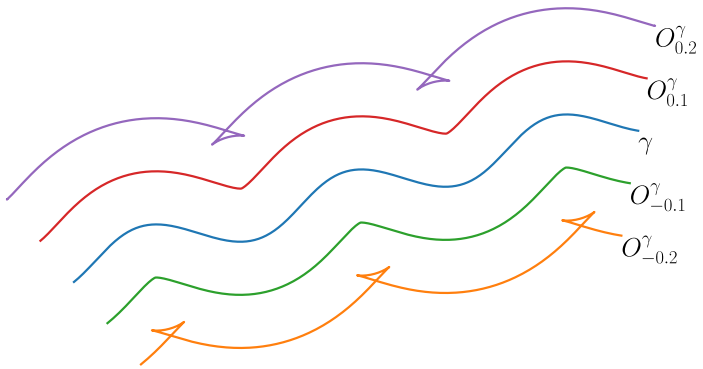


Figure 2.11: Offset curves for a given curve γ and for different values of d.

As can be observed in Figure 2.11, some O_d^γ 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 $\overline{O}_d^\gamma$ that at least satisfies the property

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

Informally, if we imagine a ball of radius d to roll along one side of the curve γ, then $O_d^\gamma(t)$ 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 $O_d^\gamma(t)$ is not injective.

If we force injectivity of O_d^γ 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_d^γ.

Definition 2.3.12. 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 $(s,t) \in I(O_d^\gamma)$, 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 2.3.13. For a self-intersection set $I(\gamma)$, we call the set

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

the trim range of γ.

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

(a) Non-injective offset curves. (b) Trimmed offset curves on $\overline{[0,1] \setminus T(O_d^\gamma)}$.

Figure 2.12: Trimming two offset curves at intervals of self intersections for a given curve γ and for different values of d.

Now, the restriction $O_d^\gamma|_{T(O_d^\gamma)}$ of the curve O_d^γ to the trim range $T(O_d^\gamma)$ is not necessarily defined for all $t \in [0,1]$. Similarly, the restriction $O_d^\gamma|_{\overline{[0,1]\setminus T(O_d^\gamma)}}$ of the curve O_d^γ to the trim range complement $\overline{[0,1] \setminus T(O_d^\gamma)}$ is not necessarily defined for all $t \in [0,1]$. However, we can define a domain scaling map ξ^M, that maps a union $M \subset [0,1]$ of finite intervals onto $[0,1]$, where M will be equal to the trim range or its complement. In other words, we want to define a map that allows us to write

$$O_d^\gamma(\xi^M([0,1])) = O_d^\gamma(M)$$

for any finite union M of closed intervals. We write the connected components of M as $[s_i, t_i]$ for $i \in \{1, ..., m\}$. By setting

$$\tilde{s}_1 := 0 \quad \text{and} \quad \tilde{t}_m := 1 \quad \text{and} \quad \tilde{s}_{j+1} := \tilde{t}_j := \frac{s_{j+1} + t_j}{2}$$

for $j \in \{1, ..., m-1\}$, we can define the maps

$$\xi_i^M(t) = \begin{cases} \frac{t_i - s_i}{\tilde{t}_i - \tilde{s}_i}\, t & \text{if } t \in [\tilde{s}_i, \tilde{t}_i], \\ 0 & \text{else,} \end{cases}$$

that map $[\tilde{s}_i, \tilde{t}_i]$ onto $[s_i, t_i]$ and any other value onto 0. Notice that

$$\bigcup_{i=1}^{m} [\tilde{s}_i, \tilde{t}_i] = [0, 1].$$

This way, the domain scaling map

$$\xi^M(t) = \sum_{i=1}^{m} \xi_i^M(t)$$

maps $[0, 1]$ onto M. Since both $\overline{[0,1] \setminus T(O_d^\gamma)}$ and $T(O_d^\gamma)$ are unions of closed intervals, we can use this procedure to define the trimmed offset curve as

$$\overline{O}_d^\gamma(t) = O_d^\gamma \circ \xi^{\overline{[0,1]\setminus T(O_d^\gamma)}}(t)$$

or

$$\overline{O}_d^\gamma(t) = O_d^\gamma \circ \xi^{T(O_d^\gamma)}(t)$$

respectively. This curve now satisfies Equation (2.11).

As has been shown, the construction of meaningful offset curves is in no way trivial and requires a multitude of tools. The simple definition of an offset curve (Definition 2.3.11) is only seemingly adequate. Nonetheless, larger offsets, that would lead to self-intersection, are required by the industry. But since self-intersecting curves do no represent sensible objects, we sought for a far more convoluted process.

In fact, the creation of fillets also plays a role in the creation of offset curves, since the procedure presented leaves behind non-differentiable places on the trimmed offset curves, which are undesirable for CoolingGen's purposes.

2.3.5 The Creation of Fillets in 2D

Given two curves γ_1 and $\gamma_2 \in \mathbb{R}^2$ defined on $t \in [0, 1]$ we can sometimes 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. Segments of these circles can replace non-differentiable segments of a curve. More formally, we can make the following definition.

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

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

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

points

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

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

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

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 usually not exist for all radii $r \in \mathbb{R}$.

Figure 2.13: A fillet curve of radius 1 and the respective curves γ_1 and γ_2.

We will find a procedure that yields a fillet circle for a given radius r and a pair of curves γ_1, γ_2 if it exists and returns nothing otherwise. We only need to find the center point (x_0, y_0). Then we can use the parametric definition of a circle and 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 2.13, 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 fillet circle is equal to $(0,0)$. Therefore, in this case,

$$\phi_r^{\gamma_1,\gamma_2}(t) = (\cos(2\pi t), \sin(2\pi t)),$$

which is the unit circle.

Theorem 2.3.15. Let γ_1, γ_2 and r such that a fillet circle $\phi_r^{\gamma_1,\gamma_2}$ exists. Then the midpoint $M = (x_0, y_0)$ of a fillet circle is given by

$$M = O_{\pm r}^{\gamma_1}(s_1) = O_{\pm r}^{\gamma_2}(s_2)$$

for $(s_1, s_2) \in I(O_{\pm r}^{\gamma_1}, O_{\pm r}^{\gamma_2})$.

