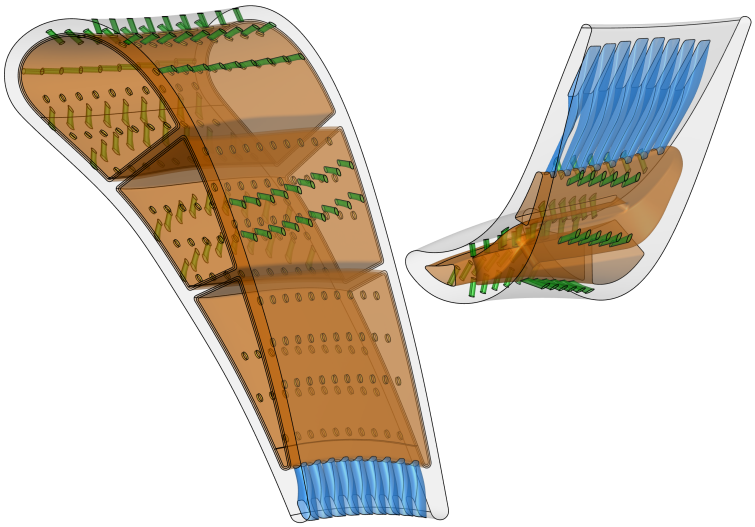
GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

DLR

Master's thesis in
Applied Computer Science

CoolingGen

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



December 7, 2022

Julian Lüken
julian.lueken@dlr.de

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

First examiner:	Prof. Dr. Gerlind Plonka-Hoch
Second examiner:	Prof. Dr. Christoph Lehrenfeld
Advised by:	M.Sc. Robin Schöffler

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, December 7, 2022

Abstract

The generation of cooling geometries within turbine blades is an important aspect of jet engine design and research. Generating such geometries with conventional CAD tools is very time-consuming. CoolingGen is a parametric CAD tool for the creation of cooling geometries within turbine blades. Due to it's parametrization, there is a semantic interface between user input and mechanical function, which is not given in conventional CAD tools. In this thesis the theoretical and practical aspects of parametric geometry generation by CoolingGen are highlighted. It serves as a manual of important geometric methods and processes within CoolingGen.

Zusammenfassung

Die Generierung von Kühlungsgeometrien innerhalb von Turbinenschaufeln ist ein wichtiger Teilaspekt des Triebwerkdesigns und der Triebwerksforschung. Mit konventionellen CAD Tools lassen sich solche Geometrien zwar erzeugen, allerdings nur unter großem Zeitaufwand. CoolingGen ist ein parametrisches CAD Tool zur Erstellung von Kühlungsgeometrien innerhalb von Turbinenschaufeln. Durch die Parametrisierung gibt es eine semantische Schnittstelle zwischen Eingabe und mechanischer Funktion, welche in konventionellen CAD Tools nicht gegeben ist. In dieser Thesis werden die theoretischen und praktischen Aspekte der parametrischen Geometrieerzeugung durch CoolingGen beleuchtet. Sie dient als Handbuch wichtiger geometrischer Methoden und Prozesse innerhalb von CoolingGen.

Contents

1 Introduction

Modelling cooling geometries inside of turbine blades and vanes using computer-aided design (CAD) methods currently is a difficult but necessary process in the design of modern jet engine turbines. To simplify this process, the German Aerospace Center (DLR, Deutsches Zentrum für Luft- und Raumfahrt) has been concerned with finding an alternative to conventional CAD software. As part of many CAD-related research projects at the DLR, parametric CAD has proven to be a simpler alternative to its conventional counterpart. However, the complexity and inevitable inter-dependence of cooling structures have ever since been a challenge in the development of a parametric CAD tool made solely for turbine cooling. In this thesis, we present CoolingGen, a high-performance tool that uses custom-made algorithms to evidently satisfy the special goal of creating three-dimensional models of turbine cooling structures.

1.1 Cooling Design

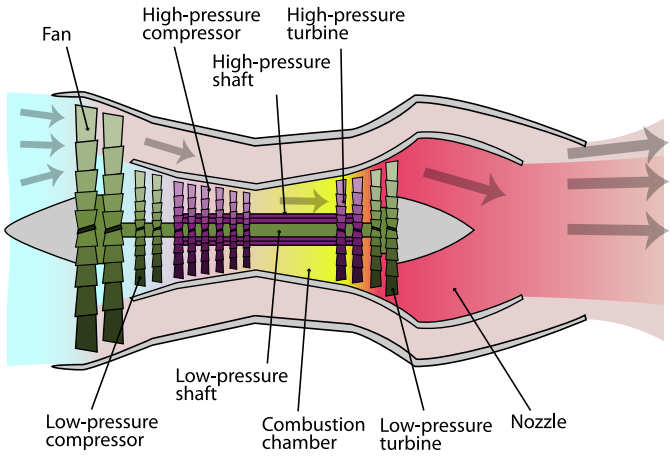
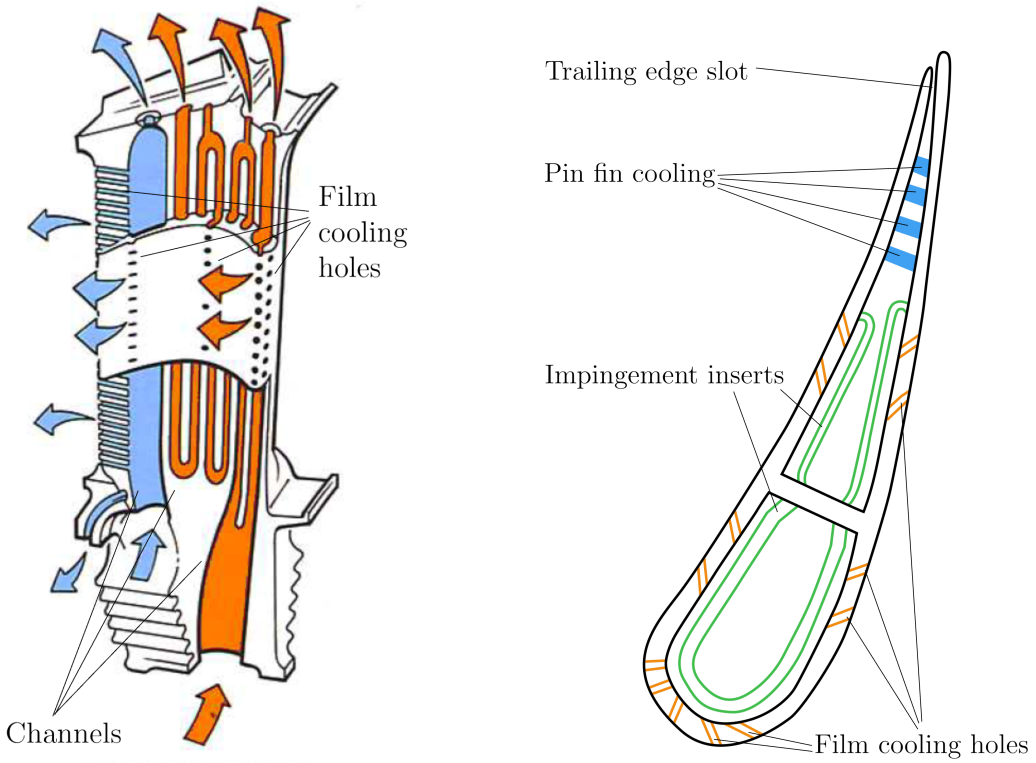


Figure 1.1: Schematic diagram illustrating the operation of a 2-spool, low-bypass turbofan jet engine, with low pressure spool in green and high pressure spool in purple [Aai08].

In a jet engine, kinetic energy is supplied from high pressure gas combustion. To increase the jet engine's thermodynamic efficiency, the firing temperature of the combustion chamber must be increased. An example jet engine can be seen in Figure 1.1. In modern jet engine turbines, the hot gas temperature after combustion at up to $2000K$ exceeds the materials' thermal limits. To combat this extreme condition, 20% to 30% of the compressed air mass that precedes the combustion chamber is used as a coolant, which is passed through the blades and vanes of the high pressure turbine. On the other hand, the amount of compressed air used for cooling correlates negatively with the thermodynamic efficiency of the jet engine. Alongside material selection and treatment of the coolant, the filigree design of cooling measures includes the creation of sensible cooling geometries that are located within each turbine blade or vane [Gia20].



(a) Three-dimensional view [Roy15].

(b) Profile view [Gia20].

Figure 1.2: Two different turbine cooling blade setups showing the different geometries of the cooling structures.

Turbine cooling geometries consist of but are not limited to the following parts: internal cooling, film cooling, impingement cooling, pin fin cooling and ejection slots. The internal cooling is comprised of cooling channels, which are hollow sections inside a turbine blade or vane. They allow for a coolant to pass through. A rib-like structure is added to the inside of these channels. These ribs are referred to as rib turbulators. The film cooling consists of holes that transfer the coolant from the inside of a channel to the outside of the blade or vane. An array of these holes provides a protective fluid layer that restrains net heat transfer to the blade or vane. Impingement inserts are thin sheets that reside within channels and have cylindrical holes in them. Another type of geometry which is also referred to as impingement cooling is comprised of holes that connect neighboring channels. Pin fins connect the suction and the pressure-side of the trailing edge channel. The ejection slots are hollow bodies that transport the coolant from the inside of the trailing edge channel to the trailing edge or pressure side of the turbine blade or vane [Gia20]. Most of these geometries can be seen in Figure 1.2.

1.2 State of the Art

Conventional CAD systems are modelers that allow their user to create free-form curves and surfaces using interactive control points that can be moved to any location on a three-dimensional canvas. In these systems, common geometric features can often also be generated using a set of input parameters. This gives the user an unlimited amount of strategies for creating a CAD model fit for their use. However, the amount of possible modelling strategies impedes the adaptability

of most resulting models [CCC16].

Because of it's intrinsic adaptability, parametric feature-based CAD as part of conventional CAD has become an important and widely used tool among many mechanical engineering applications. To adapt the geometry of a mechanical model using parametric CAD, the user simply has to change a few parameters instead of moving control points, which in themselves bear no underlying semantic structure regarding their mechanical function [Sha95].

Using existing CAD software, the creation of cooling geometries is labour-intensive and requires expertise. This is because the construction of blade and vane geometries for the most part does not rely on common features that are included within existing CAD software, but rather on more specific and convoluted features that are only used for turbine cooling.

An alternative to conventional CAD systems are fully parametric CAD systems that rely on a parent-child structure of a model and its constituent parts. In fully parametric CAD software, the user only specifies a select set of (non-)geometric parameters that determine the shape of the output geometry. The number of input parameters determines the flexibility of the shape of an output model. A high number of parameters generally improves flexibility, although keeping this number as low as possible ensures simple user interaction. Developing such a software at the DLR ourselves yields the benefit of the option of implementing arbitrarily complex geometric features using a purpose-made parametrization at will.

At the DLR, blade and vane contours are already successfully designed using such a fully parametric CAD tool called BladeGen. We therefore know that the approach of automating the modelling process bears great potential. In the associative structure of parametric CAD, blades and vanes serve as the parent elements of the cooling geometries we are concerned with in the development of CoolingGen.

1.3 Problem Statement

CoolingGen is a fully parametric CAD software using Non-Uniform Rational B-spline (NURBS) surfaces. The program was written in the programming language C and allows for the creation of three-dimensional turbine cooling geometries of blades and vanes. It was first developed by Timo Schumacher and Christian Voß in 2013 at the DLR. The 2013 release, while being a great proof of concept, lacked an underlying framework of established geometric methods and thus failed to fulfill real world requirements. In this version of CoolingGen, only the creation of cooling channels was supported. The program was tailored to fit only a few parameter configurations on a certain parent blade surface. The input parameters were handled using a custom ASCII file format.

In 2022, CoolingGen was largely revised as part of this thesis project to support more realistic cooling channels. State of the art film cooling, impingement sheets and ejection slots were added in this context. The input parameters are since specified in the Extensible Markup Language (XML). To create a complete set of cooling geometries for a single blade or vane, CoolingGen currently averages out at approximately 70 seconds on a modern home computer.

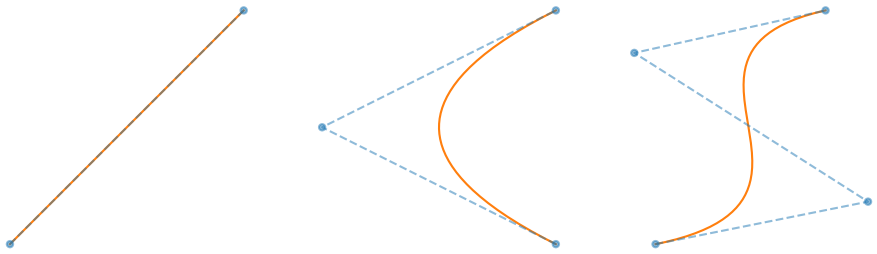
This thesis essentially highlights the geometric methods that are imperative to achieve CoolingGen's goals in terms of accuracy and speed in Chapter 2. The fundamental methods of jet engine design required for channel creation are presented in Chapter 3. Combining the methods from Chapter 2 and Chapter 3, we are able to create all aforementioned cooling geometries. Their exact construction is explained and illustrated using examples in Chapter 4.

2 Geometric Methods

As stated before, 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 projection, 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 CAD. In this section, we present the famous algorithm of de Casteljau and some important properties of Bézier curves according to [Far01].



(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

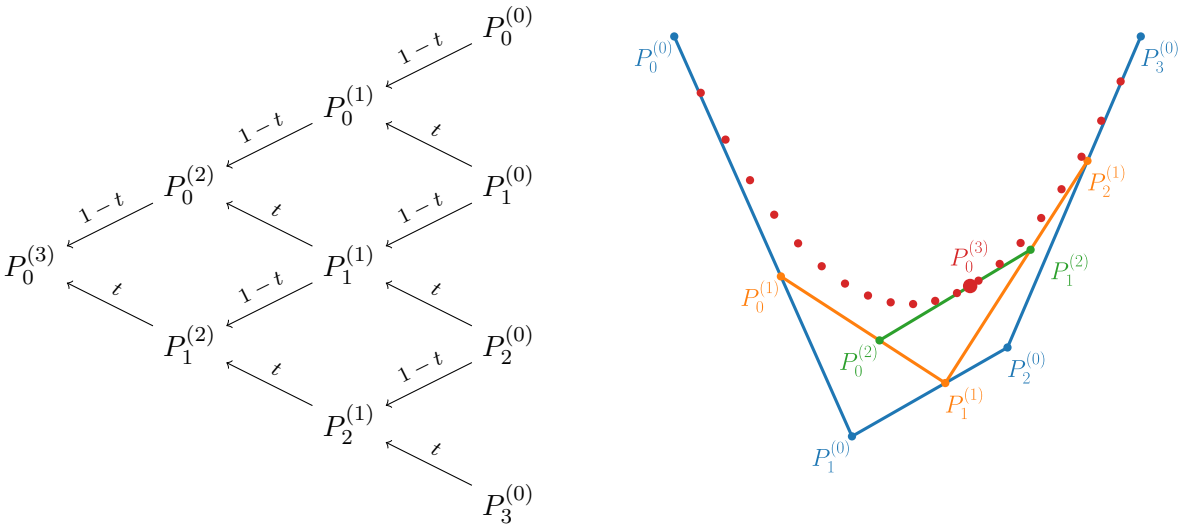
$$P_j^{(n)} = \sum_{k=j}^{n+j} \binom{n}{k} (1-t)^{n-k} t^k P_{j+k}$$

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 CAD 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

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

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 NURBS, which is their prevailing counterpart.

