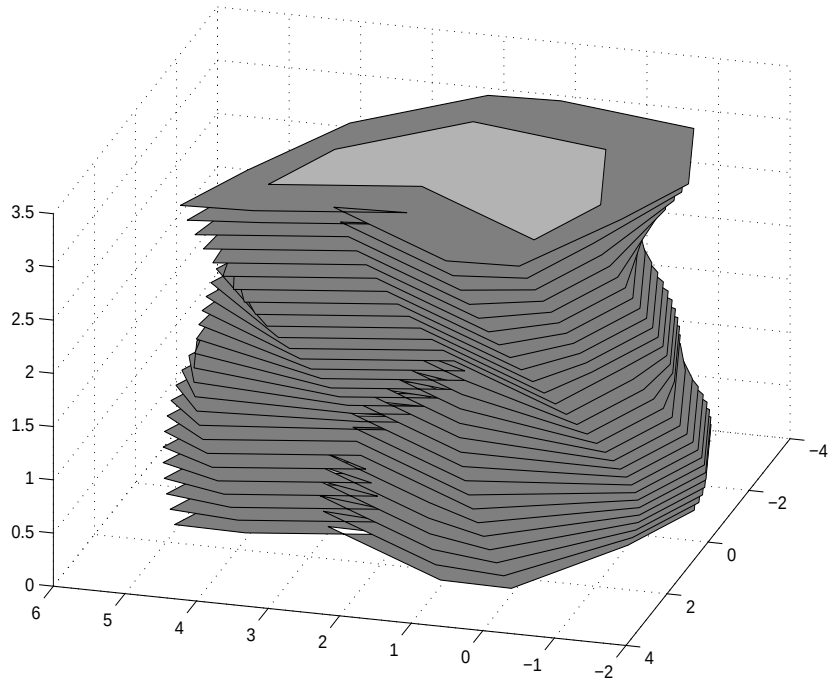
Contact of Bodies in $2D$-space:

Implementing the Discrete Legendre Transform

AI Master's Thesis
Intelligent Autonomous Systems Group



B. Koopen

Faculty of Science
University of Amsterdam

19th February 2002

Abstract

A major factor in the complexity of robot path planning can be found in the demand for collision-free solutions – the agent has to reach its goal without running into obstacles that it might find on its way. We can actually use the notion of two colliding or touching bodies to determine a contact-boundary based upon the shape of the bodies.

We will however focus on handling the local touching of bodies in a 2D setting. Dorst and Van den Boomgaard [4, 6] found that the tangential dilation operation, describing the local touching contact of surfaces/functions, could be reduced to a simple addition operation when using a transform found in the area of convex analysis – the Legendre-Fenchel transform.

2D bodies can however not be described using a continuous function – it will typically be represented using some polygonal representation, hence we will need to find a way to use discrete data combined with the Legendre transform. In this thesis we will examine the basic theory for the discrete Legendre transform which in turn will be used as a foundation for handling discretized functions effectively, mimicing the behaviour of Legendre transformed continuous functions. We then present our algorithm for computing the contact of two polyons using our findings and proposed solutions combined with an adapted version of a concept proposed by Lucet in [8] which allows us to calculate the Legendre transform of concave and convex discrete functions in a worst-case linear instead of the original exponential computational complexity. Moreover, we have been able to extend the use of this approach for non-convex parts of a polygon representing some non-convex body by decomposing it into its concave and convex parts at a decomposition-based additional price.

keywords: collision, polygon, discrete Legendre transform.

Acknowledgements

First and foremost, i would like to thank my supervisor Leo Dorst for his support and advice during the course of the project. Our talks have always, in one way or another, provided me with new insights for approaching the difficulties that we encountered along the way and have strengthened my motivation in pursuing my interests in the area of computational geometry and mathematics in general.

Secondly, many thanks go out to all the 'Golden Boys'. Without them, having gone through the paces would simply not have been the same. I especially want to express my gratitude towards Erik de Ruiter, Matthijs Spaan and Wouter Caarls, with whom i spend most of my time while working on the project and for sharing their insights and opinions.

And last but not least, i want to thank my parents for their everlasting moral support and in helping me through the rough phases.

"The walls felt, to their surprise, smooth, and the floor, save for a step now and again, was straight and even, going ever up at the same stiff slope. The tunnel was high and wide, so wide that, though the hobbits walked abreast, only touching the side-walls with their outstretched hands, they were separated, cut off alone in the darkness. Gollum had gone in first and seemed to be only a few steps ahead. While they were still able to give heed to such things, they could hear his breath hissing and gasping just in front of them. But after a time their senses became duller, both touch and hearing seemed to grow numb, and they kept on, groping, walking, on and on, mainly by the force of the will with which they had entered, will to go through and desire to come at last to the high gate beyond. Before they had gone very far, perhaps, but time and distance soon passed out of his reckoning, Sam on the right, feeling the wall, was aware that there was an opening at the side: for a moment he caught a faint breath of some air less heavy, and then they passed it by".

– **J.R.R. Tolkien**: The Two Towers.

Contents

Chapter 1

Introduction

The analysis of potentially contacting bodies is of great importance in robot-object interaction and computer-graphics. Such an analysis could for instance be used to construct efficient software for obstacle-avoidance and path planning. This would then give us the possibility of determining the path past an object with a high accuracy based on illegal positions of the robot, hence yielding a tighter traverse past the object. But even when the objects are fully known, this is a problem that is not easily solved in real time.

Let us consider using only $2D$ objects and especially polygonal representations of 'real world' objects. Imagine polygons $\mathcal{O}$ and $\mathcal{P}$ to represent some sort of obstacle and a polygon $\mathcal{Q}$ to represent a robotic agent at a location $\mathcal{A}$ operating in an environment $\mathcal{E}$.