Proof. Let $\tilde{\phi}(t) = r(\sin 2\pi t, \cos 2\pi t) + O_r^{\gamma_1}(s_1)$. Then we can calculate

$$\begin{aligned}\tilde{\phi}(t) &= r(\sin 2\pi t, \cos 2\pi t) + \gamma(s_1) \pm rN^{\gamma}(s_1) \\ &= r(\sin 2\pi t, \cos 2\pi t) + \gamma(s_1) \pm r\frac{\nabla\gamma(s_1)^{\perp}}{||\nabla\gamma(s_1)||}.\end{aligned}$$

Notice that $(\sin 2\pi t, \cos 2\pi t)$ is equal to an arbitrary unit vector, so for some fixed $\sigma_1 \in [0,1]$ the equation

$$(\sin 2\pi\sigma_1, \cos 2\pi\sigma_1) = \frac{\nabla\gamma(s_1)^\perp}{||\nabla\gamma(s_1)||}$$

will be satisfied, since $\frac{\nabla\gamma(s_1)^\perp}{||\nabla\gamma(s_1)||}$ is a fixed unit vector. In the same manner, we can construct σ_2 for the equivalent curve definition $\tilde{\phi}(t) = r(\sin 2\pi t, \cos 2\pi t) + O_r^{\gamma_2}(s_2)$. Notice that σ_1 and σ_2 also satisfy

$$\tilde{\phi}(\sigma_1) = \gamma_1(s_1) \quad \text{and} \quad \tilde{\phi}(\sigma_2) = \gamma_2(s_2).$$

By calculating $\nabla\tilde{\phi}(t)$, we find

$$\nabla\tilde{\phi}(t) = 2\pi(\cos 2\pi t, -\sin 2\pi t),$$

which after normalization yields the equalities

$$\frac{\nabla\gamma_1(s_1)}{||\nabla\gamma_1(s_1)||} = \pm\frac{\nabla\tilde{\phi}_r^{\gamma_1,\gamma_2}(\sigma_1)}{||\nabla\tilde{\phi}_r^{\gamma_1,\gamma_2}(\sigma_1)||} \quad \text{and} \quad \frac{\nabla\gamma_2(s_2)}{||\nabla\gamma_2(s_2)||} = \pm\frac{\nabla\tilde{\phi}_r^{\gamma_1,\gamma_2}(\sigma_2)}{||\nabla\tilde{\phi}_r^{\gamma_1,\gamma_2}(\sigma_2)||}.$$

Therefore, $\tilde{\phi}$ is in fact a fillet circle. □

Theorem 2.3.15 tells us we can use offset curve and intersection methods to find the center point (x_0, y_0) in the general case using the following steps given a radius r and two curves γ_1 and γ_2.

1. Calculate the two offset curves $O_r^{\gamma_1}$ and $O_r^{\gamma_2}$.
2. Find their intersection parameter set $I(O_r^{\gamma_1}, O_r^{\gamma_2})$.
3. For $(s,t) \in I(O_r^{\gamma_1}, O_r^{\gamma_2})$, set $(x_0, y_0) := O_r^{\gamma_1}(s) = O_r^{\gamma_2}(t)$.

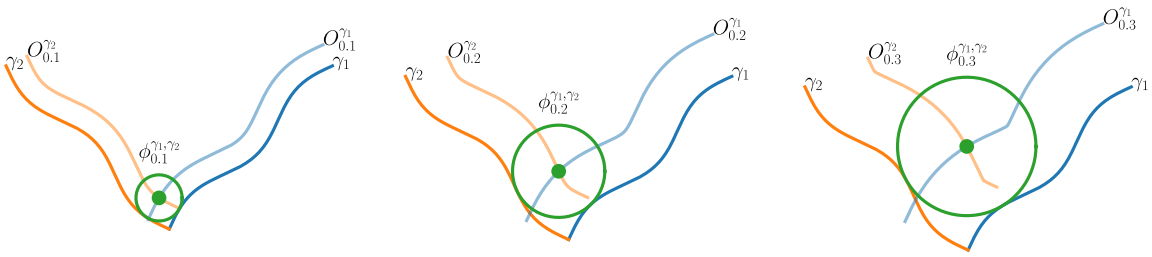


Figure 2.14: Construction of the fillet circle of different radii.

As can be seen in Figure 2.14, the curve $\phi_r^{\gamma_1,\gamma_2}$ constructed this way fulfills the requirements of Definition 2.3.14 and therefore is a fillet circle. Given the fillet circle, we now only need to trim the curves γ_1 and γ_2 and $\phi_r^{\gamma_1,\gamma_2}$.

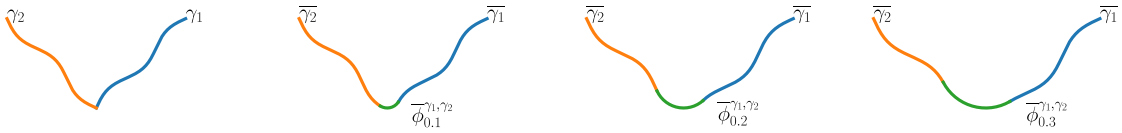


Figure 2.15: Trimmed curves and fillets

In CoolingGen, the curves which we want to connect by a fillet are always connected at their boundary points. Without loss of generality, we let $\gamma_1(0) = \gamma_2(0)$. The trimmed input curves are then given by

$$\overline{\gamma_1}(s) = \gamma_1|_{[s^*,1]}\left(\frac{s-s^*}{1-s^*}\right) \quad \text{and} \quad \overline{\gamma_2}(t) = \gamma_2|_{[t^*,1]}\left(\frac{t-t^*}{1-t^*}\right)$$

for $(s^*, t^*) \in I(O_r^{\gamma_1}, O_r^{\gamma_2})$. The parameters s and t are scaled such that the domains of the trimmed curves $\overline{\gamma_1}$ and $\overline{\gamma_2}$ remain equal to $[0, 1]$.