2.2 Non-Uniform Rational B-splines (NURBS)

NURBS are another tool for curve and surface modelling. There is a somewhat common joke that describes the acronym NURBS as "Nobody Understands Rational B-Splines", which can be found at the back of [Pie97]. In this section, we invalidate this punch line. First of all, we discuss B-splines and construct B-spline curves and surfaces according to [Far01], including the famous algorithm of de Boor. Then we apply simple transformations to the B-spline curves and surfaces to acquire NURBS curves and surfaces as described in [Pie97].

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

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

□

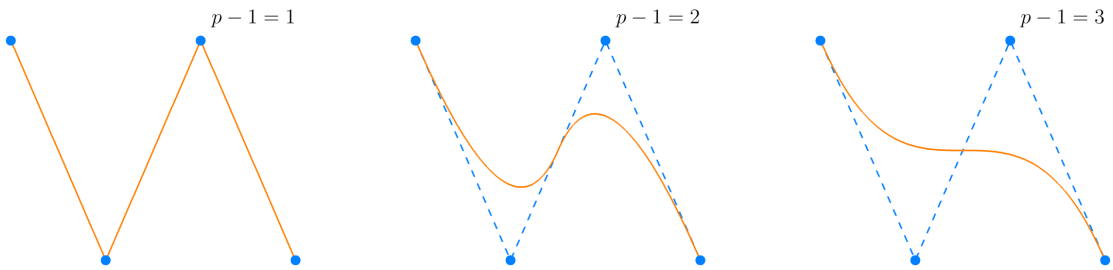


Figure 2.3: A set of control points and three NURBS curves of different degrees.

An example of a NURBS curve can be seen in Figure 2.3. 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.

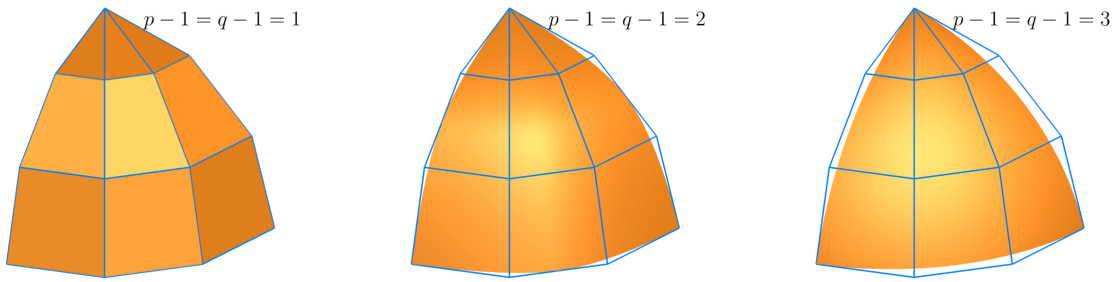


Figure 2.4: A set of control points and three NURBS surfaces of different degrees.

An example of a NURBS surface can be seen in Figure 2.4. 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

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

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

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

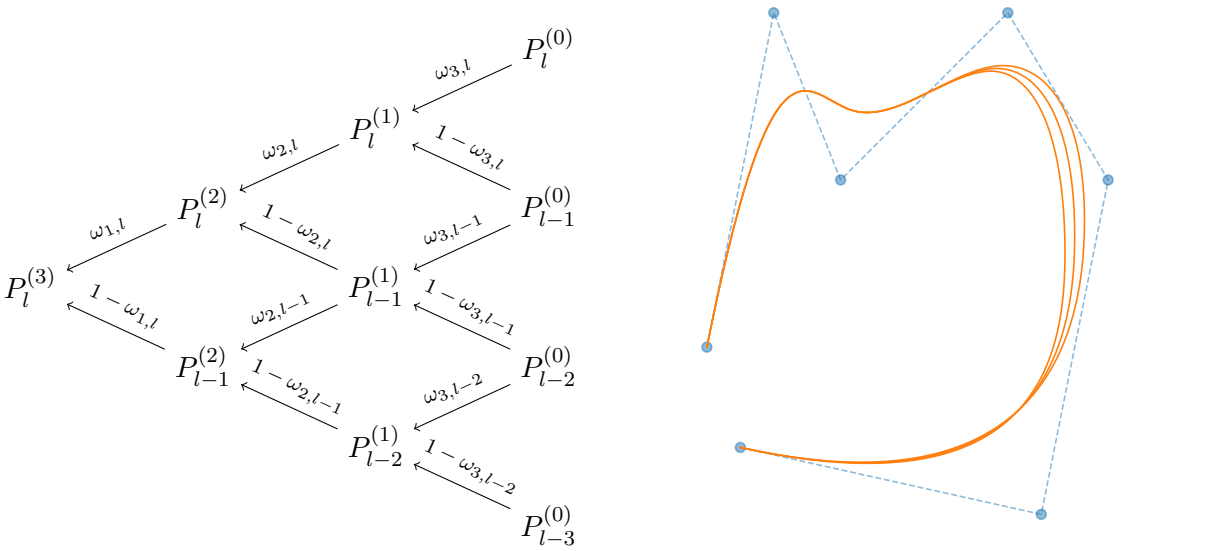
  for $j = 1, 2, ..., p - m - 1$ **do**

    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.

The algorithm is visually represented in 2.5. 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 & 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

In this section, we present an algorithm that projects a point onto a parametric curve or surface according to [Pie97]. More formally, our problem statement is the following. 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.

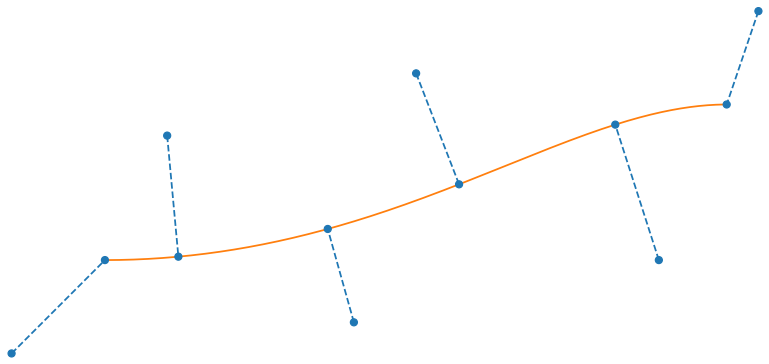


Figure 2.6: Point projection. Each blue point on the orange curve is the projection of the point connected to it with a dotted line.

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 FINDPOINTPROJECTIONINITVALUES(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 Projection

Input
  M — number of iterations
  $\epsilon > 0$ — differentiation step (small)
procedure POINTPROJECTION(γ, Q)
  $l, u, m \leftarrow$ findPointProjectionInitValues(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. An example of point projection can be seen in Figure 2.6.

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 $Q \in \mathbb{R}^d$ onto a parametric surface $\beta(u, v)$ with $u, v \in [0, 1]$ by generalizing the presented methods. This procedure is also described

in [Pie97]. 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.

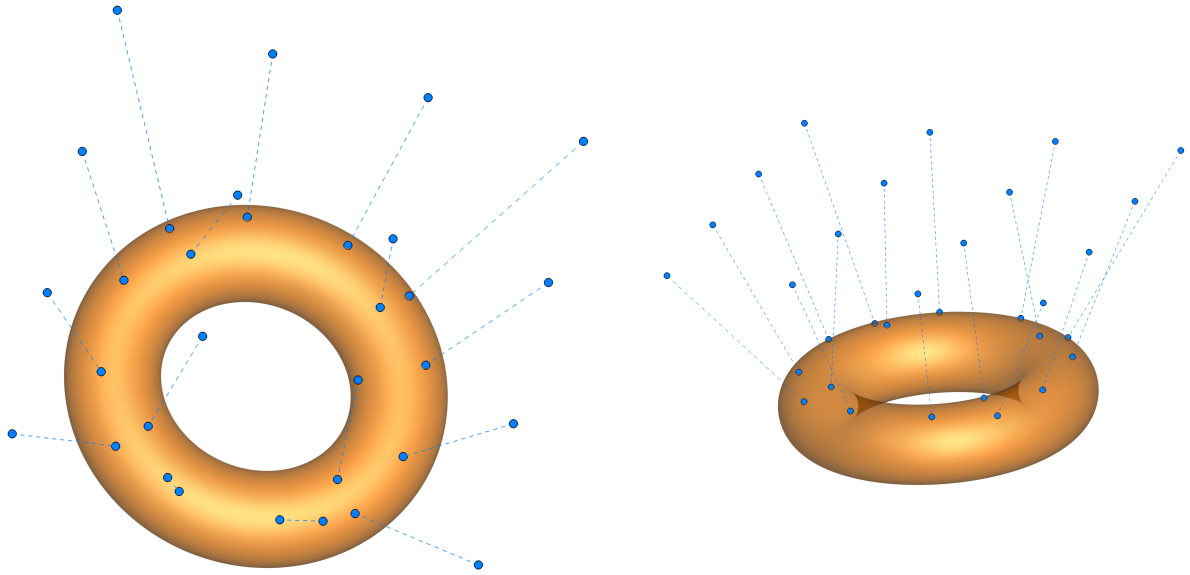


Figure 2.7: Point projection. Each blue point on the orange surface is the projection of the point connected to it with a dotted line.

2.3.2 Ray-Curve & Ray-Surface Intersection

This section describes the intersection of a ray and a curve according to [Har96].

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

More formally, 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 small distance increment ss 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 projection 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 $\emptyset$.

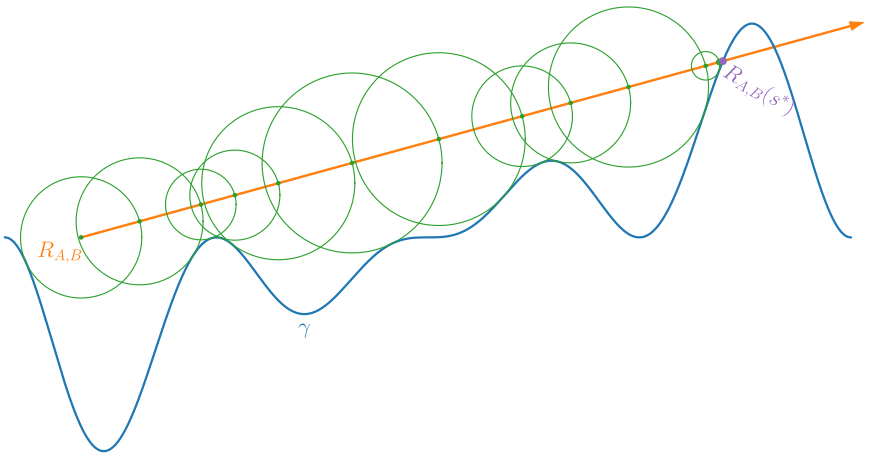


Figure 2.8: Ray marching.

Figure 2.8 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 projection algorithm proves to be relatively simple.

Algorithm 6 Ray Marching

```
1: procedure IntersectCurveRay(γ, A, B)
2:     B ← B/||B||, s ← 0, P ← A
3:     for k = 1, 2, ..., N do
4:         s ← D(γ(pointProjection(γ, P)), P)
5:         P ← P + sB
6:         if s < ε then
7:             return {P}
8:     return ∅
```

Using the point projection algorithm for surfaces (instead of the point projection algorithm for

curves) to find the distance between a point and a surface, we can easily generalize this algorithm to find the intersection of a ray and a surface. Formally, the only difference between the curve and the surface version of the algorithm lie in the point projection, which was described in Section 2.3.1.

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 $I(\gamma_1, \gamma_2) = \emptyset$.

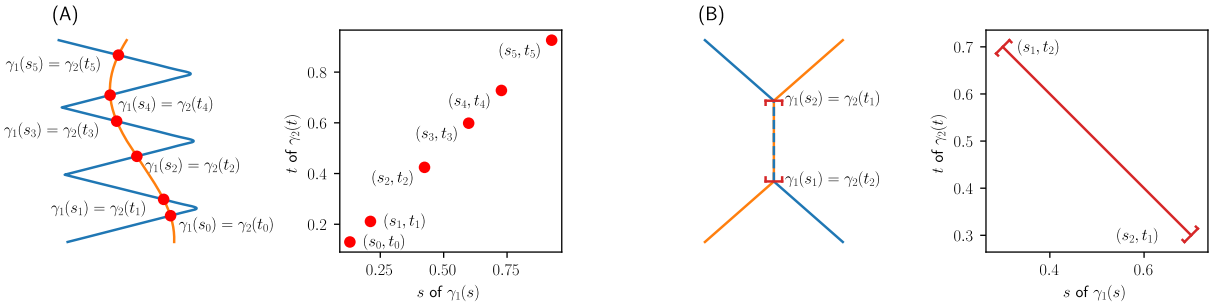


Figure 2.9: In (A) we have $|I(\gamma_1, \gamma_2)| < \infty$, whereas in (B) we have $|I(\gamma_1, \gamma_2)| = \infty$.

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 2.9).

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 $[a,b]$.

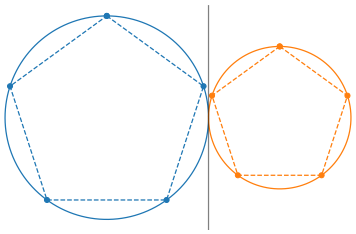


Figure 2.10: Two curves that intersect in a point tangentially and their linear approximations.

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 2.10) 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 point 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 $O(N^2)$.