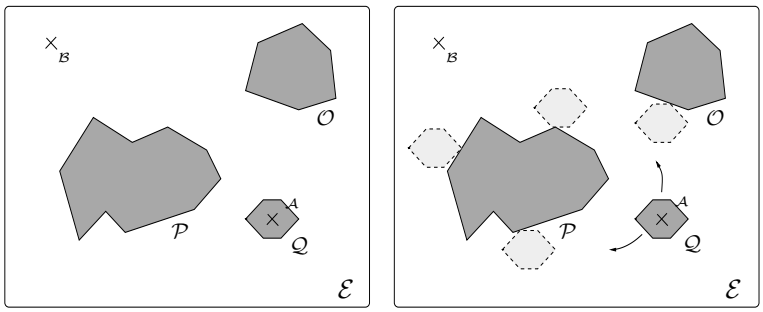
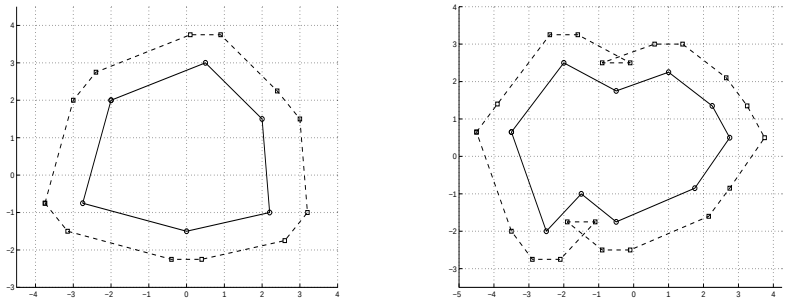


Figure 1.1:

In order to get the agent from $\mathcal{A}$ to rendez-vous point $\mathcal{B}$, it will have to find its way past the obstacles $\mathcal{O}$ and $\mathcal{P}$ such that it in no way runs into $\mathcal{O}$ or $\mathcal{P}$. We can actually use this requirement by using an operation to compute where $\mathcal{Q}$ touches the obstacles and use the outcome to generate a collision-free traverse past $\mathcal{O}$ and $\mathcal{P}$ – if we know the positions of $\mathcal{Q}$ sliding along the outskirts of both $\mathcal{O}$ and $\mathcal{P}$, we would know where it would not be desirable to have our agent $\mathcal{Q}$. These forbidden areas are called *contact-boundaries*[1] given $\mathcal{O}$ and $\mathcal{P}$, denoted by $\mathcal{O}\breve{\oplus}\mathcal{Q}$ and $\mathcal{P}\breve{\oplus}\mathcal{Q}$ and represented by a dashed line.

[1] See fig.1.2(a) and 1.2(b)



(a) (b)

Figure 1.2: Contact-boundaries $\mathcal{O}\breve{\oplus}\mathcal{Q}$ and $\mathcal{P}\breve{\oplus}\mathcal{Q}$.

When considering this type of operation i.e. the touching of two objects to produce their contact-boundary, one possible approach for this analysis is a transform which has its roots in convex analysis theory, the *Legendre-Fenchel transform* $\mathcal{L}$. Dorst and Van den Boomgaard rediscovered this transform in a more general notation while developing their own *Slope transform*. In [6], they found that describing the local contact of two objects, the *tangential dilation*, denoted by $\breve{\oplus}$, which they introduced a few years earlier, could in fact be obtained using the Legendre transform of the separate objects such that

$$(\mathcal{P}\breve{\oplus}\mathcal{Q})(x) = \mathcal{L}^{-1}\big[[\mathcal{L}[\mathcal{P}] + \mathcal{L}[\mathcal{Q}]\big](x)$$

hence computing a difficult tangential dilation operation by taking an easier Legendre transform of each object and taking the inverse Legendre of their sum. Dorst and Van den Boomgaard noted that a similar feature can be found in the computation of signal-convolution using the *Fourier* transform, but instead of *additive*, like the tangential dilation, convolution becomes *multiplicative* when transforming the separate signals to the Fourier domain. Both transforms are in fact spectral decompositions, sharing a number of algebraic similarities (see [6]): one describes a signal using frequencies, the other describes objects using tangent planes.

When examining the touching or *kissing* of two objects in combination with the tangential dilation more closely, we find that objects are considered to be kissing when they are tangent at the point of contact. In doing so, we have no restriction of intersection of the objects in any way[2].
When using the tangential dilation, we allow a possibility of obtaining a contact-boundary which can be *self-intersecting*[3] (see fig.1.2(b)). We can however re-obtain the classical dilation if we omit all the self-intersections, if any, such that we do indeed get a contact-boundary without any intersections of the objects [4].

[2] See fig.1.3.
[3] See [6]
[4] $sup(\mathcal{R}\breve{\oplus}\mathcal{S})(x) = (\mathcal{R} \oplus \mathcal{S})(x)$ (see [4], section 2.2) – this will not be treated in this thesis.

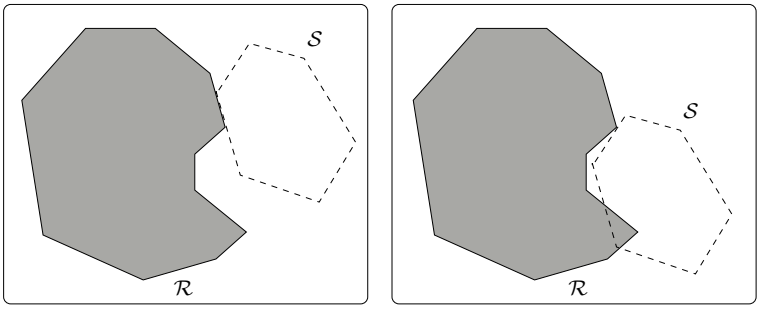


Figure 1.3: Kissing and kissing with penetration.

The emphasis of this thesis will mainly be on the development of a discrete algorithm of the transform, based upon the theory for functions by Dorst and Van den Boomgaard, which will be able to handle polygonal data. In order to do this, we need to examine the workings of the Legendre transform hence requiring the basic theory, operations and difficulties when using discrete data; this will be treated in chapter 2.

Chapter 3 examines the parallels of Fourier and Legendre transform based upon the findings of Dorst and Van den Boomgaard [6] using the decomposition found by Corrias in [2] and focuses on the possibility of using this decomposition to develop a *fast* discrete Legendre transform in much the same way as the fast discrete Fourier transform, found by Cooley and Tukey [9].