Now, instead of actually trimming $\phi_r^{\gamma_1,\gamma_2}$, we can construct a circular arc between the end points $D_i := \overline{\gamma_i}(0)$ of the trimmed curves for $i \in \{1, 2\}$. Let $M = (x_0, y_0)$ the fillet circle center. By sweeping the line segment $\overline{M_1 D_1}$ by a certain angle Θ, we can retrieve $\overline{\phi}_r^{\gamma_1,\gamma_2}$. This angle can be calculated by

$$\Theta = \arccos \frac{(D_1 - M) \cdot (D_2 - M)}{||D_1 - M||\ ||D_2 - M||},$$

where $(\cdot)$ is the standard dot product. We can now write the trimmed fillet curve as

$$\overline{\phi}_r^{\gamma_1,\gamma_2}(t) = R_{+t\Theta}(D_1 - M) + M = R_{-t\Theta}(D_2 - M) + M,$$

for $t \in [0, 1]$, where R_Θ is the rotation matrix given by

$$R_{t\Theta} := \begin{pmatrix} \cos t\Theta & -\sin t\Theta \\ \sin t\Theta & \cos t\Theta \end{pmatrix}. \tag{2.12}$$

We are now left with the two trimmed input curves $\overline{\gamma_1}$ and $\overline{\gamma_2}$ and the fillet curve $\overline{\phi}_r^{\gamma_1,\gamma_2}(t)$.

2.3.6 Intersecting a Surface and a Plane

In this section, we want to find an algorithm that intersects a plane and a surface in $\mathbb{R}^3$. The intersection genereally takes the form of a surface, a curve, a point or any union of the aforementioned. In the case of CoolingGen, we only care about the case that the intersection is a single curve. This assumption is implicitly made in many of the statements involving the intersection algorithm. First, we need to introduce some basic notions.

Definition 2.3.16. A plane P with support vector $S \in \mathbb{R}^3$ and normal vector $N \in \mathbb{R}^3$ with $||N|| = 1$ is given by the set

$$P_{S,N} := \{S + V : \mathbb{R}^3 \ni V \perp N\}.$$

It should be noted that for $\tilde{S} \in P_{\tilde{S},N}$ we have the identity $P_{\tilde{S},N} = P_{S,N}$. In other words, the choice of the support vector is arbitrary as long as it lies in the same plane.

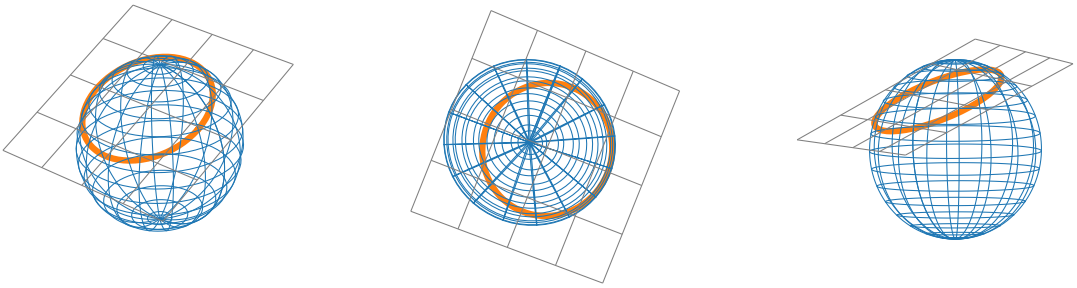


Figure 2.16: Intersecting a surface (blue) and a plane (grey) yields a curve (orange).

In linear Algebra, it is often the case that a plane is defined by a support vector and two direction vectors. In CAD however, it is instead commonplace to use the pair (S, N) of support vector and normal vector, which is always perpendicular to the two direction vectors of the plane. It should be noted that planes, which are two-dimensional affine subspace of $\mathbb{R}^d$, can only be described that way in the case $d = 3$. In our algorithm, the normal vector notation will prove very helpful.

Definition 2.3.17. The cross product of two vectors $A, B \in \mathbb{R}^3$ is given by

$$A \times B := ||A||\,||B||\,\sin(\theta)\,N,$$

where θ is the angle between A and B and $A, B \perp N$ with $||N|| = 1$. Here, $||\cdot||$ is the Euclidean norm of a vector. If A and B are parallel, then $A \times B = 0$

The cross product maps two vectors A, B onto another vector C. If A and B are not parallel to each other, then the three vectors A, B, C span $\mathbb{R}^3$. In other words, the cross product produces a vector that is perpendicular to the two input vectors.

Theorem 2.3.18. Let $S \in \mathbb{R}^3$ a support vector and $A, B \in \mathbb{R}^3$ two non-parallel direction vectors. Then the plane $\tilde{P} = \{S + tA + sB : t, s \in \mathbb{R}\}$ can be written as $P_{S,N}$ with

$$N := \frac{A \times B}{||A \times B||}.$$

Proof. Let $X \in \tilde{P}$. Then

$$X - S \;\in\; \tilde{P} - S = \{tA + sB : t, s \in \mathbb{R}\}.$$

But then because $A, B \perp N$ we have $X - S \in P_{0,N}$, where 0 is the null vector. Now,

$$X - S + S = X \in P_{0,N} + S = P_{S,N}.$$

□

As Theorem 2.3.18 shows, the notation using the pair (S, N) is valid. Given a parametric

surface $\beta(u, v)$ with $(u, v) \in [0, 1]^2$ and two parameters $(u^*, v^*) \in [0, 1]^2$, we can construct a plane that is tangent to $\beta(u^{(0)}, v^{(0)})$ by using the u and v gradients of β and taking their cross product. We make the following definition.

Definition 2.3.19. Let $\beta(u, v)$ with $(u, v) \in [0, 1]^2$ be a parametric surface that is differentiable in u and v. Then the tangent plane of β at $(u^{(0)}, v^{(0)}) \in [0, 1]^2$ is given by the plane

$$T_\beta(u^{(0)}, v^{(0)}) := P_{\beta(u^{(0)}, v^{(0)}), N_\beta(u^{(0)}, v^{(0)})},$$

where

$$N_\beta(u, v) := \frac{\nabla_u \beta(u, v) \times \nabla_v \beta(u, v)}{||\nabla_u \beta(u, v) \times \nabla_v \beta(u, v)||}$$

is the normal vector of the surface β at (u, v).