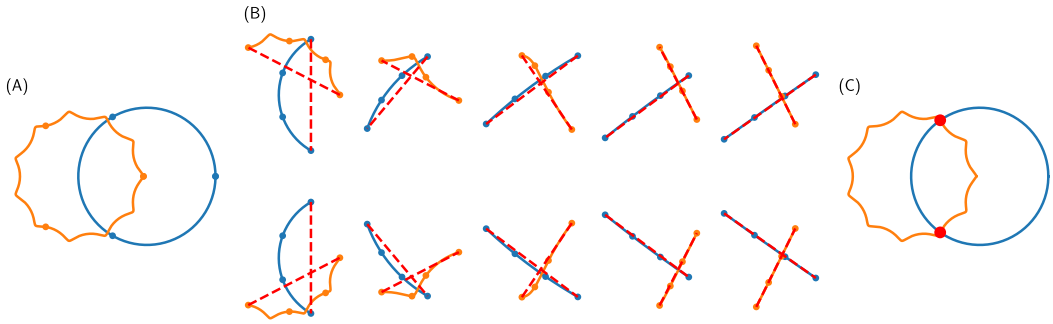


Figure 2.11: Demonstration of the curve intersection algorithm. Figure (A) shows the input curves, Figure (B) the different steps and Figure (C) the result. The dots on the curve represent the samples on the curve. The red lines represent the intersection of two line segments.

The algorithm in Figure 2.11 first calculates estimates of the intersection points and then refines those estimates until the target distance falls below α. If a maximum recursion depth $d_{\max}$ is reached before falling below target distance, the estimated point is not returned. 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 of parametric curves. To do this, instead of iterating over all line segments in the first recursion step, we iterate in a triangular manner (in Algorithm 8, replace `[for` $j = 0, ..., N-1$`]` with `[for` $j = i+2, ..., N-1$`]`, but only when calling `intersectCurvesRecursion` for the first time). Details about self-intersections are presented 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 & 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. The definition of offset curves in this thesis is adapted from [ELK97].

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 $\nabla\gamma(t)$.

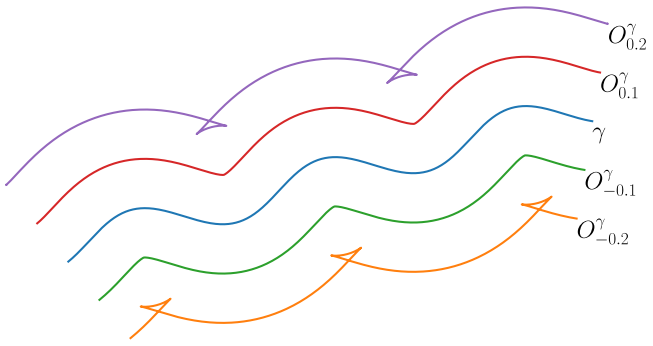


Figure 2.12: Offset curves of a given curve γ for different values of d.

As can be observed in Figure 2.12, 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 2.13: Trimming offset curves at their self intersections.

Now, the restriction $O_d^\gamma|_{T(O_d^\gamma)}$ of the curve O_d^γ to the trim range $T(O_d^\gamma)$ (as depicted in Figure 2.13) 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

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

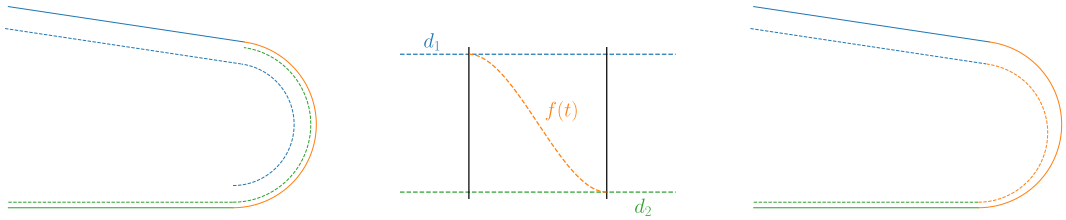
maps $[0,1]$ onto M. Since both $\overline{[0,1] \setminus T(O_d^\gamma)}$ and $T(O_d^\gamma)$ are unions of closed intervals, we can use this procedure to define the trimmed offset curve as

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

or

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

respectively. This curve now satisfies Equation (2.11).



(a) (b) (c)

Figure 2.14: In (a), the blue curve γ_1 and the blue curve γ_2 are offset by different amounts d_1 and d_2, respectively. Offsetting the orange curve β by a constant offset results in a curve that does not align with γ_1 and γ_2 simultaneously. However, if we offset β by a function $f(t)$ as in (b) instead, we can smoothly connect γ_1 and γ_2 using the cubic interpolation offset curve $O_{f(t)}^{\beta}(t)$, which can be seen in (c).

When offsetting adjacent curves by different offsets, the resulting curves do not align. We therefore introduce offset curves $O_{f(t)}^{\gamma}(t)$, which are offset by a function $f(t)$ instead of a constant value, $t \in [0, 1]$. We construct a specific $f(t)$ to ensure smooth connection of neighboring offset curves. To do this, we set $f'(0) = f'(1) = 0$, $f(0) = d_1$ and $f(1) = d_2$. Using a third-degree polynomial to solve for this system of equations, we derive the function

$$f(t) = 2\,(d_1 - d_2)\,t^3 \;-\; 3\,(d_1 - d_2)\,t^2 \;+\; d_1.$$

We call the curve $O_{f(t)}^{\gamma}$ the cubic interpolation offset curve between d_1 and d_2. An example can be seen in Figure 2.14.

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. We call such segments fillets [SL16]. Informally, we can use fillets to round off corners. Corners are often the result of trimming curves at their intersection points. However, corners in

the geometries of CoolingGen negatively affect their structural stability. More rigorously, 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 2.15: 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 2.15, 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 $(s_1, s_2) \in I(O^{\gamma_1}_{\pm r}, O^{\gamma_2}_{\pm r})$.

Proof. Let $\tilde{\phi}(t) = r(\sin 2\pi t, \cos 2\pi t) + O^{\gamma_1}_r(s_1)$. 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 $\tilde{\phi}(t) = r(\sin 2\pi t, \cos 2\pi t) + O^{\gamma_2}_r(s_2)$. 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

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

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 $O^{\gamma_1}_r$ and $O^{\gamma_2}_r$.
2. Find their intersection parameter set $I(O^{\gamma_1}_r, O^{\gamma_2}_r)$.
3. For $(s,t) \in I(O^{\gamma_1}_r, O^{\gamma_2}_r)$, set $(x_0, y_0) := O^{\gamma_1}_r(s) = O^{\gamma_2}_r(t)$.

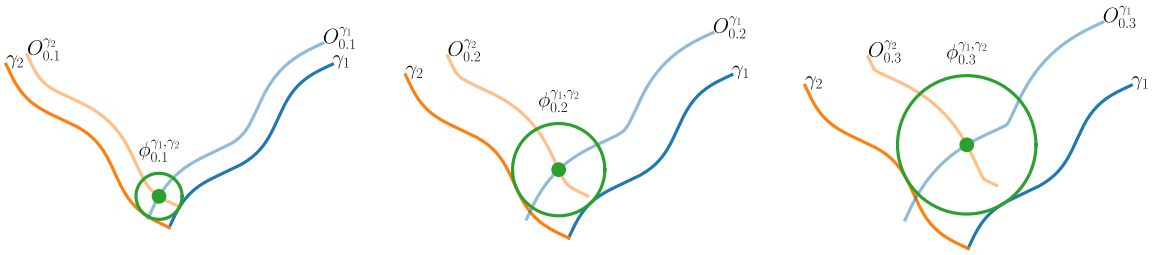


Figure 2.16: Construction of the fillet circle of different radii.

As can be seen in Figure 2.16, the curve $\phi^{\gamma_1,\gamma_2}_r$ 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 $\phi_r^{\gamma_1,\gamma_2}$.

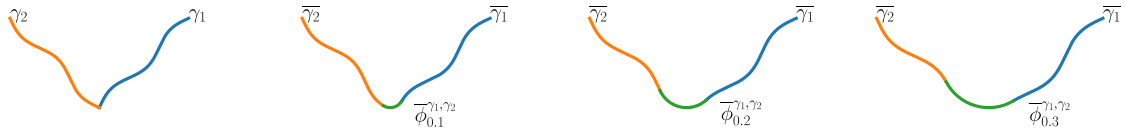


Figure 2.17: Trimmed curves and fillets

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 $\overline{MD_1}$ by a certain angle Θ, we can retrieve $\overline{\phi}_r^{\gamma_1,\gamma_2}$. This angle can be calculated by

$$\Theta = \arccos \frac{\langle D_1 - M, D_2 - M\rangle}{||D_1 - M||\ ||D_2 - M||},$$

where $\langle \cdot, \cdot \rangle$ is the standard scalar 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 & a Plane

In this section, we present an algorithm that intersects a plane and a surface in $\mathbb{R}^3$, which is largely motivated by [BK90]. This algorithm is used for the creation of channel turn and ejection slots. The intersection generally 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 surface-plane 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.

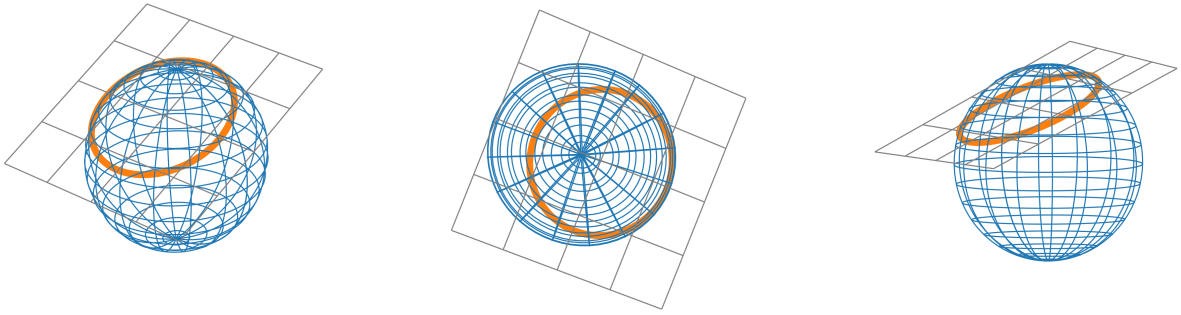


Figure 2.18: The intersection of a surface and a plane generally does not yield a curve. For our purposes, however, this type of intersection is the only case of interest.

Definition 2.3.17. The distance between a plane $P_{S,N}$ and a point $Q \in \mathbb{R}^3$ is defined as

$$D(P_{S,N}, Q) := \min_{V \in P_{S,N}} D(V, Q).$$

Theorem 2.3.18. The projection of a point $Q \in \mathbb{R}^3$ onto the plane $P_{S,N}$ is given by

$$Q_{\text{proj}} := Q - \langle Q - S, N \rangle N,$$

where $\langle \cdot, \cdot \rangle$ is the standard scalar product. This projection yields the unique solution

$$Q_{\text{proj}} = \arg\min_{V \in P_{S,N}} D(V, Q)$$

to the minimization problem in 2.3.17.

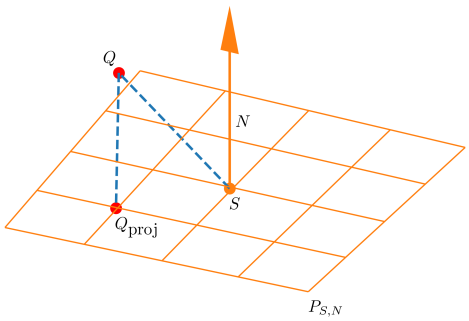


Figure 2.19: Projection of a point onto a plane as described by Theorem 2.3.18.

In linear algebra, it is often the case that a plane is defined by a support vector and two direction vectors. In CAD it is instead commonplace to use the pair (S, N) of support vector and normal vector. The normal vector is 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 helpful in

combination with the cross product.

Definition 2.3.19. 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, the set $\{A, B, C\}$ is a basis of $\mathbb{R}^3$. In other words, the cross product maps to a vector that is perpendicular to the two input vectors.

Theorem 2.3.20. 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 2.3.20 shows, the notation using the pair (S, N) is valid. Given a parametric surface $\beta(u, v)$ with $(u, v) \in [0, 1]^2$, we can construct a plane that is tangent to $\beta(u, v)$ for all $(u, v) \in [0, 1]^2$ by using the u and v gradients of β and taking their cross product. This yields the normal vector of the tangent plane. The support vector of the plane is just equal to the point $\beta(u, v)$. We make the following definition.

Definition 2.3.21. 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 $(u, v \in [0, 1]^2$ is given by the plane

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

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 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 $(u^{(0)}, v^{(0)})$ a priori, we can find points in the neighborhood of $(u^{(0)}, v^{(0)})$ that also lie in the plane.

To find a neighboring point, we find the line of intersection $L^{(0)}$ 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 intersection parameter pair $(u^{(0)}, v^{(0)})$, we can easily do this using the following theorem.

Theorem 2.3.22. 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. □

The tangent intersection line for the intersection parameters $(u^{(0)}, v^{(0)})$ is equal to

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

for $t \in \mathbb{R}$. Using an appropriate step size $\alpha \in \mathbb{R}$, we can project the point $L^{(0)}(\alpha) \in T_\beta(u^{(0)}, v^{(0)}) \cap P_{\beta(u^{(0)},v^{(0)}),N}$ onto the surface β using the methods from Section 2.3.1 to acquire a candidate for the point $\beta(u^{(1)}, v^{(1)})$. If this point lies outside of the plane $P_{S,N}$, we discard the point, choose a step size less than α and repeat the process. Numerically, we accept a candidate if it is within ϵ-distance of the plane $P_{\beta(u^{(0)},v^{(0)}),N}$ for a small value of $\epsilon > 0$. If $\beta(u^{(1)}, v^{(1)})$ has been found, we repeat this process to find $\beta(u^{(2)}, v^{(2)})$. To do this, we construct $L^{(1)}$ by intersecting $P_{S,N}$ and $T_\beta(u^{(1)}, v^{(1)})$ and project $L^{(1)}(\alpha)$ onto β. We can repeat this iteration step until $|\alpha| < \alpha_0$ for some $\alpha_0 > 0$. If this condition is satisfied, we return the list of the n parameters of intersection, which we can write as $L = \{(u^{(i)}, v^{(i)}) : i \in \{1, 2, ..., n\}\}$.

Algorithm 9 Surface Plane Intersection

Input
- $\alpha > \alpha_0$ — step size and minimum step size
- $\xi < 1$ — step size scaling factor
- ϵ — absolute tolerance

Output
- $\{(u^{(i)}, v^{(i)}) : i \in \{1, 2, ..., n\}\}$ — set of points on the intersection

procedure INTERSECTSURFACEPLANE($\beta, S, N, (u^{(0)}, v^{(0)})$)
- $L \leftarrow$ list containing only $(u^{(0)}, v^{(0)})$
- **while** $|\alpha| > |\alpha_0|$ **do**
  - $(u, v) \leftarrow$ last entry in L
  - $M \leftarrow \left(\frac{N_\beta(u,v) \times N}{||N_\beta(u,v) \times N||}\right)$
  - $P \leftarrow \beta(u, v) + \alpha M$
  - $u_{\text{proj}}, v_{\text{proj}} = \text{pointProjection}(\beta, P)$
  - **if** $D(\beta(u_{\text{proj}}, v_{\text{proj}}), P_{S,N}) > \epsilon$ **then** $\alpha \leftarrow \xi\alpha$
  - **else** append $(u_{\text{proj}}, v_{\text{proj}})$ to L
- **return** L

After successfully running Algorithm 9, we are left with n points which lie on the intersection curve. We interpolate between these points using NURBS [Pie97]. 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. This scheme proves useful for CoolingGen, because a start parameter pair $(u^{(0)}, v^{(0)})$ is practically always known in this context. Otherwise, a start point can be found using the ray marching technique, which was discussed in Section 2.3.2.