In [8], Lucet describes a method of obtaining an even lower computational complexity using tangent-support indexing. Both Corrias and Lucet however focus on using the Legendre transform for solving certain types of non-linear differential equations (Hamilton-Jacobi-equations) and in doing so omit data-points, due to the transform, which are essential when using discrete data describing (a part of) a polygon. Both an adapted version of Lucets' tangent-support and the problems contained in the discrete Legendre transform using discrete data will be treated in chapter 4.

Chapters 5 and 6 focus on the problems that we encounter with polygons in general and possible solutions for describing (sections of) polygons which for instance are not purely concave or convex, like polygon $\mathcal{P}$ in fig.1.1, and in doing so, stay close to the actual theory.

We conclude the thesis with the customary conclusions and (possible) future work.

Chapter 2

Legendre Transform

Before we can focus on developing an algorithm for the tangential dilation using *discrete* data, we will need to examine the theory behind the Legendre transform. In doing so, we have to concentrate on continuous data first i.e. functions.

2.1 Functions

As mentioned in the introduction, the Legendre transform can be found in the areas of convex analysis and specific non-linear differential calculus; it is defined for continuous functions $f : \mathbb{R} \to \mathbb{R}$, denoted by $\mathcal{L}$.

$$\underline{\text{Def}}: \quad \mathcal{L}[f](\omega) \;\equiv\; \big\{ f(u) - \omega u \mid f'(u) - \omega = 0 \big\} \tag{2.1}$$

Note that $\mathcal{L}$ in def.(2.1) transforms f to a function $\mathcal{L} : \Omega \to \mathbb{R}$ with the slope ω as a base.

The next couple of sections will mainly focus on studying convex *or* concave functions with just *one* extreme value.

2.1.1 A concave example

Define some smooth, concave function $f : \mathbb{R} \to \mathbb{R}$:

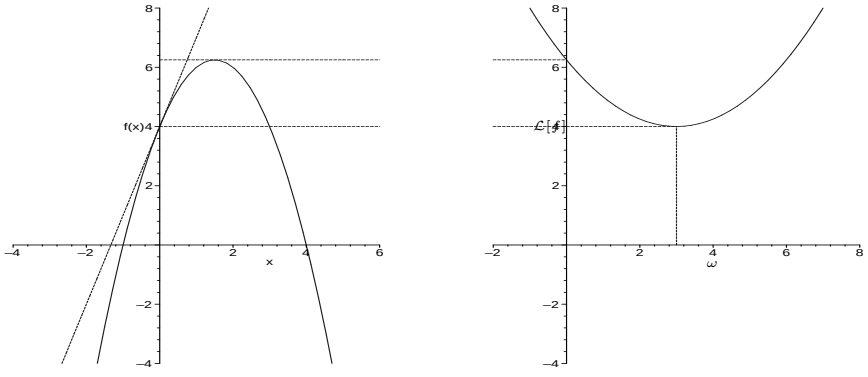
$$f(x) := -x^2 + 3x + 4$$

To get a good feeling of what the Legendre transform does, we need some characteristic sample-points, for instance, the maximum of f at $(\frac{3}{2}, 6\frac{1}{4})$ and (0,4). As can be seen in fig.2.1(b), the maximum of f is transposed to $(0,6\frac{1}{4})$ and (0,4) has become the minimum at (3,4) using the Legendre transform.

An easy, graphical way of looking at this transform using fig. 2.1(b), is by drawing the tangent of a point $P := (p_x, p_y)$ for $f(x)$ in the direction of the y-axis. We then take the intersection-point of the tangent and the axis – this will become the $\mathcal{L}[f]$-coordinate of P; note that this can be found using eq.(2.1) $\rightarrow \quad \mathcal{L}[f] = p_y - \omega p_x$. Since we have the first order derivative, we can easily calculate ω, using

$$\omega = f'(x) = -2x + 3$$

If we then combine the two, we get the Legendre transformed point of P. Take P to be (0,4) for instance; for $x = 0 \ \rightarrow \ \omega = 3$. The tangent in p is of the form: $y = 3 \cdot x + 4$. The intersection-point of the tangent and the y-axis is at (0,4), yielding the transformed point at (3,4).



(a) (b)

Figure 2.1: $f(x)$ and its Legendre transformed $\mathcal{L}[f](\omega)$.

To prove that the Legendre transform of a second degree polynomial actually yields a second degree polynomial $\mathcal{L}[f]$, def.(2.1) can be rewritten through substitution for $f := ax^2 + bx + c$, using

$$\omega = 2ax + b \ \rightarrow \ x = \frac{\omega - b}{2a}$$

yielding

$$\begin{aligned}
\mathcal{L}[f](\omega) &= \left(a\Big(\frac{\omega - b}{2a}\Big)^2 + b\Big(\frac{\omega - b}{2a}\Big) + c \right) \ - \ \Big(\omega\frac{\omega - b}{2a}\Big) \\
&= \Big(\frac{a(\omega^2 - 2b\omega + b^2)}{4a^2} + \frac{b\omega - b^2}{2a} + c\Big) - \Big(\frac{\omega^2 - b\omega}{2a}\Big) \\
&= -\frac{1}{4a}\omega^2 + \frac{1}{4a}2b\omega - \frac{1}{4a}b^2 + c = -\frac{1}{4a}(\omega - b)^2 + c
\end{aligned}$$

□

Now one can deduce that a function with a steeper slope-function will become wider under the Legendre transform using the rewritten term posted above; for higher values of a, directly influencing the slope, we will find wider Legendre transforms due to the term $\frac{1}{4a}$. Conversely, wider functions yield a steeper transform.