In this section (and in CoolingGen), the surface $\beta(u, v)$ is always differentiable in u and v. Suppose we want to intersect the plane $P_{S,N}$ with the parametric surface $\beta(u, v)$ for $(u, v) \in [0, 1]^2$. If the surface and the plane do intersect in some point, there exists $(u^{(0)}, v^{(0)}) \in [0, 1]^2$ such that $S = \beta(u^{(0)}, v^{(0)})$ is a valid representation for the support vector. If we know such a $(u^{(0)}, v^{(0)})$, then surely there are points in the neighborhood of $(u^{(0)}, v^{(0)})$ that also lie in the plane.

To find such a neighboring point, we construct the line of intersection of the tangent plane $T_\beta(u^{(0)}, v^{(0)})$ and the target plane $P_{S,N} = P_{\beta(u^{(0)}, v^{(0)}), N}$. Using the normal notation of the planes and the fact that $S = \beta(u^{(0)}, v^{(0)})$ lies in the intersection, we can easily find the tangent intersection line.

Theorem 2.3.20. Let P_{S,N_1} and P_{S,N_2} be two planes with N_1 and N_2 not parallel. Then

$$P_{S,N_1} \cap P_{S,N_2} = L_{S,S+N}(-\infty, \infty),$$

where $N = \frac{N_1 \times N_2}{||N_1 \times N_2||}$ and $L_{S,S+N}(-\infty, \infty)$ is the line through S and $S + N$.

Proof. Let $t \in \mathbb{R}$. Notice that $tN \perp N_1$ and $tN \perp N_2$. Therefore, $tN \in P_{0,N_1} \cap P_{0,N_2}$, which is equivalent to $S + tN \in P_{S,N_1} \cap P_{S,N_2}$. But $S + tN = (1 - t)S + t(S + N) = L_{S,S+N}(t)$ according to Definition 2.3.8. □

We can define the tangent intersection line for the intersection parameters $(u^{(0)}, v^{(0)})$ as

$$L^{(0)}(t) = \beta(u^{(0)}, v^{(0)}) + t \cdot \left(\frac{T_\beta(u^{(0)}, v^{(0)}) \times N}{||T_\beta(u^{(0)}, v^{(0)}) \times N||} \right),$$

for $t \in \mathbb{R}$.

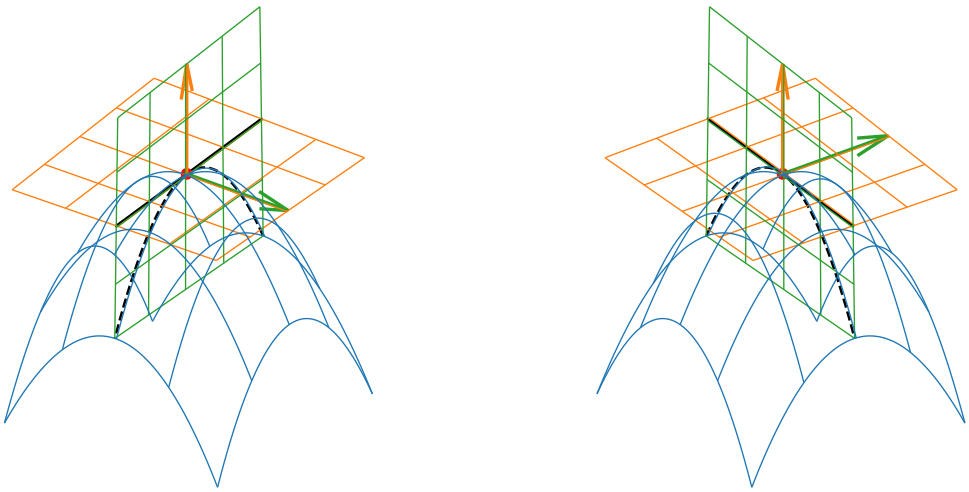


Figure 2.17: We intersect the blue surface β with the green plane $P_{S,N}$. The intersection point $\beta(u^{(0)}, v^{(0)})$ is represented by the red dot. The orange tangent plane at $(u^{(0)}, v^{(0)})$ is intersected with the green plane to find the black line $L^{(0)}$. In a neighborhood of the red dot, the black line runs close to the black dashed line, which is the true intersection. The arrows represent the normal vectors of the planes.

Using an appropriate step size $\alpha \in \mathbb{R}$, we can project the point $L^{(0)}(\alpha)$ (which lies in both the tangent plane $T_\beta(u^{(0)}, v^{(0)})$ and the target plane $P_{S,N} = P_{\beta(u^{(0)},v^{(0)}),N}$) onto the surface β using the methods from Section 2.3.1 to acquire the point $\beta(u^{(1)}, v^{(1)})$. If this point lies too far away from the tangent intersection line $L^{(0)}$, we discard the point and choose a step size less than α and repeat the process. This iteration step is repeated until the step size falls below a certain threshold α_0.

We are then left with n points which lie on the intersection curve. We interpolate between these points using NURBS. If necessary, the control points of the NURBS curve can then be projected onto the plane $P_{S,N}$ to ensure that the curve is planar.