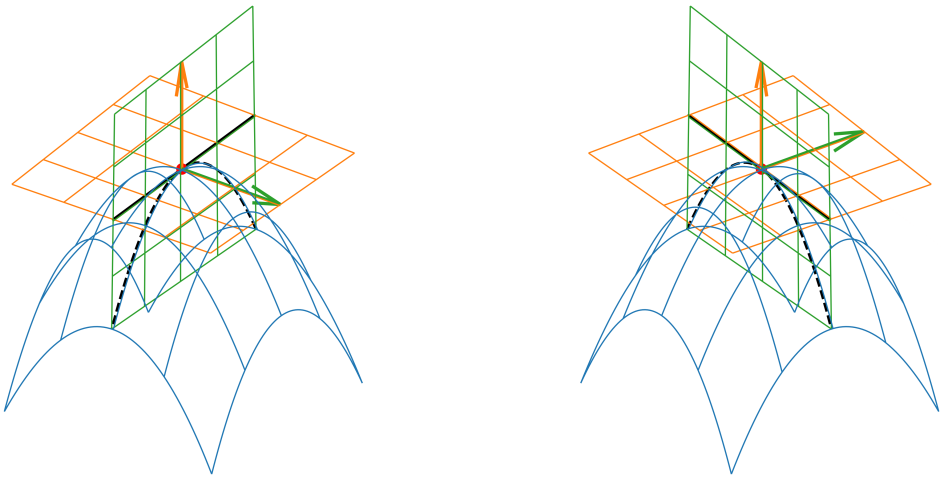


Figure 2.20: Intersection of a surface β (blue) with the plane $P_{S,N}$ (green). The intersection point $\beta(u^{(0)}, v^{(0)})$ is represented by the red dot. We intersect the tangent plane (orange) of β at $(u^{(0)}, v^{(0)})$ with $P_{S,N}$, yielding the line $L^{(0)}$ (black). Near the intersection point, $L^{(0)}$ and the true solution (black dashed curve) are close. The arrows represent the normal vectors.

2.3.7 Coons Patch

Given four parametric boundary curves $c_0(t)$, $c_1(t)$, $d_0(t)$ and $d_1(t)$ for $t \in [0,1]$ with $c_0(0) = d_0(0)$, $c_0(1) = d_1(0)$, $c_1(0) = d_0(1)$ and $c_1(1) = d_1(1)$ we can construct an intermediate parametric surface $\mathfrak{C}$ by using linear interpolation on pairs c_0, c_1 and d_0, d_1 and bilinear interpolation the corner points $c_0(0), c_0(1), c_1(0), c_1(1)$ using bilinear blending, in other words

$$\begin{aligned}\mathfrak{C}(u,v) = &(1-v)\,c_0(u) + v\,c_1(u)\\ &+ (1-u)\,d_0(v) + u\,d_1(v)\\ &- (1-u)(1-v)\,c_0(0) - u(1-v)\,c_0(1)(1) - (1-u)v\,c_1(0) - uv\,c_1(1),\end{aligned}$$

where $(u,v) \in [0,1]^2$. Such a surface is called a Coons patch [Coo67].

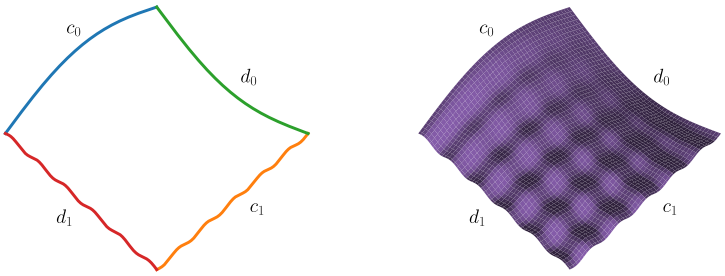


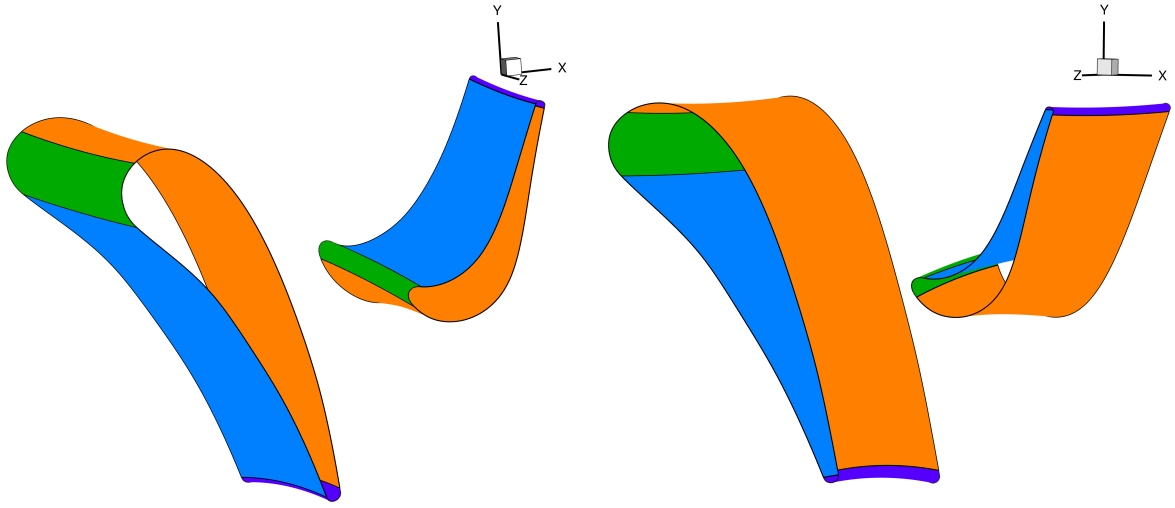
Figure 2.21: Example of a Coons patch $\mathfrak{C}$.

3 Jet Engine Specific Methods

CoolingGen creates geometries which lie inside a blade or vane surface $B(u, v)$, which is an output of BladeGen, a tool developed by the DLR specifically for the design of blade and vane 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 convenient to describe these geometries in cylindrical coordinates. The transformation from Cartesian coordinates to cylindrical coordinates in this case is given by the map

$$Z : (x, y, z) \mapsto (x, r, \theta),$$

where $\theta = \text{arctan2}(y, z)$ is the circumferential angle and $\sqrt{y^2 + z^2}$ is the distance from the turbine hub. This transformation 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 vane and the right surface represents a rotor blade. The blade has a named partitioning into the leading edge (green), the pressure side (blue), the suction side (orange) and the trailing edge (purple).

A distinction is made between two kinds of surfaces: rotor blades and stator vanes. While the rotor blades rotate around the hub, the stator vanes have fixed positions. Furthermore, the blade or vane 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.

3.1 The S2 Stream Surface & Stream Surface Coordinates

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 or vanes, 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 (or vane) sections, whereas their counterpart, the S_2 surfaces, which are also known as meridional surfaces, describe the flow in the axial direction [Wu52].

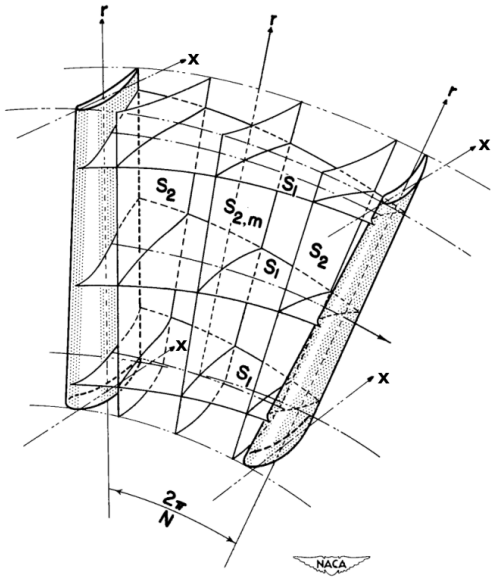


Figure 3.2: Example S_1 and S_2 surfaces of a turbomachine [Wu52].

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

$$m(u, v) = \int_0^u \sqrt{x(t, v)^2 + r(t, v)^2}\, \mathrm{dt}$$

is the v-isoparametric arc length of the stream surface. To design the blades (and vanes) 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 [Wu52]. Each stream line described by $S_{2,m}$ can then be written as v-isoparametric curve $S_{2,m}(\cdot, v)$.

By intersecting the transformed u-isoparametric curves $Z(B(u, \cdot))$ of the blade (or vane) with each such stream line $S_{2,m}(\cdot, v)$ in their first two arguments (x, r), we can construct a curve

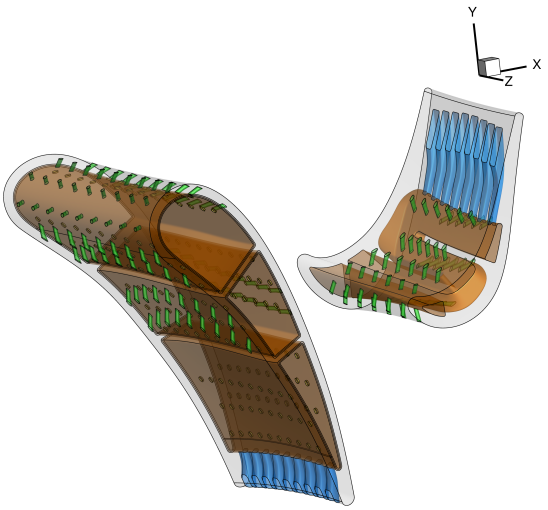
$$B^{(v)}_{\text{stream}}(u) = (m, r\theta),$$

where m is the intersection arc length of $S_{2,m}(\cdot, v)$, r is the intersection distance from the hub and θ is the intersection circumferential angle. This coordinate system with respect to arc length m and radius-angle product $r\theta$ is a so called stream surface coordinate system. Given v and the stream surface, each point $(m, r\theta)$ can bijectively be mapped back onto the respective Cartesian coordinates (x, y, z).

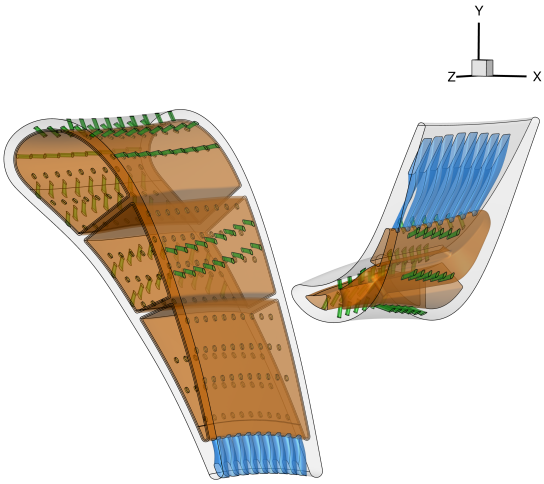
Since this coordinate transformation produces a set of planar curves for $v_i \in [0,1]$, where $i \in \{1,2,...,N\}$, we can use two-dimensional operations on such a set of stream surface curves instead of using three-dimensional operations directly on the Cartesian represented $B(u,v)$. This yields a significant performance boost. The resulting stream surface curves can then be transformed back to the Cartesian coordinate system. Using surface skinning, which is presented in [Pie97], we can interpolate between families of stream surface curves to find a surface. This approach will be used in the construction of the internal geometries.

4 Results

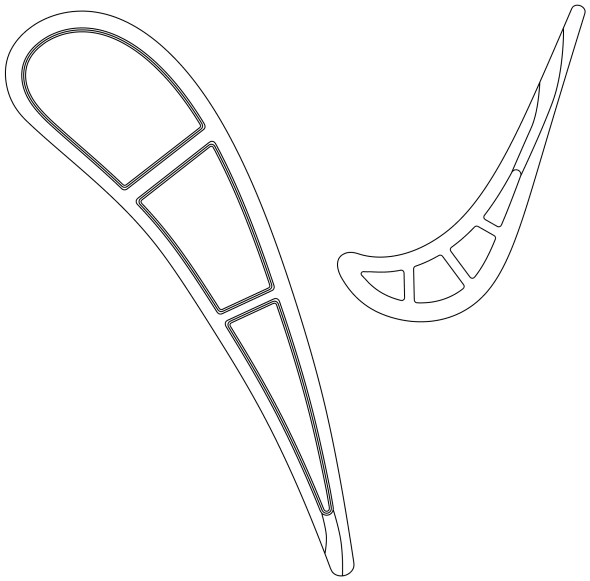
In this chapter, we present the construction of different blade and vane cooling geometries using CoolingGen. For this purpose, we require the methods presented in Chapter 2 and Chapter 3. As mentioned before, CoolingGen is capable of producing four distinct types of geometries: channels, film cooling holes, impingement inserts and trailing edge slots. These four types are supported on rotor blades as well as on stator vanes.



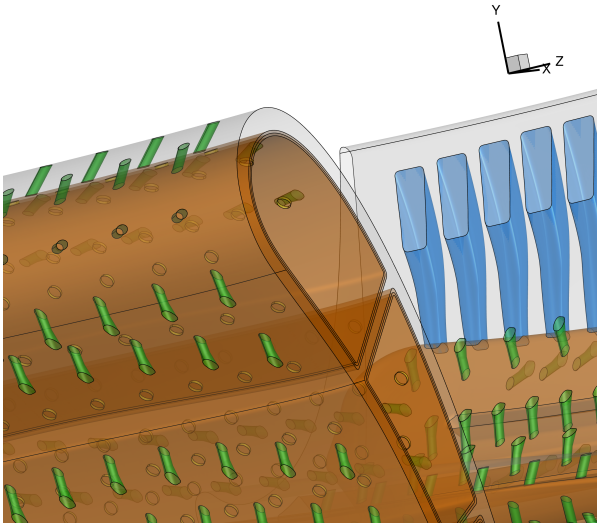
(a) View on pressure side.



(b) View on suction side.



(c) Profile view.



(d) Detailed view.

Figure 4.1: Like Figure 3.1, but equipped with cooling geometries. The outer orange surfaces are channels. The green surfaces are film cooling holes. The inner orange surfaces are impingement inserts. The short yellow cylindrical surfaces represent the holes inside the impingement inserts. In the detailed view, we can see the impingement insert of the leading edge channel of the stator and the slots on the pressure side of the rotor.

4.1 Channels

In the construction of channels with CoolingGen, 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.