2.1.2 Translations

A continuous, concave second degree polynomial yields a continuous, convex second degree polynomial under the Legendre transform, as can be seen in

fig.2.1(a) and 2.1(b). We also know that a translation of a second degree polynomial returns a second degree polynomial. Define a function $\bar{f} = -x^2$ to be the 'standard' concave parabola such that

$$\bar{f}_p(x) + q = f(x)$$

with

$$\bar{f}_p(x) := \bar{f}(x - p)$$

where (p, q) can be seen as a translation vector. Hence, our original function f can be written using $\bar{f}$ and the translation vector $(\frac{3}{2},6\frac{1}{4})$, moving $\bar{f}$ up and to the right.

$$\begin{aligned} \bar{f}_p(x) + q &= -(x - \frac{3}{2})^2 + 6\frac{1}{4} \\ &= -(x^2 - 3x + 2\frac{1}{4}) + 6\frac{1}{4} \\ &= f(x) \end{aligned}$$

The Legendre transform of translations of an n-th degree polynomial also yield an n-th degree polynomial, as can be found in [4] and [6]. Take some function f which we translate by a vector (p, q). The Legendre transform can then be written as:

$$\begin{aligned} \mathcal{L}[f_p](\omega) + q &= \{(f_p(u) - \omega u) + q \mid f_p'(u) - \omega = 0\} \\ &= \{(f(v) - \omega(v + p)) + q \mid f'(v) - \omega = 0\} \\ &= \{(f(v) - \omega v - \omega p) + q \mid f'(v) - \omega = 0\} \\ &= -\omega p + q + \{f(u) - \omega u \mid f'(u) - \omega = 0\} \end{aligned}$$

Note that we actually retrieve the original Legendre transform for $f(u)$ and adding a line $-\omega p$. So, we can rewrite this as

$$-\omega p + q + \mathcal{L}[f](\omega) \tag{2.2}$$

We can use f and $\bar{f}$ as defined above to see that this actually works using the same translation vector and eq.(2.2) for

$$\begin{aligned} \mathcal{L}[f](\omega) &= \frac{1}{4}\omega^2 - \frac{3}{2}\omega + 6\frac{1}{4} \\ \mathcal{L}[\bar{f}](\omega) &= \frac{1}{4}\omega^2 \end{aligned}$$

such that

$$\begin{aligned} \mathcal{L}[\bar{f}_p](\omega) + q &= -\frac{3}{2}\omega + \mathcal{L}[\bar{f}](\omega) + 6\frac{1}{4} \\ &= \frac{1}{4}\omega^2 - \frac{3}{2}\omega + 6\frac{1}{4} \\ &= \mathcal{L}[f](\omega) \end{aligned}$$

2.1.3 Combining concave functions

We can use the Legendre transform for computing the *contact-boundary* of two functions, g^* and f, both defined in an individual frame, denoted by $\mathcal{F}$ and $\mathcal{G}$.

For the time being, consider g^* and f to be rigid bodies where g^* can for instance represent a robot moving around and f to be an obstacle in the vicinity of g^*. If we were to move/slide g^* around our fixed object in such a way that that g^* does not penetrate the boundary of f we would obtain the *contact-boundary*; g^* *kisses* f. The result can be perceived as being a boundary itself; g^* can move around f freely as long as it does not venture too close to f. Note that we can describe the possible configurations of our robot using rotations $\mathcal{R}$ and translations $\mathcal{T}$. In this thesis we will focus on the latter one, using the robot in a fixed orientation, hence omiting the complex boundaries which occur when adding rotations.

Using the notion obtained above for 2D data, we can return to functions. In a schema, computing the contact-boundary would look like this.

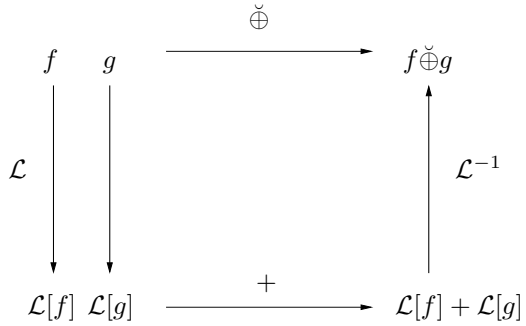


Figure 2.2: Finding the contact-boundary using the Legendre transform.

So, to get the contact-boundary $f \breve{\oplus} g$, we can add up the separate Legendre transforms of the 2 functions and do an inverse Legendre on the sum (see [4] and [6]). Note that fig.2.2 uses g instead of g^*. The function g^* is actually known as the *transpose* of g; g is the point-mirrored image of g^*:

$$g(x) = -(g^*(-x))$$

An additional factor in determining the contact-area, is the use of a *reference-point* ρ. In general, ρ can be any point in $\mathcal{G}$, determining the relative position of g^* with respect to f, hence influencing the transform of g^*; denote this as g^*_ρ. Take ρ to be the point (p, q) in $\mathcal{G}$. By simply applying a vector translation we can move g^* to a new position with (p, q) placed in the origin of $\mathcal{G}$.

$$g^*_{-p}(x) - q$$

Using this translation and the notion of a transpose function, we no longer have to worry about keeping track of ρ; it's simply the *origin* of $\mathcal{G}$. A few examples will help to clarify the influence and importance of the choice of a reference-point.

Let's look at a simple setup. Define 2 functions $f : \mathbb{R} \to \mathbb{R}$ and $g^* : \mathbb{R} \to \mathbb{R}$;

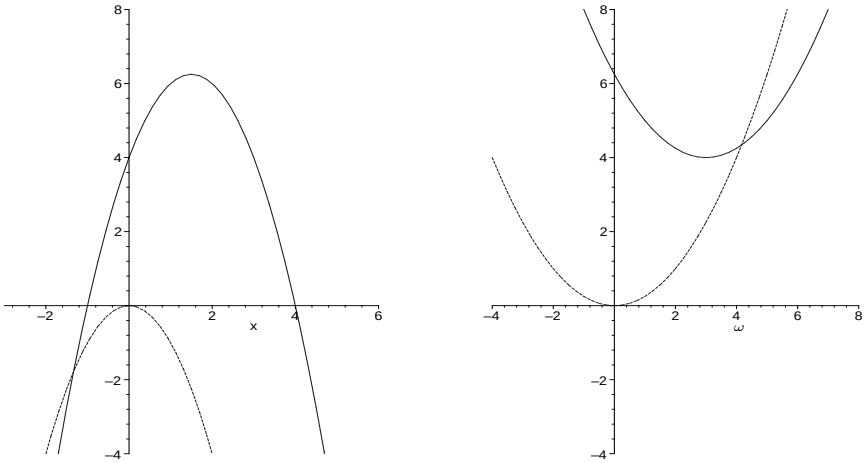
- $f(x) := -x^2 + 3x + 4$
- $g^*(x) := x^2$