3 Jet Engine Specific Methods

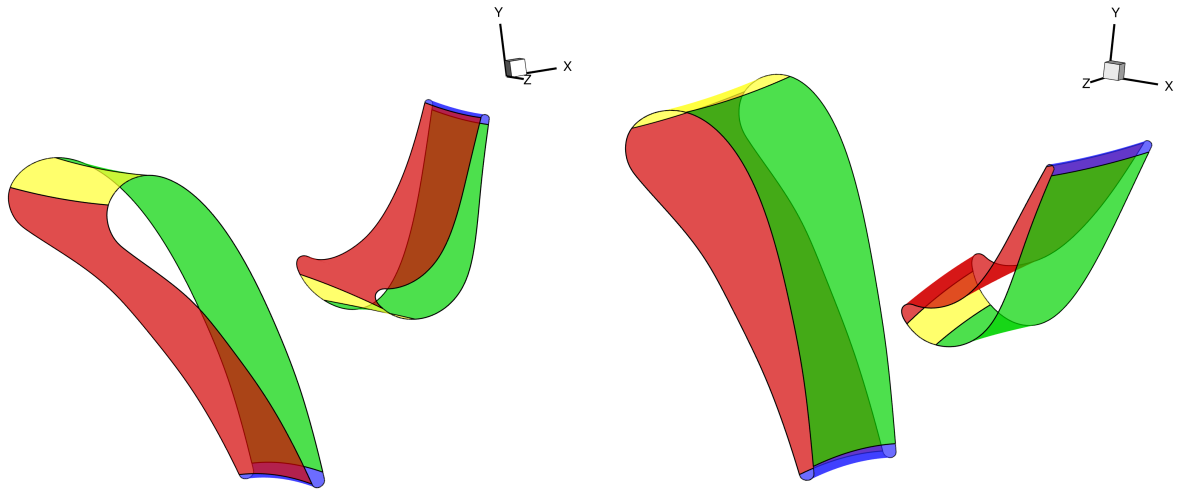
3.1 Fundamental Terms

(sauce: books on cooling) The creation of cooling geometries inside of turbine blades is not only vital for the enhancement of the thermodynamic efficiency and power output of turbine engines. The materials melting temperature is often exceeded by that of the turbine inlet. To withstand this extreme condition, cooling is required. The blade geometries that we are concerned with in the development of CoolingGen are the ones that succeed the inlet in axial direction.

CoolingGen creates geometries which lie inside of such a blade surface $B(u, v)$, which is an output of BladeGen, a tool developed by the DLR specifically for the design of blade surfaces. $B(u, v)$ is closed in u-direction. For a given u, increasing v yields a further distance from the turbines hub. Because of rotational symmetry of the turbine, it is sometimes convenient to describe these geometries in cylindrical coordinates. The transformation from Cartesian coordinates to cylindrical coordinates is given by the map

$$Z : (x, y, z) \mapsto (x, \sqrt{y^2 + z^2}).$$

Given the associated circumferential angle $\theta = \arctan2(y, z)$, the transformation to cylindrical coordinates is invertible.



(a) View on pressure side.

(b) View on suction side.

Figure 3.1: In (a) and (b), the left is surface represents a stator and the right surface represents a rotor. Yellow marks the leading edge, red the pressure side, green the suction side and blue the trailing edge.

We make the distinction between two kinds of blades, rotor and stator blades. While the rotor blades rotate around the hub, the stator blades have fixed positions. This was taken into account when designing CoolingGen, so both kinds are currently supported. Furthermor, the blade surface $B(u, v)$ has a named partitioning in u-direction into four distinct parts. The

leading edge, the suction side, the trailing edge and the pressure side.

3.2 The S2 Stream Surface and Stream Surface Coordinates

- figure of s2m with correct labels

(source: Wu, 1952 at NACA) A stream line is a line that at every point is tangential to the velocity vector field of the flow of the fluid. Its generalization, the stream surface, describes a surface with the same property. The flow in a turbine can be described by a generalization of the stream surface. When designing turbine blades, it is common practice to describe the flow in two families of stream surfaces. These surfaces are called S_1 and S_2. The S_1 surfaces, also known as *blade-to-blade* surfaces, describe the flow behavior between individual blade sections, whereas their counterpart, the S_2 surfaces, which are also known as *meridional* surfaces, describe the flow through the turbine.

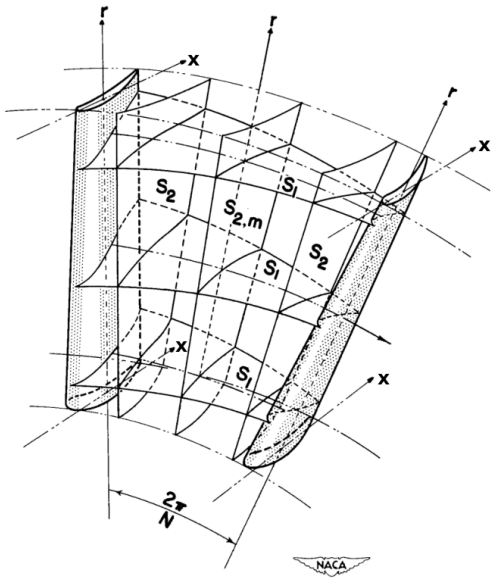


Figure 3.2: Example S_1 and S_2 surfaces of a turbomachine.

Each S_2 surface is a parametric surface and therefore a map

$$s(u, v) : [0, 1]^2 \to \mathbb{R}^3, \quad (u, v) \mapsto (x, r, m),$$

where x is the axial coordinate, $r = \sqrt{y^2 + z^2}$ the distance from the turbine hub and m is the arc length.

To design the blades for turbomachinery, BladeGen and CoolingGen make use of a certain S_2 surface, the so called $S_{2,m}$ surface. This surface lies midway between two blades. Each stream line described by $S_{2,m}$ can then be written as $S_{2,m}(\cdot, v)$. By intersecting the cylindrically transformed u-isoparametric curves $Z(B(u, \cdot))$ of the blade with each stream line $S_{2,m}(\cdot, v)$, we can construct a curve $B^{(v)}_{\text{stream}}(u)$ from the intersection points $P_{u,v}$, given by

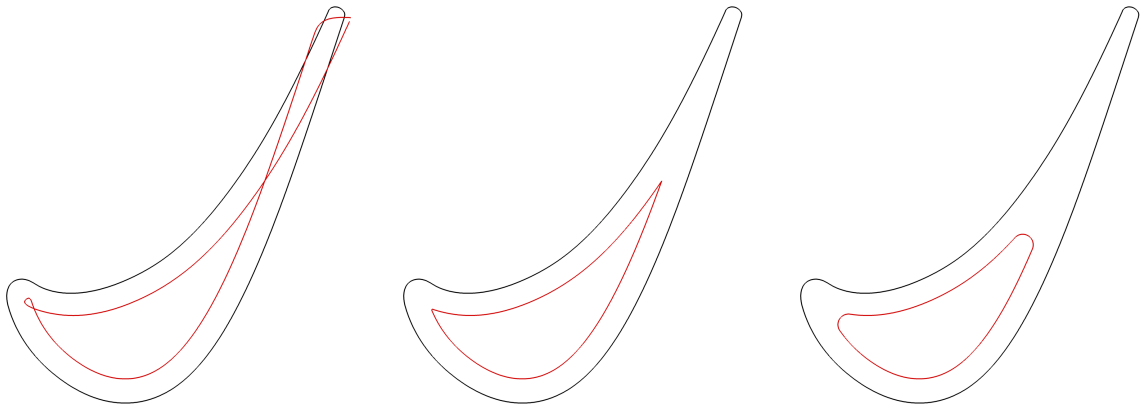
$$B^{(v)}_{\text{stream}}(u) = (m, r\theta, v),$$

where m is the first argument of $P_{u,v}$, the radius r is equal to the second argument of $P_{u,v}$ and θ is the circumferential angle and v is constant for each $B^{(v)}_{\text{stream}}$, making the curve planar. This coordinate system with respect to arc length m and radius-angle product $r\theta$ is a so called *stream surface coordinate system*. Even though the input blade surface $B(u, v)$ lies in $\mathbb{R}^3$, the most important methods of CoolingGen are operations on objects in this stream surface coordinate system.