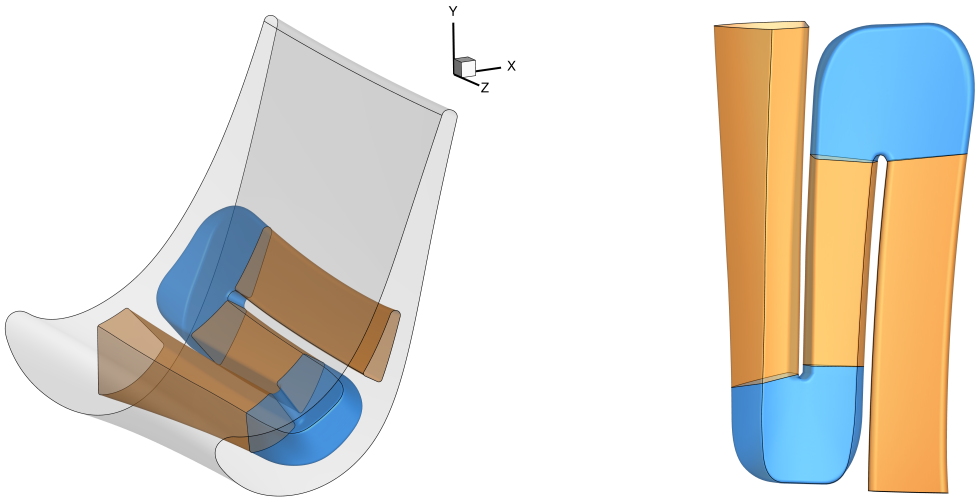
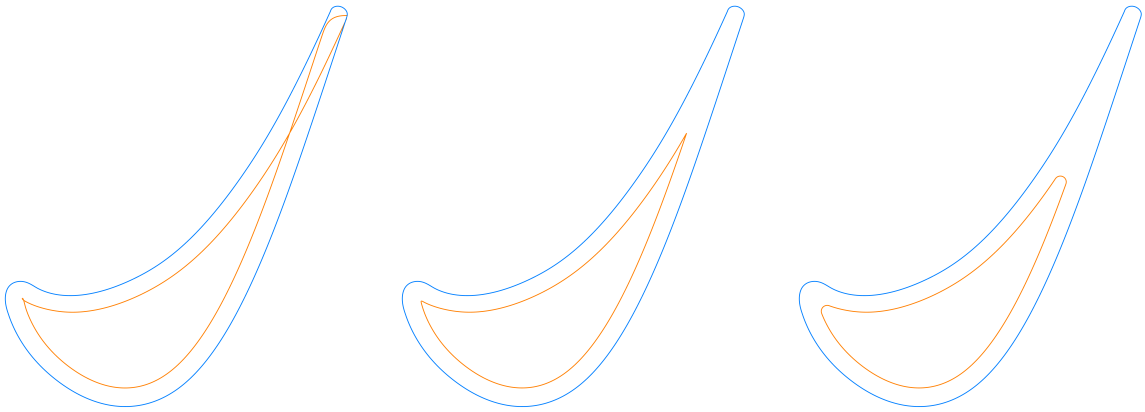


Figure 4.2: Distinguishing chambers (orange) and turns (blue) in a channel.

4.1.1 Chambers

The construction of chambers of a cooling channel is presented in this section.



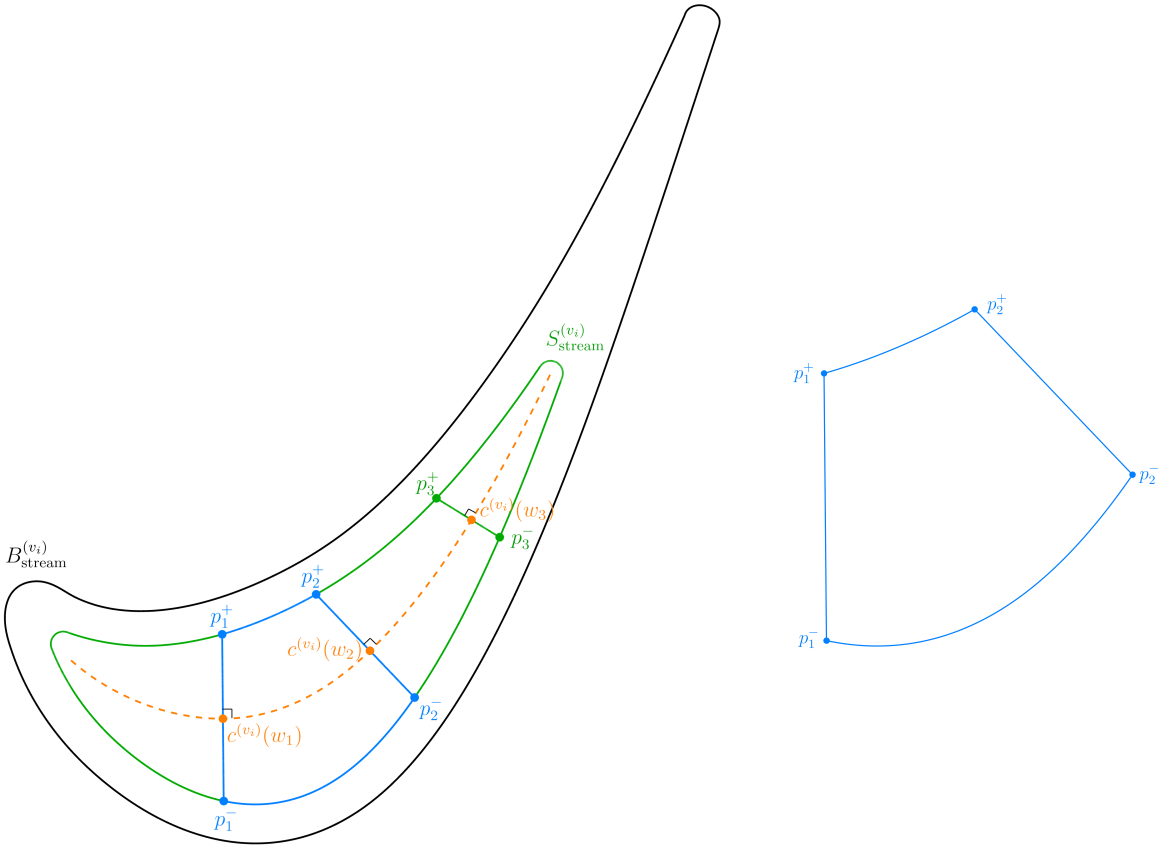
(a) Offsetting. (b) Trimming. (c) Adding fillets.

Figure 4.3: 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 3.1 to a fixed number N of v-isoparametric curves of the blade (or vane) to calculate the profiles $B^{(v_i)}_{\text{stream}}(u)$ for $v_i \in [0, 1]$, $i \in \{1, ..., N\}$. These profiles are then offset by an input parameter. The profile is

automatically trimmed adequately (as described in Section 2.3.4) and fillets are added to round off the emerging corners. The fillet radius is also declared as an input. We refer to the resulting profile as shrunk profile $S^{(v_i)}_{\text{stream}}(u)$. This process is depicted in Figure 4.3. The subdivision into leading edge, pressure side, suction side and trailing edge is kept by using the fillet boundaries as transition points.

Next, we calculate the camber curve $c^{(v_i)}$ of each shrunk profile $S^{(v_i)}_{\text{stream}}$. The camber curve 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 the 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 set of parameters $w_j \in [0,1]$ representing the positions of each wall as parameter of 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 $\omega_j \in (0, \pi)$.



(a) Partitioning the profile.

(b) Resulting partition.

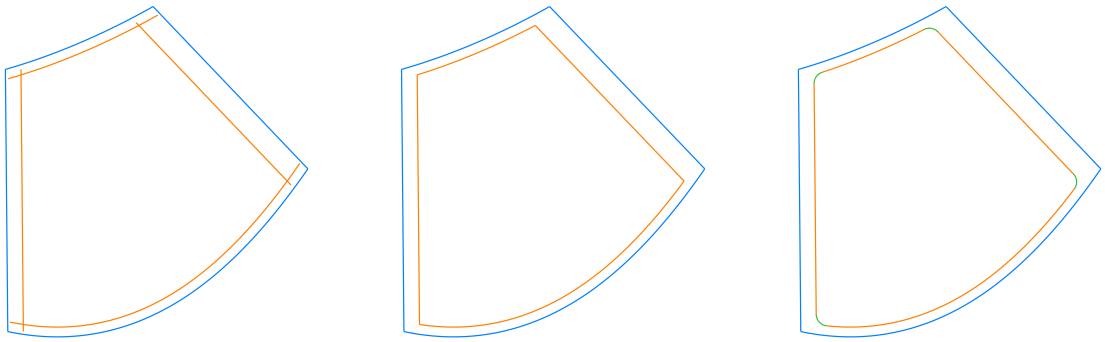
Figure 4.4: Profile partitioning along rays that emerge from the camber curve. The camber curve is represented by the dashed line. In this example, we create four chambers. The walls are represented by the line segments $\overline{p_j^+ p_j^-}$ for $j \in \{1, 2, 3\}$.

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_{ω_j} is a rotation matrix like presented in Equation (2.12). Using Algorithm 6, we can find the respective intersection points p_j^+ and p_j^- of the rays and the shrunk profile $S_{\text{stream}}^{(v_i)}$. The line segment $\overline{p_j^+ p_j^-}$ now represents the j-th wall. This process is repeated for all $j \in \{1, ..., N_C - 1\}$.

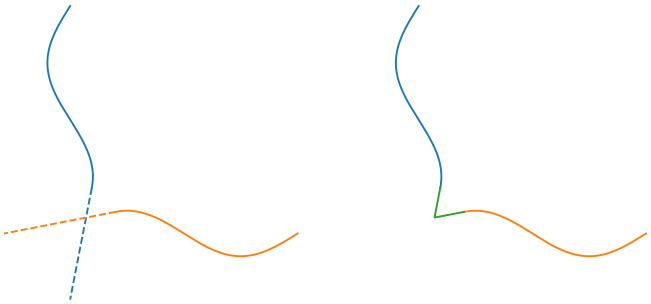
Next, the blade (or vane) is partitioned at the intersection points p_j^+ and p_j^- for all $j \in \{1, ..., N_C - 1\}$. For each such partitioning, we are left with four chamber-delimiting curves per chamber profile. This procedure is exemplified in Figure 4.4. If there is only one chamber, the a priori partitioning scheme (leading edge, suction side, pressure side and trailing edge) is kept. In case of multiple chambers, the leading and trailing chambers have to be treated differently, because in this case each of them are delimited by only one wall. In this case, we use the leading edge partition or trailing edge partition in place of a wall.



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

Figure 4.5: Construction of a chamber profile.

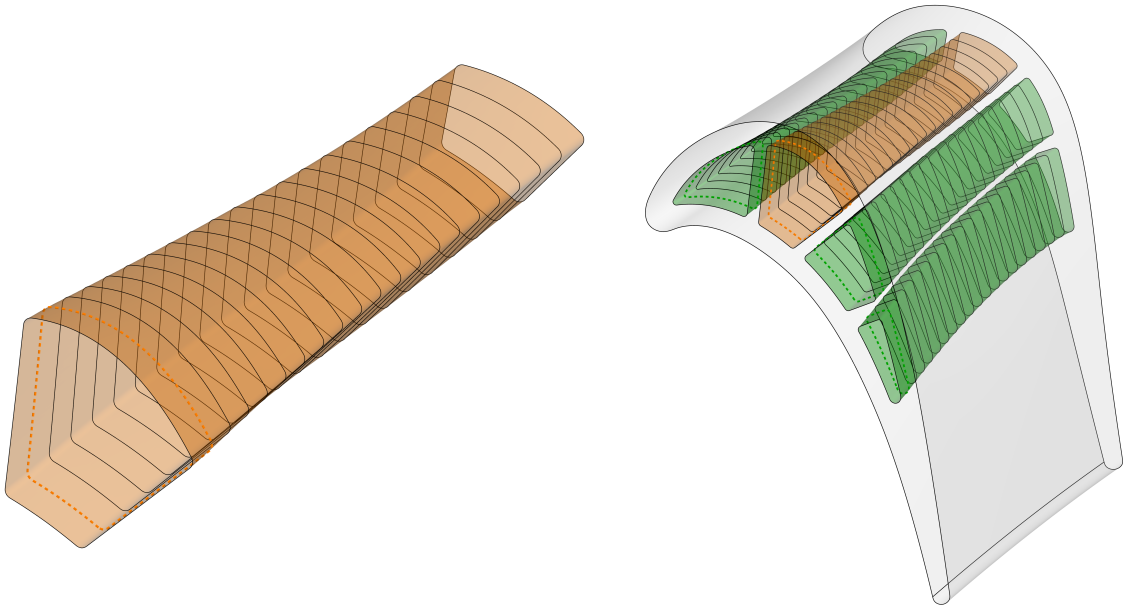
After partitioning, we offset the chamber-delimiting curves. Each chamber-delimiting curve is offset differently. In the wall input, the thickness of the wall in each direction is specified as an input. In fact, this input specification is done using two B-spline curves. For each wall, the user of CoolingGen specifies two sets control points $\{(v_k^1, d_k^1) : k \in \{1, ..., K\}\}$ and $\{(v_j^2, d_j^2) : j \in \{1, ..., J\}\}$, where $v_k^1, v_j^2 \in [0, 1]$ are surface parameters and d_k^1 are the respective thicknesses of the leading and trailing walls, respectively. Similarly, the suction and pressure side thicknesses for each chamber are defined by two B-spline curves.



(a) Linear continuation. (b) Intersection of (a).

Figure 4.6: Intersection of the linear continuation of two curves.

The leading and trailing partition offsets are cubic interpolation offset curves between suction and pressure side offsets to guarantee continuity and differentiability of the chamber profile at the leading and trailing edge transitions, see Figure 2.14. After offsetting, the chamber-delimiting curve end points do not necessarily intersect. Therefore, 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 end points and B represent the (positive at the end point of the curve or negative at the start point of the curve) gradient at the curve boundary points. These rays can be understood as the linear continuation of the chamber-delimiting curves (see Figure 4.6).



(a) A single chamber.

(b) Four chambers.

Figure 4.7: Skinned chamber surfaces.

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 at their end points. We apply the fillet curve method presented in Section 2.3.5 between neighboring chamber-delimiting curves. The offsetting, trimming and fillet creation steps can be seen in Figure 4.5.

As described in Section 3.1, we can use the inverse coordinate transformation from the $(m, r\theta)$ stream surface coordinate system to the Cartesian coordinate system and use surface skinning to create chamber surfaces. Figure 4.7 shows an exemplary result of this procedure.

4.1.2 Turns

As mentioned before, a turn lets coolant flow between two chambers. Notice that the presented method, unlike the construction of chambers, takes place in Cartesian coordinates.