Note that we will use the g^* as shorthand for $g^*_{(0,0)}$. Now, g can be found using

$$-g^*(-x) = g(x) \;\rightarrow\; g(x) = -((-x)^2) = -x^2$$

To use graphics, we simply place the origin/reference-point of $\mathcal{G}$ on the origin of $\mathcal{F}$. This yields fig.2.3(a). In fig.2.3(b), we see the Legendre transforms of f and g, $\mathcal{L}[f]$ and $\mathcal{L}[g]$, with

$$\begin{aligned}\mathcal{L}[f](\omega) &= \frac{1}{4}\omega^2 - \frac{3}{2}\omega + 6\frac{1}{4}\\ \mathcal{L}[g](\omega) &= \frac{1}{4}\omega^2\end{aligned}$$



(a) (b)

Figure 2.3: Transforming f and g.

yielding the second degree polynomial

$$(\mathcal{L}[f] + \mathcal{L}[g])(\omega) = \frac{1}{2}\omega^2 - \frac{3}{2}\omega + 6\frac{1}{4}$$

seen in fig.2.4(a). Now we can actually compute the contact-boundary using the inverse Legendre transform. Since we are still focusing on second degree polynomials, we only need the basic inverse transform, denoted as

$$\mathcal{L}^{-1}[F](u) \equiv \{F(\omega) + u\omega \mid F'(\omega) = -u\} \tag{2.3}$$

That it is indeed the inverse follows from

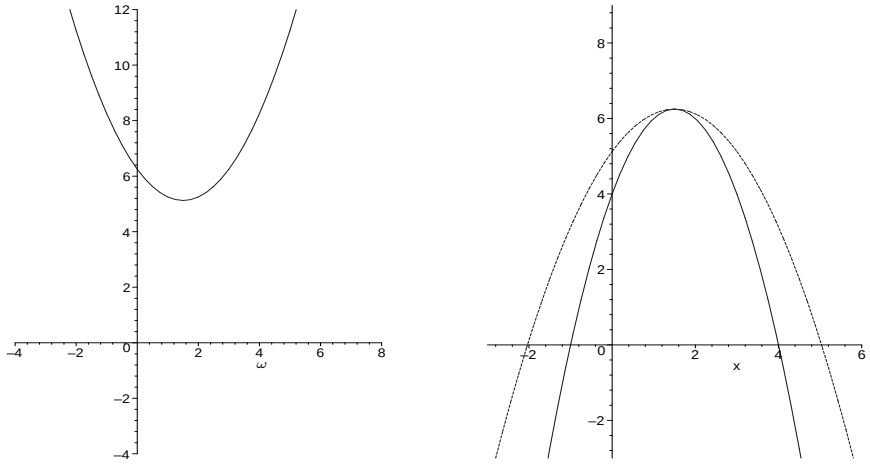
$$\mathcal{L}^{-1}\big[\mathcal{L}[f]\big](u) = f(u)$$

as found in [4] and [6]. The boundary formed between f and $f\breve{\oplus}g$, as can be seen in fig.2.4(b), is the area where g^* may not go, given reference-point ρ and f. To put it in an other perspective, g^* can be slid along, with the origin of $\mathcal{G}$ exactly on $f\breve{\oplus}g$ [1]; g^* and f will only kiss each other. We can determine $f\breve{\oplus}g$ in terms of x by applying a rewrite using the first order derivative [2]. So

$$-x = \omega - \frac{3}{2} \;\rightarrow\; \omega = -x + \frac{3}{2}$$

resulting in

$$\begin{aligned}\mathcal{L}^{-1}\big[\mathcal{L}[f] + \mathcal{L}[g]\big](x) &= (\frac{1}{2}(-x+\frac{3}{2})^2 - \frac{3}{2}(-x+\frac{3}{2}) + 6\frac{1}{4}) + x(-x+\frac{3}{2}) \\ &= -\frac{1}{2}x^2 + \frac{3}{2}x + 5\frac{1}{8} = (f\breve{\oplus}g)(x)\end{aligned}$$



(a) (b)

Figure 2.4: Constructing $f\breve{\oplus}g$.

The same thing can be done for g^* with different reference-points and leaving f unaltered, for instance $g^*_{(0,2)}$. This in turn yields

$$g(x) = -x^2 + 2$$

The result, as can be seen in fig.2.5, is that $f\breve{\oplus}g$ has moved up 2 compared to fig.2.4(b) due to the choice of ρ

$$(f\breve{\oplus}g)(x) = -\frac{1}{2}x^2 + \frac{3}{2}x + 7\frac{1}{8}$$

[1] $f\breve{\oplus}g$ represented by dotted line.
[2] See eq.2.3.

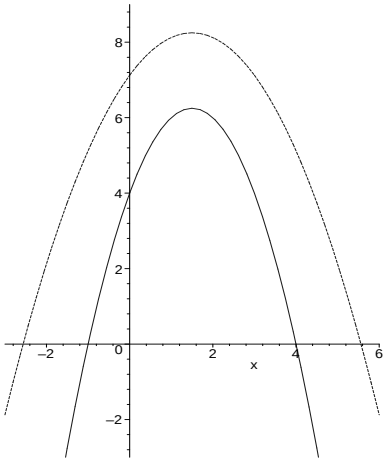


Figure 2.5:

Another possible reference-point for g^* could be $(\frac{1}{2}, 2\frac{1}{4})$, so $g^*_{(\frac{1}{2},2\frac{1}{4})}$, such that

$$g(x) = -x^2 + x + 2$$

Note that if we look at g^*, ρ is to the right of the minimum of g^*. This means that the actual contact-boundary needs to be wider on the right side of f and tighter on the left side.

$$(f \breve{\oplus} g)(x) = -\frac{1}{2}x^2 + 2x + 6\frac{1}{2}$$

This notion is confirmed in fig.2.6.