4 Results

4.1 Chambers

Channels are hollow sections inside a turbine blade which allow for a coolant to pass through. For the construction of channels, we distinguish between straight and curved sections of the channel. The straight sections represent hollow bodies that let the coolant pass in v-direction of the turbine blade. These sections are called *chambers*. The curved sections let the coolant pass in u-direction of the turbine blade. We call the latter sections *turns*. Inside turns, the coolant flows between chambers. In this section, we present how chambers are constructed, whereas turn construction is thoroughly explained in the next section.



(a) Offsetting. (b) Trimming. (c) Adding fillets.

Figure 4.1: Shrinking a profile in $(m, r\theta)$. The black curve represents $B^{(v_i)}_{\text{stream}}(u)$, whereas the red curve in (c) represents $S^{(v_i)}_{\text{stream}}(u)$.

First, we apply the stream surface coordinate transformation presented in Section 3.2 to a fixed number N of v-isoparametric curves of the blade to calculate the profiles $B^{(v_i)}_{\text{stream}}(u)$ for $i \in \{1, ..., N\}$. These profiles are then offset by an input parameter. The profile is automatically trimmed adequately and fillets are added, the radius of which are also declared as an input. We refer to the resulting profile as *shrunk profile* and declare it $S^{(v_i)}_{\text{stream}}(u)$. This process is depicted in Figure 4.1.

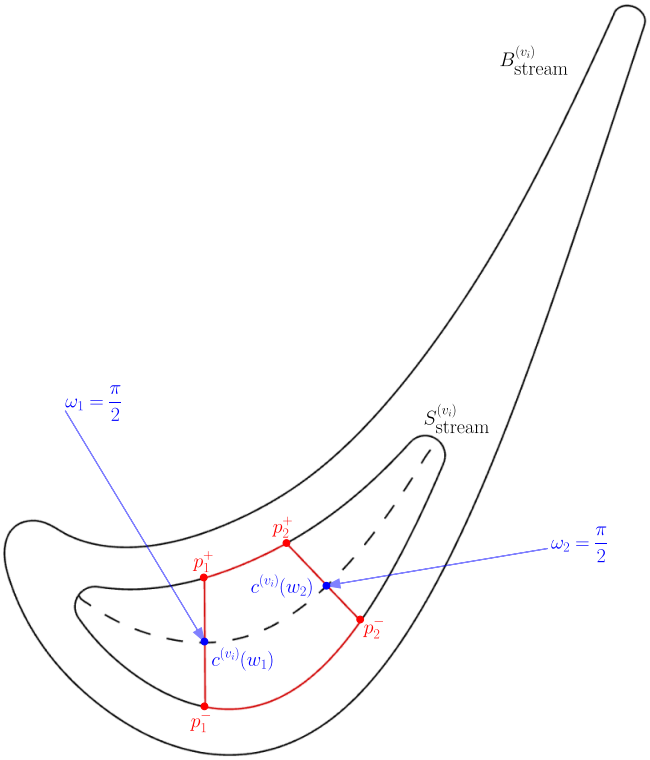


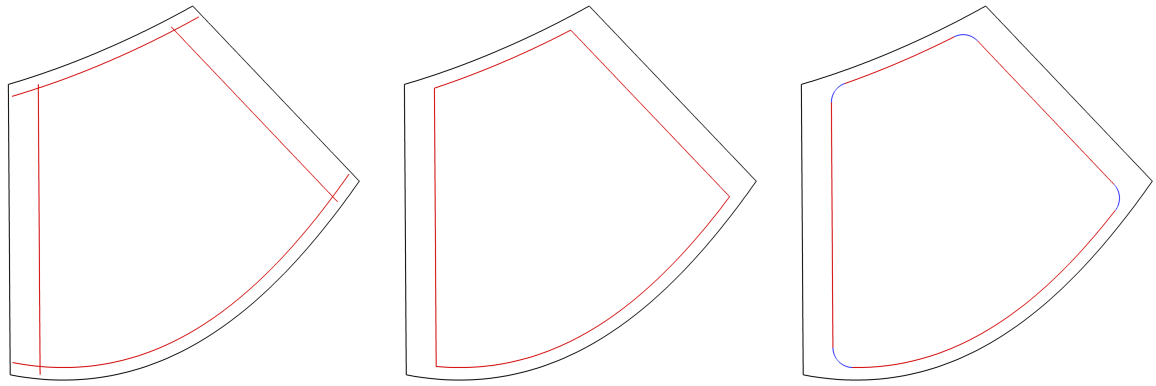
Figure 4.2: Subdivision of profile along the camber curve. The camber curve is represented by the dashed line.

We calculate the camber curve of each shrunk profile $S^{(v_i)}_{\text{stream}}$. The camber curve $c^{(v_i)}$ is a design aid which is defined by the following property. A point x lies on the camber curve $c^{(v_i)}$ of $S^{(v_i)}_{\text{stream}}$ if and only if for some d its offset curve $O_d^{S^{(v_i)}_{\text{stream}}}$ has a self intersection at x. Along the camber curve, we define the wall input, which is given by a sorted set of parameters w_j representing the positions of each wall relative to the camber curve, $j \in \{1, ..., N_C - 1\}$, where N_C is the number of chambers that shall be created. For each position, we also define an angle ω_i. Each position-angle pair represents two rays

$$c(w_j) + sR_{\omega_j}\nabla c(w_j) \quad \text{and} \quad c(w_j) - sR_{\omega_j}\nabla c(w_j)$$

for $s \in [0, \infty)$, where R is a rotation matrix like the one presented in Equation (2.12). Using the Algorithm 6, we can find the intersection p_j^+ and p_j^- of the rays and the profiles. The line segment $\overline{p_j^+ p_j^-}$ now represents the j-th wall.