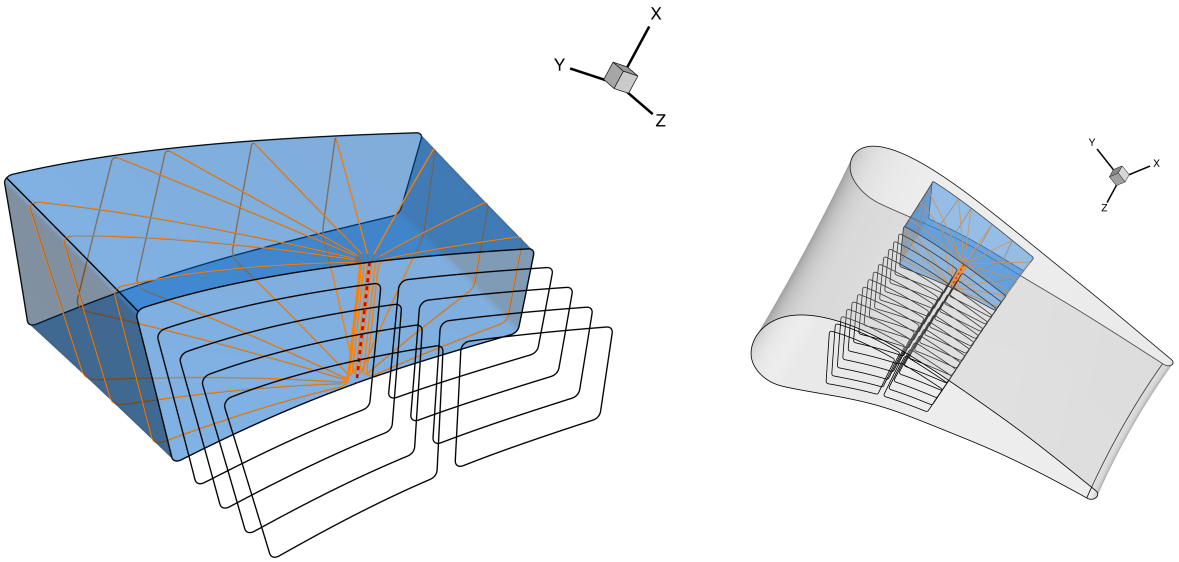


Figure 4.8: Construction of a turn. The red dashed line represents the axis around which the turn is constructed. The black curves are the chamber profiles which were calculated in Section 4.1.1. The blue surface is the combination of two neighboring chambers. The orange curves are intersections of planes that contain the turn axis and the combined chamber.

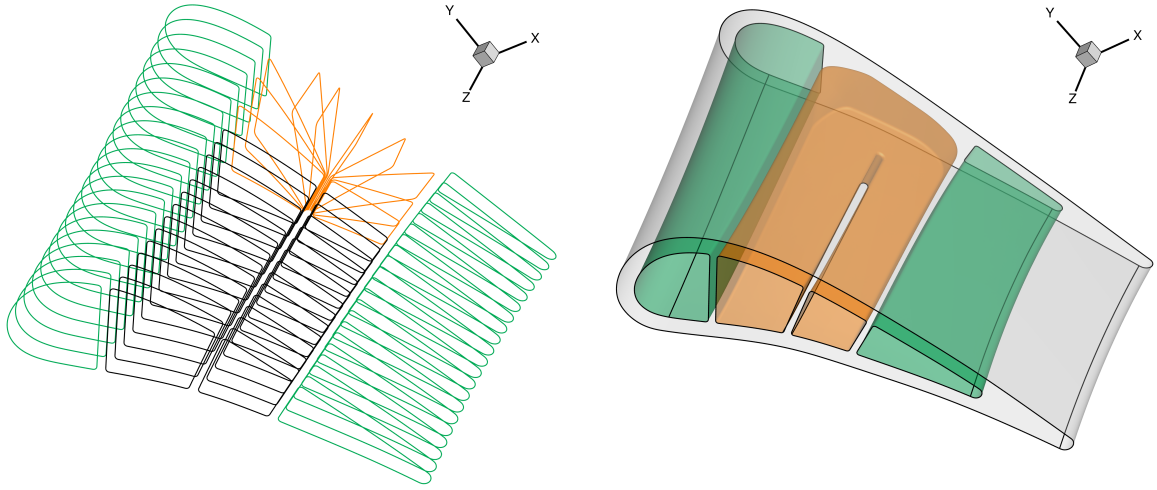
First, we construct the combined chamber, for which the two chambers that shall be connected by the turn are specified. The intermediate surface partitions are removed, and the suction and pressure sides of the two chambers are connected differentiably by inserting control points. For this purpose, the user specifies a lower bound v_L and an upper bound v_U, between which the combined chamber (and therefore the turn) is constructed.

Next, the turn axis is constructed. The wall input of the intermediate wall (see Section 4.1.1) has a Cartesian representation as a surface. The v_L-isoparametric curve end points of this wall surface are connected with a straight line. This yields the turn axis. Using the turn axis, we can construct a set of planes which intersect the turn axis. A set of angles (relative to the intermediate wall) are specified as an input. The combined chamber is partitioned into a suction and a pressure side.

We use Algorithm 9 to find the intersection of these planes and the suction and pressure side of the combined chamber, yielding two curves for each plane. The intersection curves start at the turn axis and end at the suction or pressure side boundary, respectively. The start points of these curves do not intersect, but both start points lie on the turn axis. We connect these points using a straight line, which lies inside the turn axis. If the end points do not intersect, we also connect them using a straight line.

Since all of these curves (including the straight lines, which are represented as curves within CoolingGen) are planar by construction, we can use two-dimensional operations to offset, trim and add fillets. Since CoolingGen performs two-dimensional operations on the (x, y) plane, the curves have to be rotated.

To ensure a differentiable connection between the chambers, the axis-aligned curve is offset. The size of the offset is given by the cubic interpolation in terms of the angle between the wall thickness on the leading and on the trailing side, similarly to Figure 2.14. Similar to the procedure in 4.1.1, the pressure- and suction-sided curves are then trimmed accordingly and fillets are added between them.



(a) Profile curves. (b) Surfaces.

Figure 4.9: Three cooling channels.

After rotating the planar curves back from the (x, y)-plane onto their original plane, we once again use skinning to combine the planar curves to turn surfaces. However, these turn surfaces do not contain the chambers. As mentioned at the beginning of this chapter, the coherent combination of turns and chambers is called channel. To construct a channel from turns and chambers, we find the profiles and planar curves that belong to the channel. We align their start points and direction by shifting the parameter space in the following way.

Given a parametric curve $\gamma(t)$ with $t \in [0, 1]$, we can align the curve by applying the map

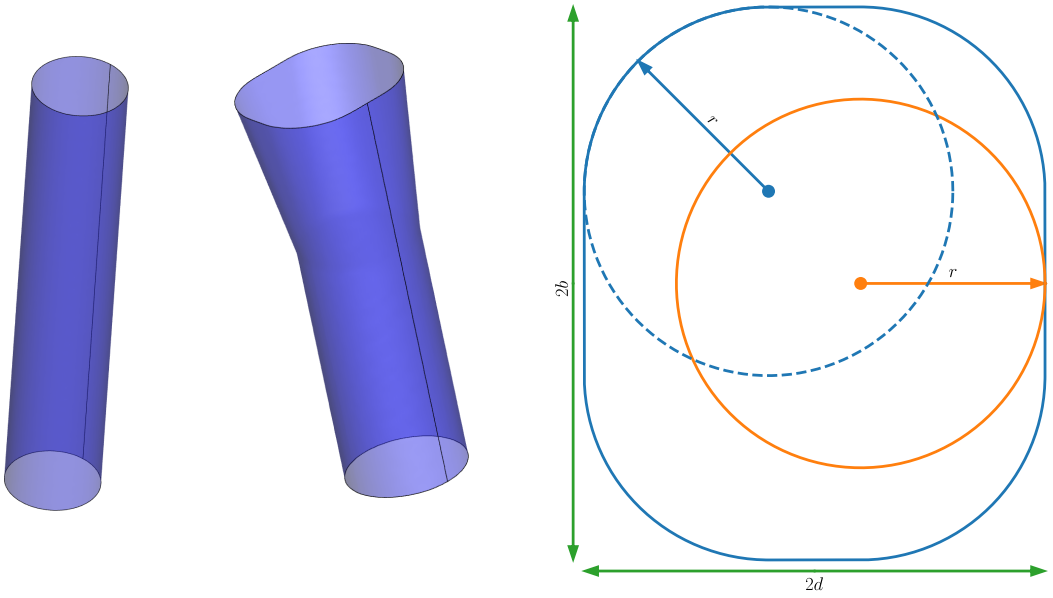
$$\Phi(s, \sigma, t) : [0, 1] \to [0, 1] \quad t \mapsto (s + \sigma t + 1) \bmod 1,$$

to the input parameter t, where $s \in [0, 1]$ is the parametric shift, $\sigma \in \{1, -1\}$ is the curve direction, and mod is the modulus operator. Using this map and Algorithm 5, we can align two curves at a point.

The reason why this is necessary may become apparent through Figure 4.2. The black curve on the channel represents the $u = 0$ isoparametric curve of the channel surface. As can be seen, this curve switches position throughout the turn. In the first chamber, the curve lies on the leading edge side, then on the trailing edge side and then once again on the leading edge side. The turns show the same behavior. In the turn on the leading edge side, the curve lies on the far side of the turn, whereas in the turn on the trailing edge side, the curve lies on the near side of the turn.

4.2 Film Cooling Holes

In this section, the construction of film cooling holes is presented.



(a) Cylindrical. (b) Fan shaped. (c) f_{circular} (orange) and $f_{\text{fan shaped}}$ (blue).

Figure 4.10: Film cooling holes and their delimiting curves.

The general shape of a film cooling hole is described using two delimiting curves which are closed and a transitional v-parameter, which we will call $v_{\text{transition}} \in [0,1]$. CoolingGen currently supports two types of delimiting parametric curves, but can be extended to provide a multitude of such curves. The currently supported curves are the circular curve

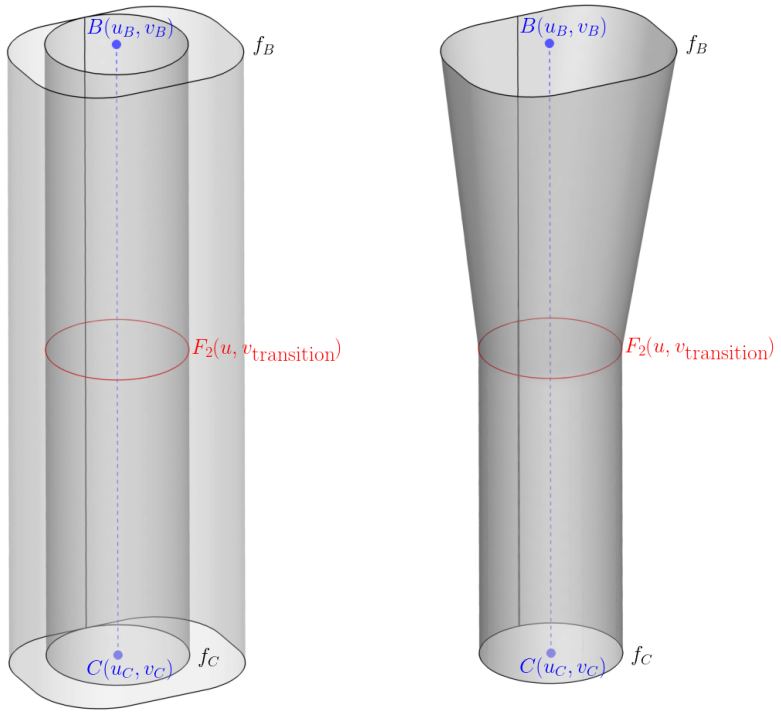
$$f_{\text{circular}}(t) = \begin{pmatrix} r\cos 2\pi t \\ r\sin 2\pi t \end{pmatrix},$$

for a given radius r, and the fan shaped curve

$$f_{\text{fan shaped}}(t) = \begin{cases} \begin{pmatrix} r \\ 0 \end{pmatrix} & \text{if } t \in \{0,1\}, \\ \begin{pmatrix} 0 \\ b \end{pmatrix} & \text{if } t = \frac{1}{4}, \\ \begin{pmatrix} r-2d \\ 0 \end{pmatrix} & \text{if } t = \frac{1}{2}, \\ \begin{pmatrix} 0 \\ -b \end{pmatrix} & \text{if } t = \frac{3}{4}, \end{cases} \quad \text{or} \quad \begin{cases} \begin{pmatrix} r\cos 2\pi t \\ r\sin 2\pi t + (b-r) \end{pmatrix} & \text{if } t \in (0, \frac{1}{4}), \\ \begin{pmatrix} r\cos 2\pi t - 2(d-r) \\ r\sin 2\pi t + (b-r) \end{pmatrix} & \text{if } t \in (\frac{1}{4}, \frac{1}{2}), \\ \begin{pmatrix} r\cos 2\pi t - 2(d-r) \\ r\sin 2\pi t - (b-r) \end{pmatrix} & \text{if } t \in (\frac{1}{2}, \frac{3}{4}), \\ \begin{pmatrix} r\cos 2\pi t \\ r\sin 2\pi t - (b-r) \end{pmatrix} & \text{if } t \in (\frac{3}{4}, 1), \end{cases}$$

where $r > 0$ is the corner radius, $b > 0$ is the height, $d > 0$ the width and $t \in [0,1]$. Both of these

curves can be seen in Figure 4.10.



(a) Extrusion surfaces F_1 and F_2. (b) Film cooling hole $F^{(v_{\text{transition}})}$.

Figure 4.11: Construction of a film cooling hole.

To construct a surface from a delimiting curve f, we canonically embed the two-dimensional curves into three-dimensional space using the map

$$E : \mathbb{R}^2 \to \mathbb{R}^3, \quad (x, y) \mapsto (x, y, 0).$$

We also require a line segment L_{P_B,P_C} between two points $P_B, P_C \in \mathbb{R}^3$. The curve $E(f)$ lies in the (x, y)-plane that has the normal vector $N_{(x,y)} = (0, 0, 1)$. We construct the rotation matrix $R \in \mathbb{R}^{3\times3}$ that rotates $N_{(x,y)}$ onto the direction $(P_C - P_B)$ of the line segment L_{P_B,P_C} by finding the angle between $(P_C - P_B)$ and $N_{(x,y)}$. The rotation axis is given by the cross product $(P_C - P_B) \times N_{(x,y)}$. Using the rotation matrix R, we can rotate $E(f)$ to calculate $RE(f)$. We can translate $RE(f)$ to the start and end points of L_{P_B,P_C} to calculate two curves

$$f_B(t) := P_B + RE(f(t)) \quad \text{and} \quad f_C(t) = P_C + RE(f(t)),$$

for $t \in [0, 1]$. For all $v \in [0, 1]$, we can linearly interpolate between $f_B(u)$ and $f_C(u)$ for each $u \in [0, 1]$ to find the surface

$$F(u, v) = (1 - v)\, f_B(u) + v\, f_C(u).$$

We call the surface $F(u, v)$ the extrusion of f along L_{P_B,P_C}. Two extrusion surfaces can be seen in Figure 4.11.