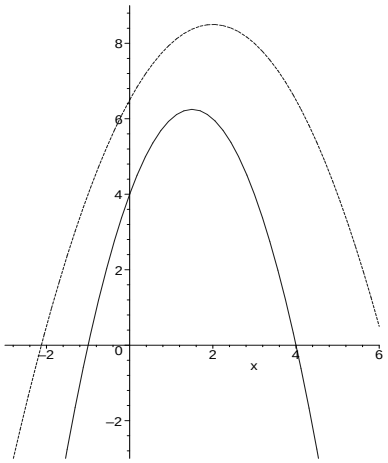


Figure 2.6:

2.2 Discrete Functions

Discrete approximations of functions for some finite domain can also be used for computing collisions – we can view these as being a part of a polygon. The problem lies in handling the data-format; these *discrete functions* are built up of edges and vertices, hence, we can not describe them exactly using some continuous polynomial.

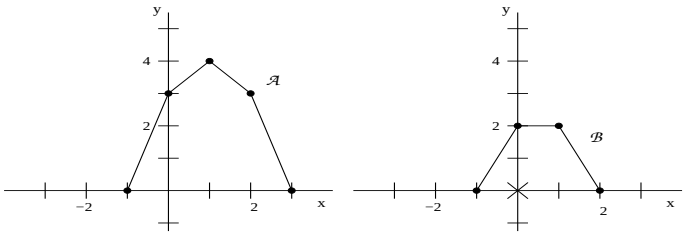


Figure 2.7: Examples of discrete functions.

We already noted that, apart from the data-format, discrete functions have finite bounds. We will examine this in more detail in section 2.2.2. But first, we will need to focus on the features of discrete functions and the influence that the Legendre transform has on them. We will examine the easier of the two in combination with the Legendre transform first.

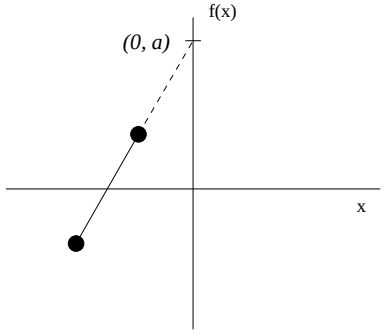


Figure 2.8: From edge to vertex.

As can be seen in fig.2.8, the edge E has a certain tangent ω, which intersects the y-axis at $(0,\ a)$. We know that in the continuous case, a becomes the $\mathcal{L}$[f]-coordinate in the Legendre frame. In the discrete case, edge E becomes the point (ω,a) under the Legendre transform.

With vertices, it is slightly more complicated. The focus in fig.2.9 lies mainly on point β. We already know what happens to the edge $\{\alpha,\ \beta\}$, it gets transformed to p_1 at $(\omega_1,\ a)$. Note that β has a tangent ω_1, but, since β also lies on the edge $\{\beta,\ \gamma\}$, it also has a tangent $\omega_2 = 0$. Moreover, all tangents ω_i which run from ω_1 to ω_2 are valid tangents such that:

$$\omega_1 \ \geq \ \omega_i \ \geq \ \omega_2$$

i.e. they form a *cone* of tangents for point β with angle φ. We know that β, since it lies on the edge $\{\alpha,\ \beta\}$, get transformed to p_1. But since β also lies on the edge $\{\beta,\ \gamma\}$, it gets transformed to p_2 as well. If we calculate some of these ω_i, we will find that they end up on the edge $\{p_1,\ p_2\}$. Note that we can accurately describe this using the cone in terms of ω_1 and ω_2

$$\omega_i = \lambda\omega_1 + (1-\lambda)\omega_2 \tag{2.4}$$

with

$$0 \leq \lambda \leq 1$$

Take the intersection of ω_1 trough β with the y-axis to be (0,o_1) and (0,o_2) for ω_2.

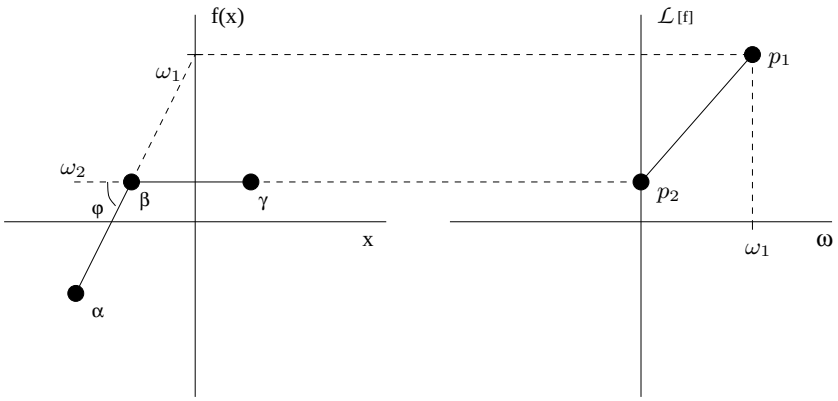


Figure 2.9: From vertex to edge.

Using the linear relation above we get

$$o_i = \lambda o_1 + (1-\lambda)o_2$$

The same linear relations hold in the Legendre domain, hence, a *vertex* in the spatial domain gets transformed to an *edge* in the Legendre domain based on the 'valid' slopes using eq.(2.4).

So, vice versa, using the insight obtained above, we can be even more specific about the transformation of an edge under the Legendre transform: edge E in the spatial domain becomes a *cone* at a point in the Legendre domain.

Before proceeding, we need to re-examine our definition of the Legendre transform to be able to handle discrete functions. It turns out, that we can use the *extended* Legendre transform as found in [6]

$$\underline{\text{Def}}: \quad \mathcal{L}[f](\omega) \ \equiv \ \max_u \left\{ f(u) - \omega u \right\} \tag{2.5}$$

Note that the *max*-term actually calculates the extreme value, so $f'(u) - \omega = 0$, hence retrieving def.(2.1). So, for every possible value of ω, we need to run through all of the vertices of the function. We can easily calculate useful ω-values using vertices (x_i, y_i) and (x_{i+1}, y_{i+1}).

$$\omega_i = \left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \right) \tag{2.6}$$

This way, we only get ω-values which are in the actual discrete function. Consider a (rough) discrete representation D_f of the function $f = -x^2$.

x	-2	-1	0	1	2
$D_f(x)$	-4	-1	0	-1	-4

Using def.(2.5) and eq.(2.6), we get[3]

ω	3	1	-1	-3
$\mathcal{L}[D_f](\omega)$	2	0	0	2

The inverse operation, as found in eq.(2.3), can be presented in a similar fashion to the Legendre transform in eq.(2.5).

$$\underline{Def}: \quad \mathcal{L}[f]^{-1}(x) \;\equiv\; \min_{\omega}\big\{F(\omega) + x\omega\big\} \tag{2.7}$$

where the x's can be obtained in much the same way as the slopes in (2.6).

$$x_j = -\Big(\frac{\mathcal{L}[f]_{j+1} - \mathcal{L}[f]_j}{\omega_{j+1} - \omega_j}\Big) \tag{2.8}$$

It has to be noted that definitions (2.5) and (2.7) work only for *concave* discrete functions. When using convex discrete functions, we have only have to use *min* in def.(2.5) and *max* in def(2.7).