The blade is then partitioned at these intersection points. For each such partitioning, we are left with four curves per chamber profile. Some of these curves are the walls, which are curves that represent straight lines between p_j^+ and p_j^-. There are four distinct cases. The first case is $N_C = 1$. In this case, the four curves are given by the pressure side, the leading edge, the suction side and the trailing edge partitioning of the input blade surface $B(u, v)$. If however $N_C > 1$, we have three more cases. The chamber profile located at the leading edge, which we refer to as leading chamber profile, consists of the partition of the pressure side between the leading edge and p_1^+, the leading edge, the partition of the suction side between the leading edge and p_1^- and



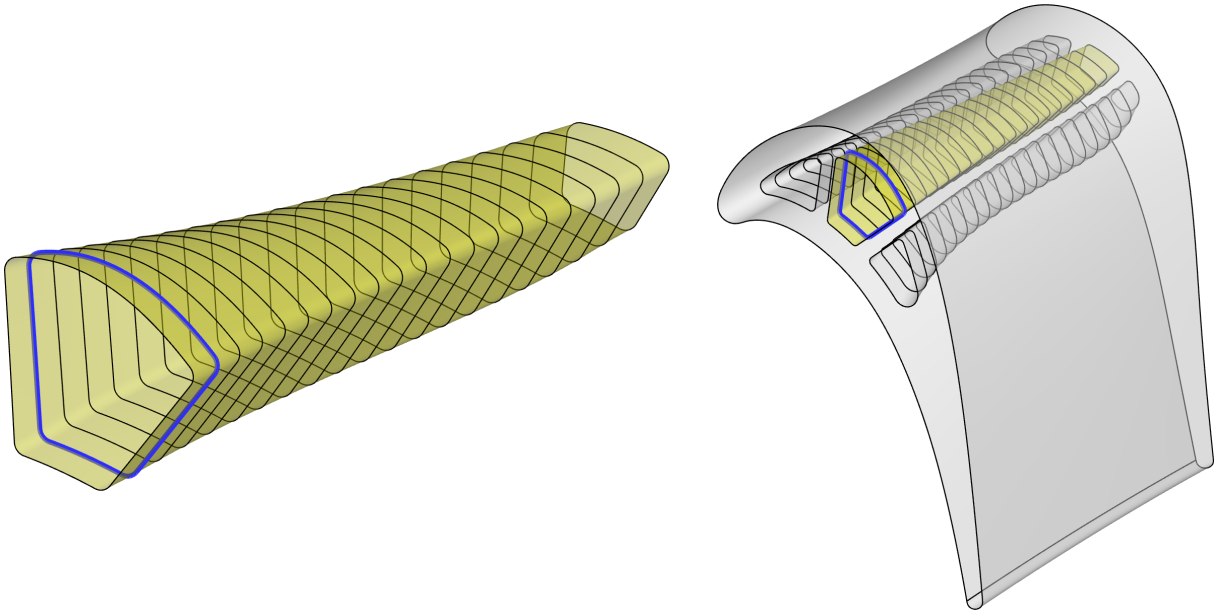
(a) Offsetting. (b) Trimming. (c) Adding fillets.

Figure 4.3: Construction of a chamber profile.

lastly the first wall curve. The chamber profile located at the trailing edge, which we refer to as trailing chamber, consists of the partition of the pressure side between trailing edge and $p^+_{N_C-1}$, the (N_C-1)-th wall, the partition of the suction side between trailing edge and $p^-_{N_C-1}$ and the trailing edge. For the j-th chamber profile, where $j \neq 1$ and $j \neq N_C - 1$, these four curves are the partition of the pressure side between p^+_j and p^+_{j+1}, the j-th wall, the partition of the suction side between p^-_j and p^-_{j+1} and the $(j+1)$-th wall. This procedure is exemplified in Figure 4.2.

After performing this partitioning, we can offset the curves. However, each curve is offset differently. In the wall input, the thickness of the wall in each direction is specified as an input. In fact, this input specification is done using NURBS curves. For each partition, the user of CoolingGen specifies control points (v_k, d_k), where $v_k \in [0,1]$ is the surface parameter and d_k is the respective thickness with $k \in [1, ..., K]$. The leading and trailing profile offsets are linearly interpolated between suction and pressure side offsets to guarantee differentiability of the chamber profile at the leading and trailing edge transitions, respectively. Since after offsetting, the four chamber profile parts do not necessarily align, we connect the curve ends either by curve-curve intersection, ray-curve intersection or ray-ray intersection, depending on which one is applicable. The rays are in this case defined by $R_{A,B}$, where A represent the curve boundary points and B represent the gradient at the curve boundary points.

After connecting and trimming the offset curves using the methods in Section 2.3.4, we are left with four trimmed offset curves that intersect non-differentiably at their boundary points. We apply the fillet curve method presented in Section 2.3.5. These steps can be seen in Figure 4.3.



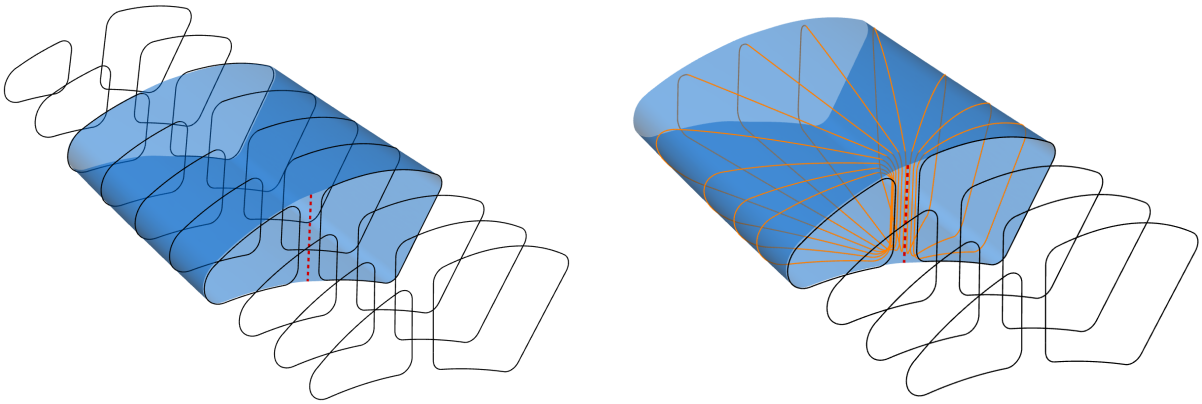
(a) A single chamber.