Setting a value $v_{\text{transition}} \in [0, 1]$, we can combine two extrusion surfaces of different curves f_1

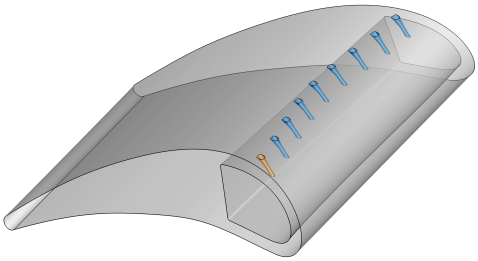
and f_2 along the same line segment. This will yield the characteristic shape of the film cooling holes. For this purpose, let $F_1(u,v)$ be the extrusion of f_1 along L_{P_B,P_C} and let $F_2(u,v)$ be the extrusion of f_2 along L_{P_B,P_C}. Such a combined surface can be seen in Figure 4.11. We can now write the combined surface of F_1 and F_2 along L_{P_B,P_C} as the linear interpolation

$$F^{(v_{\text{transition}})}(u,v) = \begin{cases} \frac{v_{\text{transition}}-v}{v_{\text{transition}}} F_1(u,0) + \frac{v}{v_{\text{transition}}} F_2(u, v_{\text{transition}}) & \text{if } v \leq v_{\text{transition}}, \\ F_2(u,v) & \text{else.} \end{cases}$$

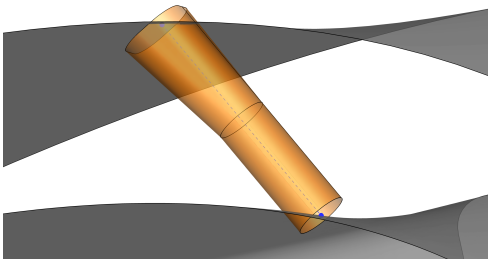
The surface $F^{(v_{\text{transition}})}(u,v)$ is a basic representation of a film cooling hole. There are currently two supported types of hole surfaces in CoolingGen. The first type is the cylindrical hole, where $f_1 = f_2 = f_{\text{circular}}$. The second type of film cooling hole is called fan shaped hole, where $f_1 = f_{\text{fan shaped}}$ and $f_2 = f_{\text{circular}}$. The input radii r are the same for each pair of curves. Both of these types can be seen in Figure 4.10.

As mentioned before, film cooling holes transport the fluid from the inside of a channel to the outside of the blade or vane. Therefore it is necessary to connect $F^{(v_{\text{transition}})}(u,v)$ with the blade or vane surface and the corresponding channel surface. Our aim is to align the f_1 side of $F^{(v_{\text{transition}})}$ to the blade or vane surface and the f_2 side of $F^{(v_{\text{transition}})}$ to one of the channel surfaces.

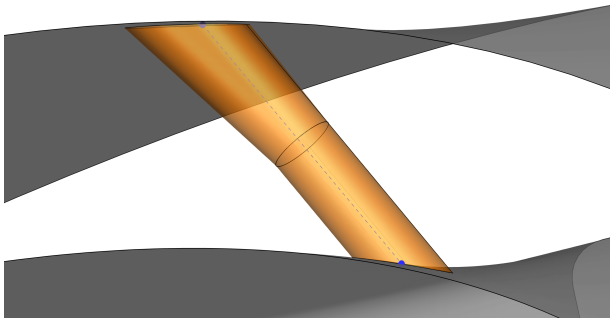
In CoolingGen, we define the starting point $P_B := B(u_B, v_B)$ of the line segment L_{P_B,P_C} in terms of the blade or vane surface $B(u,v)$ by using the input parameters $(u_B, v_B) \in [0,1]^2$. The input also specifies a direction D in which the film cooling hole should be pointed (via the specification of two angles relative to the blade or vane surface). The end point P_C of the line segment is then found by using the surface ray marching method presented in Section 2.3.2 for the ray $R_{P_B,D}$ for each channel. If multiple channels intersect $R_{P_B,D}$, then the channel surface $C(u,v)$, which has the minimum distance from $B(u_B, v_B)$, is chosen. We set the end point P_C of the line segment L_{P_B,P_C} to be equal to the chosen intersection point $C(u_C, v_C)$ for $(u_B, v_B) \in [0,1]^2$. We can then calculate the surface $F^{(v_{\text{transition}})}(u,v)$ along L_{P_B,P_C} as already explained.



(a) Channel with film cooling holes.



(b) Film cooling hole surface $F^{(v_{\text{transition}})}$.



(c) Ray intersected film cooling hole surface Φ.

Figure 4.12: Intersecting a film cooling hole using ray intersection.

Notice that using this process, the start and end points of P_B and P_C are now aligned to the corresponding surfaces $B(u, v)$ and $C(u, v)$, but the two surface boundaries $F^{(v_{\text{transition}})}(u, 0)$ and $F^{(v_{\text{transition}})}(u, 1)$ do not align with the surfaces $B(u, v)$ and $C(u, v)$, respectively. Notice also that the basic shape of the surface $F^{(v_{\text{transition}})}(u, v)$ has the following property regarding rays:

$$F^{(v_{\text{transition}})}([0,1]^2) \subset \bigcup_{v_F \in \{0,1\}} \bigcup_{u_F \in [0,1]} R_{P,Q-P}([0,\infty)),$$

where $P := F^{(v_{\text{transition}})}(u_F, v_{\text{transition}})$ and $Q := F^{(v_{\text{transition}})}(u_F, v_F)$. In other words, the film cooling hole is made up out of line segments or, equivalently, subsets of rays, the support and direction vectors of which we already know.

This observation gives rise to a solution to our boundary problem. We can use the ray marching algorithm from Section 2.3.2 to find ray intersections of all such $R_{P,Q-P}$ and $B(u, v)$ by setting $v_F = 0$. This yields a delimiting curve $B_d(u)$, which marks one boundary of the film cooling hole and lies completely within $B(u, v)$. By setting $v_F = 1$, we find the ray intersections of all such $R_{P,Q-P}$ and $C(u, v)$, yielding a delimiting curve $C_d(u)$ which marks the other boundary of the film cooling hole and lies completely within $C(u, v)$.

We are left with three important curves. The blade or vane delimiting curve $B_d(u)$, the transition isocurve $F^{(v_{\text{transition}})}(u, v_{\text{transition}})$ and the channel delimiting curve $C_d(u)$. We can linearly

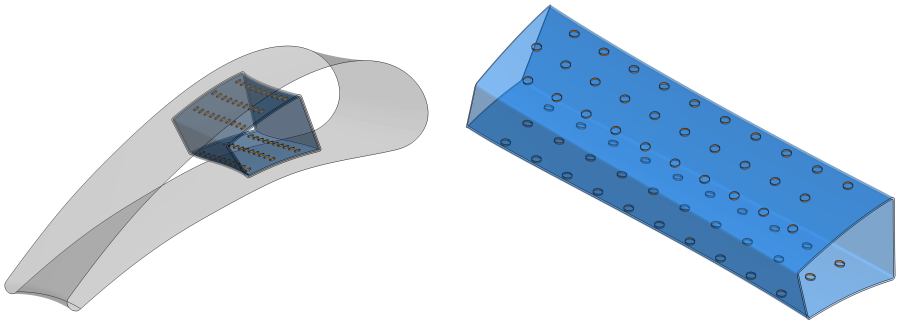
interpolate between these three curves. The intersected film cooling hole surface is then equal to

$$\Phi(u,v) = \begin{cases} \frac{v_{\text{transition}}-v}{v_{\text{transition}}} B_d(u) + \frac{v}{v_{\text{transition}}} F^{(v_{\text{transition}})}(u, v_{\text{transition}}) & \text{if } v \leq v_{\text{transition}}, \\ \frac{1-v}{1-v_{\text{transition}}} F^{(v_{\text{transition}})}(u, v_{\text{transition}}) + \frac{v-v_{\text{transition}}}{1-v_{\text{transition}}} C_d(u) & \text{else.} \end{cases}$$

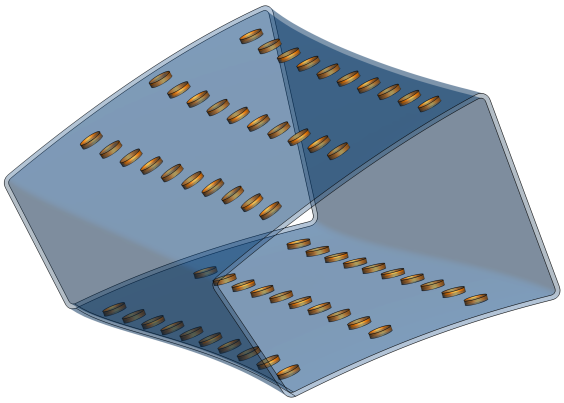
Using this procedure, we can efficiently create multiple film cooling holes. In CoolingGen, a distribution function is used to create a row of multiple film cooling holes at once that share the same direction D, radius r and shape parameters b, d, but differ in position. This distribution function takes the number of holes $H \in \mathbb{N}$, a parameter $u_B \in [0,1]$ and an interval $[v_L, v_U] \subset [0,1]$ as an input and distributes H film cooling holes on $B(u_B, [v_L, v_U])$. Multiple of these rows are usually specified.

4.3 Impingement Inserts

To construct an impingement insert, we use two surfaces to model the outer and the inner surface of the sheet that reside within the channel, respectively. The holes in this sheet are represented by cylindrical surfaces. Notice that the creation of impingement inserts is not compatible with the creation of channels with turns.



(a) Radial view with channel and blade. (b) Suction-sided view.

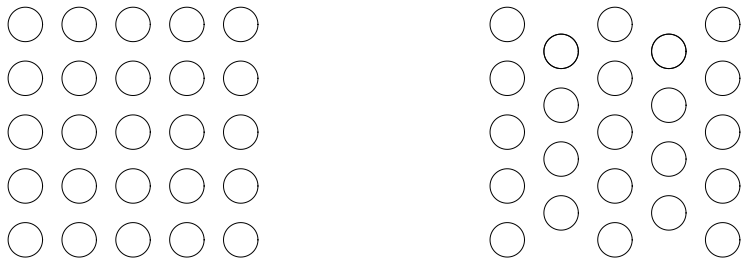


(c) Radial view.

Figure 4.13: An impingement insert with staggered holes on the suction side and aligned holes on the pressure side.

The creation of the inner and outer surfaces relies on the same principles as the creation of chambers, which was discussed in Section 4.1.1. Since we already calculated the $(m, r\theta)$ profiles of the chambers, we only need to offset, trim, add fillets and use skinning to make these surfaces. For each impingement insert, an inner distance $i > 0$, outer distance $o > 0$ and a thickness $d > 0$ is specified.

The profile partitions that run parallel to the blade or vane are offset by o, whereas the profile partitions that lie between chambers are offset by i. The new profile partitions are trimmed and fillets are added. This yields the outer impingement insert profile. The outer impingement insert profiles are then offset by d to calculate the inner impingement insert profiles. After using surface skinning, we are left with two surfaces that represent the impingement insert, an example of which can be seen in Figure 4.13.



(a) Aligned pattern.

(b) Staggered pattern.

Figure 4.14: Aligned and staggered pattern of impingement insert holes.

Using the methods from 4.2, we can construct cylindrical holes between the outer and inner surface of the impingement insert. Instead of using a distribution function for positioning, we instead specify two intervals $[u_L, u_U]$ and $[v_L, v_U]$ on the outer impingement insert surface and the number of rows N and columns M of film cooling holes per array, to place (up to) NM holes between the outer and inner impingement insert surface, inside the specified intervals. We also specify the type of hole distribution, which can be either aligned or staggered. These two types can be seen in Figure 4.14 and Figure 4.13.

4.4 Ejection Slots

Ejection slots transport the coolant from the inside of the trailing edge channel to the pressure side of the blade or vane. A slot is partially defined by two delimiting surfaces, the inlet surface $I(u, v)$ and the outlet surface $O(u, v)$ for $u, v \in [0, 1]$. The inlet surface $I(u, v)$ is a subsurface of the channel surface $C(u, v)$ that lies closest to the trailing edge. The outlet surface $O(u, v)$ is a subsurface of the blade or vane surface $B(u, v)$.

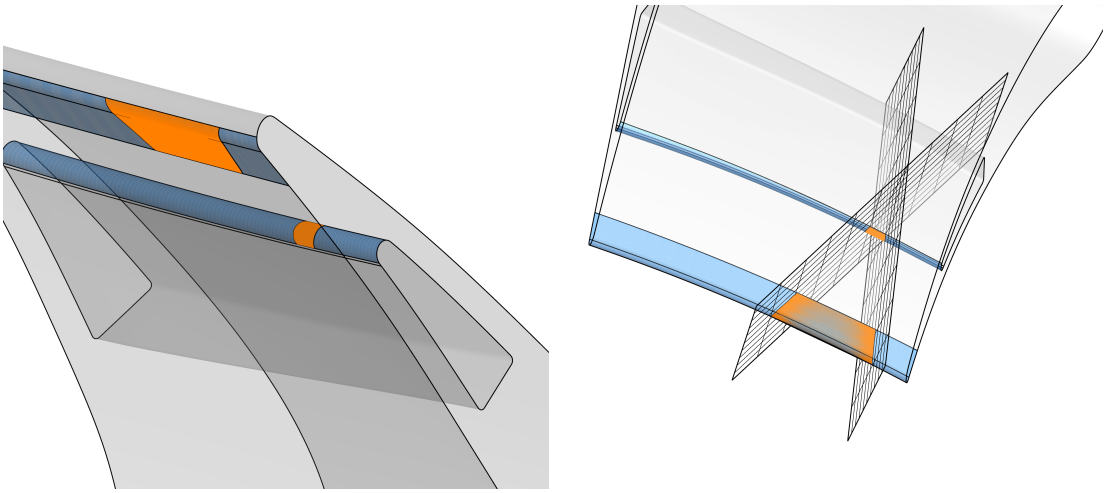


Figure 4.15: The inlet surface $I(u, v)$ on the trailing edge channel $C(u, v)$ and the outlet surface $O(u, v)$ on the blade $B(u, v)$ are calculated using plane-surface intersection.