2.2.1 Translations

As mentioned in section 2.1.2, eq.(2.2), vector translation in the Legendre domain yields the Legendre transform of the function with an additive factor dependent on the vector translation itself.

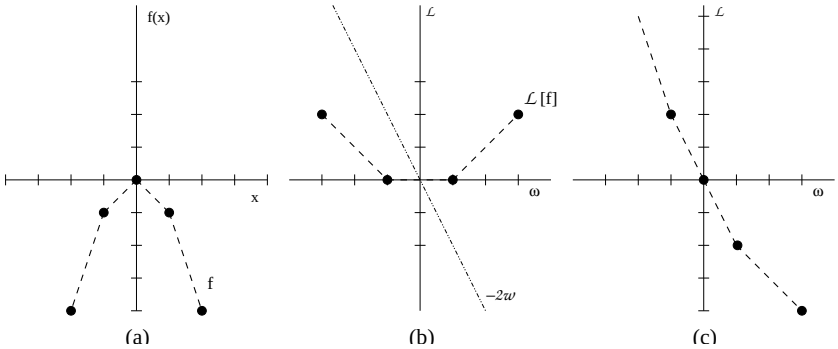


Figure 2.10: Translations in the Legendre-domain.

If we define the linear translation t to be (2,0), D_f, as introduced in the previous section, would move 2 to the right. Fig.2.10(b) shows the Legendre transform of D_f, together with the $-\omega t$-factor. In fig.2.10(c) we can see the result of the translation over t; it's simply the sum of the line $-\omega t$ and the Legendre transform in fig.2.10(b) as found in eq.(2.2), stretching and tilting it.

[3]See fig.2.10(a) and (b)

2.2.2 Infinity

An additional problem when using discrete functions that was already noted in the beginning of section 2.2 is that the domain is finite. So, before we can look at combining concave discrete functions, we have to examine the influence of a finite domain. This can be done by combining a concave function f and a band-pass function. Define a band-pass function $\mathcal{B}$ such that

$$\mathcal{B}(x) = \begin{cases} 0 & \text{if } p < x < q \\ -\infty & \text{else} \end{cases}$$

The result of this addition, as seen in fig.2.11, is quite straightforward; the polynomial f will be defined on the domain $\{p, q\}$, hence creating a 'discretized' function.

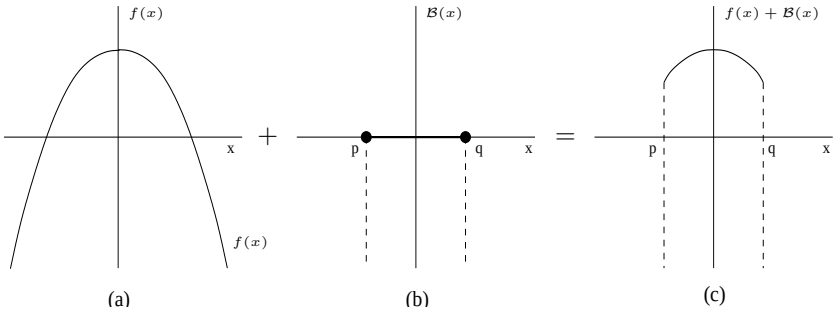


Figure 2.11: 'Discretizing' a continuous function.

To examine the influence of this discretization in the Legendre domain, we can obtain the Legendre transform of $f(x) + \mathcal{B}(x)$ using

$$f(x) + \mathcal{B}(x) \xrightarrow{\mathcal{L}} (\mathcal{L}[f] \breve{\oplus} \mathcal{L}[\mathcal{B}])(\omega) \tag{2.9}$$

as found in [4]. Since f is just a simple concave second degree polynomial, $\mathcal{L}[f](\omega)$ is a convex second degree polynomial, so we need only look at the transform of the band-pass function $\mathcal{B}$.

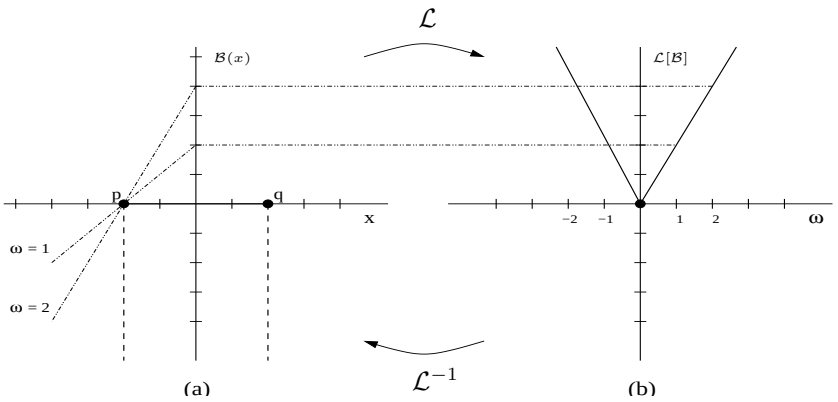


Figure 2.12:

As can be seen in fig.2.12, $\mathcal{B}$ becomes a cone under the Legendre transform[4]. Now we can compute $\mathcal{L}[f]\breve{\oplus}\mathcal{L}[\mathcal{B}]$.