(b) Three chambers inside of blade surface.

Figure 4.4: Lifted chamber surfaces.

We can use the inverse coordinate transformation from the $(m, r\theta)$ stream surface coordinate system to the Cartesian coordinate system. Interpolation between the transformed chamber profiles yields N_C many three-dimensional chamber surfaces.

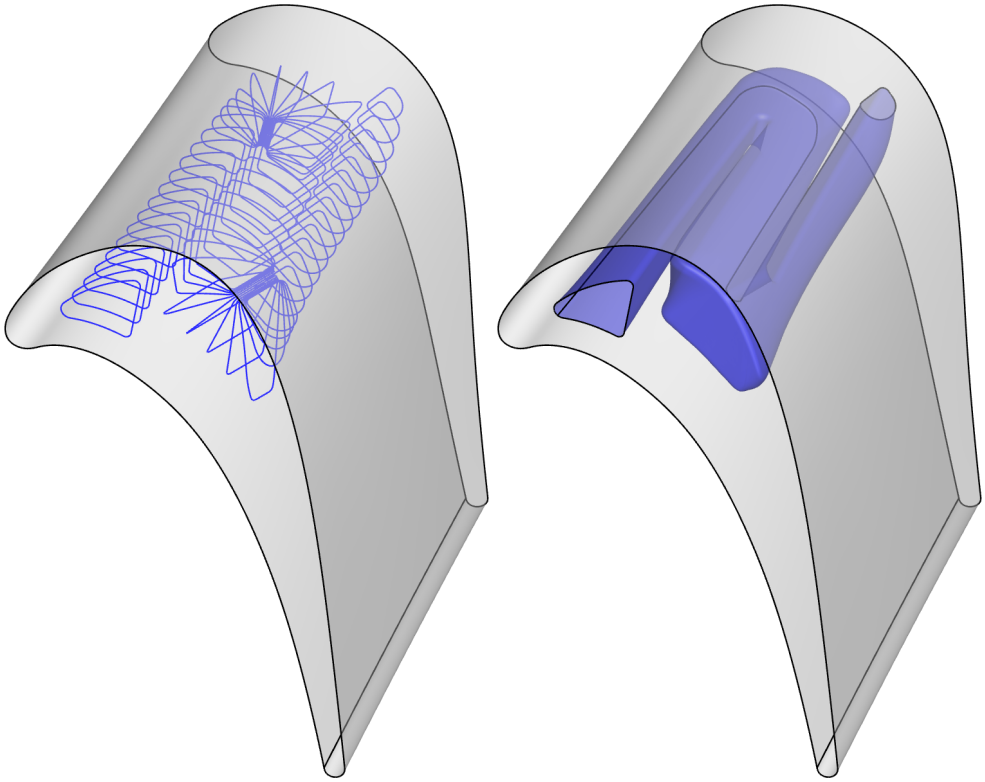
4.2 Turns



(a) Combining the chambers. The axis of rotation is marked in red.

(b) Intersecting the combined surface with affine planes that contain the axis of rotation.

Figure 4.5: Construction of a turnaround.

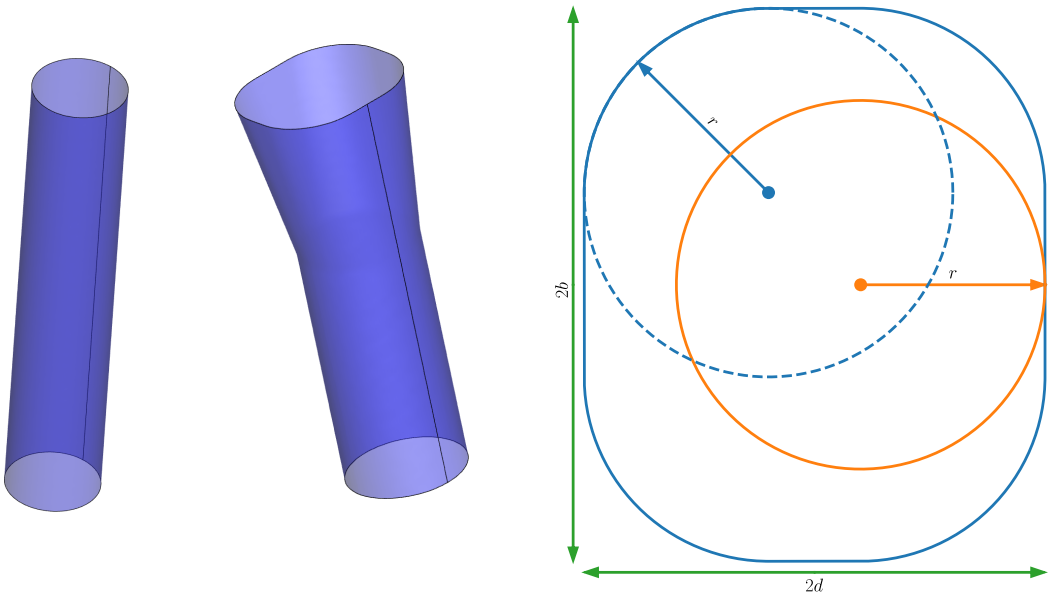


(a) Channel profiles. (b) Channel surface.

Figure 4.6: Cooling channel with three chambers and two turnarounds.

4.3 Film Cooling Holes

- parametric definitions of delimiting curves
- explain how these are made
- then explain how they are basically made out of rays and how that is helpful with intersections



(a) Cylindrical. (b) Fan shaped. (c) Delimiting curves.

Figure 4.7: Film cooling holes and their delimiting curves

4.4 Impingement Inserts

4.5 Slots

4.6 Export for CENTAUR

4.7 Export for Open CASCADE

5 Discussion

5.1 Future Work

5.2 Conclusion

[Pie97]

6 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.