Given the input lower bounds u_B^L and u_C^L and the input upper bounds u_B^H and u_C^H of $B(u, v)$ and $C(u, v)$, respectively, we can find the subsurfaces

$$B|_{u\in[u_B^L,u_B^H]}(u, v) \quad \text{and} \quad C|_{u\in[u_C^L,u_C^H]}(u, v),$$

of $B(u, v)$ and $C(u, v)$, which are represented by the blue surfaces in Figure 4.15. Furthermore, the input for the inlet surface $I(u, v)$ requires a lower bound v_C^L and an upper bound v_C^H of $C(u, v)$. Additionally, two angles ω_L and ω_H are specified in relation to the trailing edge of $C(u, v)$. Given this data, we can construct the planes

$$P_{C(u_C^L,v_C^L),D_L} \quad \text{and} \quad P_{C(u_C^L,v_C^H),D_H},$$

where normal vector D_L is given by rotating the vector $C(u_C^L, v_C^H) - C(u_C^L, v_C^L)$ by ω_L around the vector $C(u_C^H, v_C^L) - C(u_C^L, v_C^L)$. The normal vector D_H is given by rotating the vector $C(u_C^H, v_C^H) - C(u_C^H, v_C^L)$ by ω_L around the vector $C(u_C^H, v_C^H) - C(u_C^L, v_C^H)$. In this section, the figures show two especially large values for the angles ω_L and ω_H to better demonstrate the underlying geometric methods.

Given the input lower bounds u_B^L and u_C^L and the input upper bounds u_B^H and u_C^H of $B(u, v)$ and $C(u, v)$, respectively, we can find all intersections of the aforementioned planes and subsurfaces using the methods from 2.3.6. This intersection procedure yields two intersection curves per subsurface. We trim the subsurfaces at the intersection curves to find $I(u, v)$ and $O(u, v)$. A representation of $I(u, v)$ and $O(u, v)$ is represented in orange in Figure 4.15.

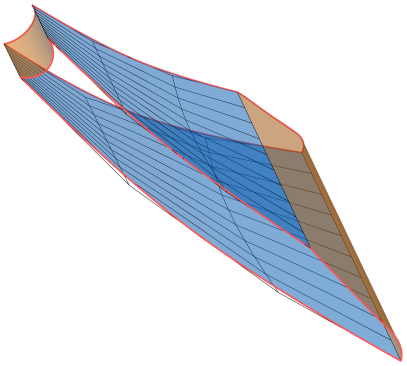
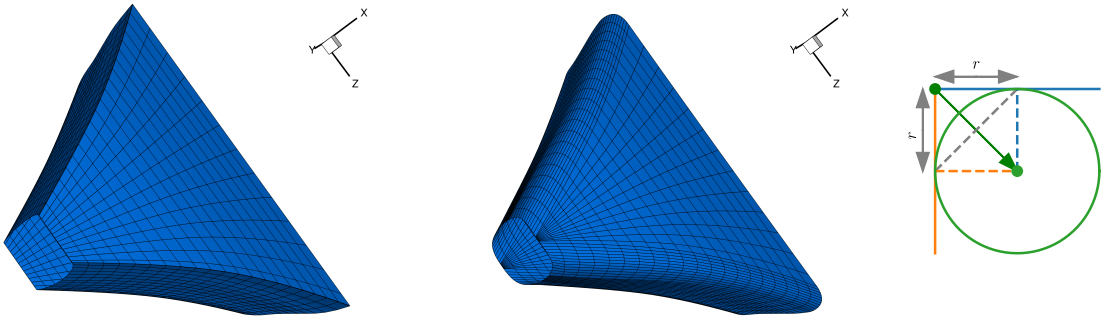


Figure 4.16: Connecting $I(u, v)$ and $O(u, v)$ yields the blue surfaces.

In the next step, we connect the boundary curves of $I(u, v)$ and $O(u, v)$. The suction-sided boundaries ($u = 0$) and the pressure-sided boundaries ($u = 1$) are connected by two NURBS surfaces. Given a value $v \in [0, 1]$, we can construct two more equidistant intermediate control points for each $u \in \{0, 1\}$. For this purpose, we define four input angles. The suction-sided inlet angle α_{suction}, the pressure-sided inlet angle α_{pressure}, the suction-sided outlet angle β_{suction} and the pressure-sided outlet angle β_{pressure}. Figure 4.16 shows an example of this procedure, in which the connecting surfaces are blue.



(a) Without fillets. (b) With fillets. (c) Fillet creation.

Figure 4.17: Adding fillets to the slot surface.

The boundaries of the emerging surface, which are represented in red in Figure 4.16 are then connected using Coons patches. Combining the Coons patches of the two boundaries with the suction- and pressure-sided surfaces we created, we can make a slot surface like on the left side of Figure 4.17.

Next, we add fillets to the isoparameter curves of the slot curves. Since the isoparametric curves are not planar, we cannot use the methods from Section 2.3.5. Instead, we use a different approach. We assume that the corners of the isoparameter curves are right-angled corners. In this case, we can use the fillet procedure shown in Figure 4.17, where we trim the curves near the corner by the input radius r and connect the trimmed curves using a circular arc, the midpoint

of which is a mirror image of the corner.

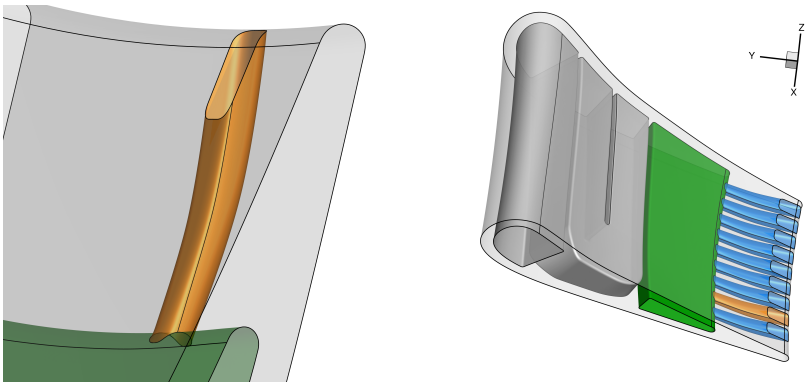


Figure 4.18: An array of slots constructed with realistic angles ω_L and ω_H.

The figures prior to Figure 4.18 show relatively big angles ω_L and ω_H to better visualize the methods used for the construction. Although the presented angles produce valid slot surfaces, smaller angles are favorable in realistic scenarios. A more realistic setup can be seen in Figure 4.18.

4.5 Implementation

The two blade setups that can be seen in Figure 4.1 were generated within 140 seconds on a computer with 8 single-threaded cores, each of which have a clock rate of 4GHz. Alongside the mathematical procedures that were presented in Chapter 2, the performance of CoolingGen was achieved through OpenMP, which is a shared memory API that allows for parallel code execution. CoolingGen currently provides two different output formats: the Tecplot [Tec22] ASCII file format and the CENTAUR [CEN22] ASCII file format. Tecplot was used to create many of the blade/vane geometry surfaces and, in the context of geometry creation, serves as a preview tool. CENTAUR is a mesh generator that is used before CFD (computational fluid dynamics) simulations.

5 Discussion

CoolingGen proves to be a fast and accurate tool to model turbine blade and vane cooling geometries. As can be seen in 4, cooling channels, impingement cooling, film cooling and ejection slots can be created. The underlying geometric methods are mostly well established and provide the ability to create realistic and adaptable cooling designs. The geometry of the resulting surfaces can be altered by changing the underlying parameters in the input XML files. Using the CENTAUR file format, it is possible to combine the surfaces, put a mesh onto the resulting surface and run a CFD simulation on said mesh.

5.1 Cooling Design Process Chain

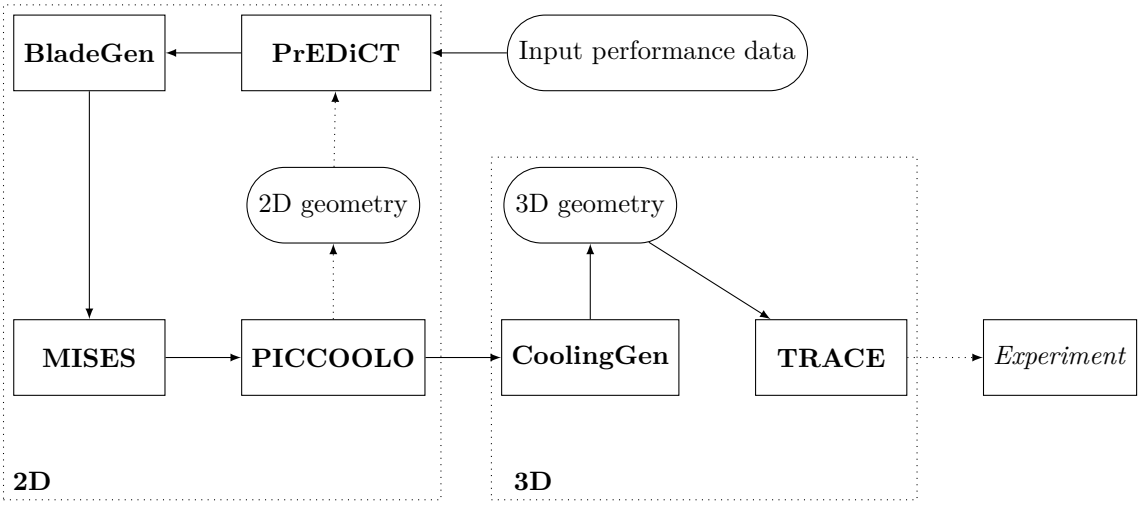


Figure 5.1: The cooling design process chain.

In order to design sensible turbine geometries, a multitude of tools are used at the DLR. First, a two-dimensional profile of the blade or vane is designed. To do this, we use the program PrEDiCT to determine the turbine geometry, flow conditions and the cooling requirements. We can then calculate the blade and vane geometries based on these data using BladeGen. MISES is a CFD simulator, which we use to simulate air flow on a two-dimensional section of the resulting blade/vane-cascades. Using PICCOOLO, we can design and evaluate two-dimensional cooling geometry based on the simulated flow data. We can iteratively apply those tools to redetermine the cooling requirements on the updated blade/vane section and redesign the two-dimensional cooling geometries.

In the future, three-dimensional cooling structures can be designed with CoolingGen using the findings of the previous two-dimensional iteration process. The input XML files for PICCOOLO and CoolingGen are quite similar, although CoolingGen does require more inputs. The resulting three-dimensional geometry can then be evaluated using a three-dimensional CFD simulator called TRACE. If a blade/vane model performs well enough, it can be manufactured and experiments can be performed with it. This cooling design process chain can also be seen in Figure 5.1.

5.2 Desiderata & Future Work

As previously mentioned, the output of CoolingGen consists only of a set of surfaces. However, these surfaces still have to be combined using the help of CENTAUR before creating a CFD mesh. Since CoolingGen (or any other DLR tool in the process chain previously presented) is currently not capable of surface trimming, the implementation of combined geometries remains a challenge that has to be considered in the future. Fortunately, the curves at which the surfaces need to be trimmed are known, as can be seen in Section 4.2 and Section 4.4. Given a notion of trimmed surfaces, it would then also be simple to define a full body made up by CoolingGen's output surfaces. Given any point in space, we could project it onto the combined surface and find the normal of the tangent plane of the combined surface at the projected point. The inner product of the projection path and the normal vector would then always be positive (negative) as long as the original point would lie inside (outside) the surface.

A second shortcoming of the current version of CoolingGen certainly is the structure of the output geometries. Right now, the output surfaces are stored as ASCII point clouds. However, CENTAUR also accepts binary file formats in which NURBS surfaces can be specified using their knot vectors and control points. These file formats provide more accurate outputs (given that the surfaces were created using NURBS) and generally yield smaller file sizes, which is favorable for the creation of CFD meshes. This mode of operation would probably signficantly decrease the running time of CoolingGen.

Lastly, even though CoolingGen currently supports all the geometries presented in Chapter 4, there are still quite a few such geometries that are considered best practices in cooling design and yet are missing. These include but are not limited to turbulation ribs, pin-fin arrays and a second kind of impingement cooling, where neighboring cooling channels are connected through holes. The presented shapes of film cooling holes do certainly also not comprise the diversity of existing film cooling.

6 References

[Aai08] K. Aainsqatsi. *Schematic diagram illustrating the operation of a 2-spool, low-bypass turbofan engine, with LP spool in green and HP spool in purple.* https://commons.wikimedia.org/wiki/File:Turbofan_operation_lbp.svg. 2008.

[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.

[BK90] R.E. Barnhill and S.N. Kersey. "A marching method for parametric surface/surface intersection". In: *Computer Aided Geometric Design* 7.1-4 (June 1990), pp. 257–280. DOI: 10.1016/0167-8396(90)90035-p.

[CCC16] Jorge D. Camba, Manuel Contero, and Pedro Company. "Parametric CAD modeling: An analysis of strategies for design reusability". In: *Computer-Aided Design* 74 (May 2016), pp. 18–31. DOI: 10.1016/j.cad.2016.01.003.

[CEN22] CENTAUR. https://www.centaursoft.com. 2022.

[Coo67] Steven A. Coons. *SURFACES FOR COMPUTER-AIDED DESIGN OF SPACE FORMS*. Tech. rep. June 1967. DOI: 10.21236/ad0663504.

[ELK97] G. Elber, In-Kwon Lee, and Myung-Soo Kim. "Comparing offset curve approximation methods". In: *IEEE Computer Graphics and Applications* 17.3 (1997), pp. 62–71. DOI: 10.1109/38.586019.

[Far01] Gerald Farin. *Curves and Surfaces for CAGD. A Practical Guide (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 2001, p. 520. ISBN: 9781558607378.

[Gia20] Tony Giampaolo. *Gas Turbine Handbook: Principles and Practice*. River Publishers, Nov. 2020. DOI: 10.1201/9781003151821.

[Har96] John C. Hart. "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces". In: *The Visual Computer* 12.10 (Dec. 1996), pp. 527–545. DOI: 10.1007/s003710050084.

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

[Roy15] Rolls Royce. *Jet Engine*. Wiley and Sons, Limited, 2015, p. 288. ISBN: 9781119065999.

[Sha95] Jami J. Shah. *Parametric and feature-based CAD/CAM. concepts, techniques, and applications*. Wiley, 1995, p. 619. ISBN: 0471002143.

[SL16] Sheryl Sorby and Dennis Lieu. *Visualization, Modeling, and Graphics for Engineering Design*. Delmar Cengage Learning, 2016, p. 1086. ISBN: 9781285172958.

[Tec22] Tecplot. https://www.tecplot.com. 2022.

[Wu52] Chung-Hua Wu. "A General Theory of Three-Dimensional Flow in Subsonic and Supersonic Turbomachines of Axial, Radial, and Mixed-Flow Types". In: *Journal of Fluids Engineering* 74.8 (Nov. 1952), pp. 1363–1380. DOI: 10.1115/1.4016114.