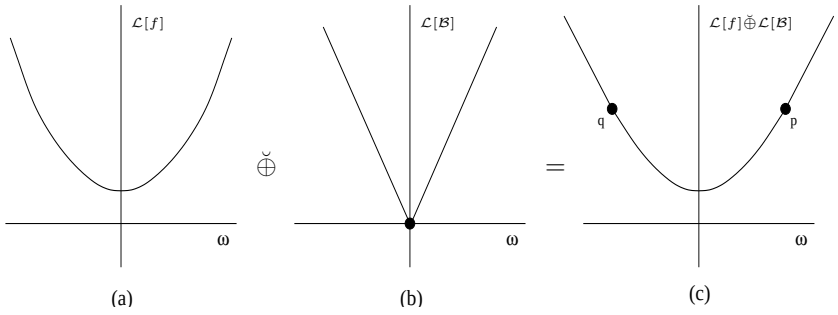


Figure 2.13: Constructing $\mathcal{L}[f]\breve{\oplus}\mathcal{L}[\mathcal{B}]$.

In fig.2.13, we see that a convex second degree polynomial and the cone result in a convex section from q to p. Beyond these points, $\mathcal{L}[f]\breve{\oplus}\mathcal{L}[\mathcal{B}]$ becomes linear as an effect of the cone, as found in [4], section 2.4. So, the cone in fig.2.13(b) limits the slopes from a certain threshold $-p\omega$ for $\omega > 0$ and $-q\omega$ for $\omega < 0$ as found in fig.2.12(b).

Now we are at the point where we can return to concave discrete functions. If we want to use a discrete function in the Legendre transform, we have to do something similar to fig.2.11, hence defining a discrete function D_f in such a way that the first and last vertex of the discrete function become the domain for which the function holds and that beyond these vertices all x-values will have a y-value of $-\infty$, so

$$D_f = \begin{cases} D_f(x) & \text{if } x_{v_1} < x < x_{v_n} \\ -\infty & \text{else} \end{cases}$$

with x_{v_1} and x_{v_n} denoting the first and last x-value of the vertices of $\mathcal{A}$.

2.2.3 Potential difficulties

Now that we can use a discrete function in the Legendre transform by adding a 'sense of infinity', we can start looking at computing the contact area of 2 discrete functions. But, problems due to the data-format of discrete functions do not make it straightforward. The following issues need to be resolved:

- loss of vertices due to Legendre and inverse Legendre transform.
- incompatible ω-values.
- handling more complex discrete functions which can not strictly be defined as being concave or convex.

The first problem is due to the calculation of the ω-values. If we have a discrete function D_f with N vertices, we get N-1 vertices for $\mathcal{L}[D_f]$, as can be seen in

[4]See [4] for proof.

fig.2.10, section 2.2.1 and the corresponding tables. When we then apply the inverse Legendre transform, we loose another vertex. We do however have to keep in mind that

$$\mathcal{L}^{-1}[\mathcal{L}[D_f]](u) = D_f(u)$$

In other words, a discrete function D_f with N vertices has to yield D_f with the same N vertices after an inverse Legendre transform of the Legendre transform $\mathcal{L}[D_f]$.

An additional problem is handling discrete functions with different ω-sets, so how to treat the addition of two sets of ω-values differing in the amount of values and/or different values. Hence, we have to find a way to combine 2 Legendre transformed discrete functions in such a way that we obtain a valid result i.e. they will only kiss each other when they 'share' a slope.

We also want to be able to handle discrete functions which have a more complex structure, *non-convex* discrete functions.

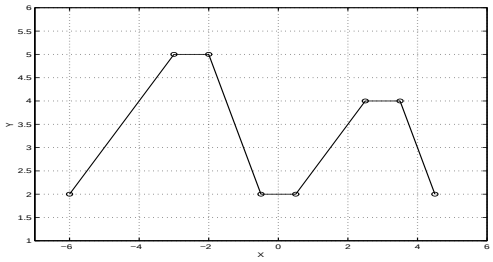


Figure 2.14: An example of a non-convex discrete function.

These non-convex discrete functions can be perceived as being built up out of several concave and convex segments, as can be seen in the example above. If we want to use this notion, we have to find a way to separate and re-attach the segments in such a way that no data is lost and that all segments are used properly in the resulting transform.

We will examine these problems in more detail in the chapter on the algorithm and propose a solution to handle them efficiently.

Chapter 3

Fast Legendre Transform

The complexity of the Legendre transform is of the same order as the Fourier Transform. For every slope-value ω and a function described by N sample-points, we would need N operations. So, for $|\Omega|$ slope values, we get a complexity of $O(|\Omega|N)$.

In the mid-1960's, *Cooley* and *Tukey* introduced a Fast Fourier algorithm based on the derivations of the discrete Fourier transform by *Danielson* and *Lanczos* [9]. Danielson and Lanczos proved that it was possible to rewrite a discrete Fourier transform over N sample-points, with N being a power of 2, as the sum of two separate Fourier transforms over $N/2$ points. Although the rewriting of the Legendre transform is slightly different from that of the Fourier Transform, we can see clear analogies.

3.1 Discrete Fourier

Using the *Danielson-Lanczos Lemma*, the discrete Fourier transform can be written as:

$$\begin{aligned} F(\omega_k) &= \sum_{u=0}^{N-1} e^{-2\pi i u \omega_k / N} f(u) \\ &= \sum_{u=0}^{(N/2)-1} e^{-2\pi i u \omega_k/(N/2)} f(2u) + W^k \sum_{u=0}^{(N/2)-1} e^{-2\pi i u \omega_k/(N/2)} f(2u+1) \\ &= F(\omega_k)^e + W^k F(\omega_k)^o \\ &= F_k^e + W^k F_k^o \end{aligned} \tag{3.1}$$

with

$$W^k \equiv (e^{-2\pi i/N})^{\omega_k}$$

where F_k^e denotes the k-th component of the N/2-length Fourier transform formed from the even components of the original data, while F_k^o does the same for the odd components. Also notice the introduction of the factor W^k which is

used as translation-factor, so quite similar to the $-\omega t$-factor introduced in the last section of the previous chapter.

3.2 Discrete Legendre

Before looking at the discrete Legendre transform, we'll introduce the notion of step-size ϵ, determined by N.

$$\epsilon \;:=\; \frac{1}{N}$$

Now, define some concave discrete function f for $D := [0,1]$. Applying the *Danielson-Lanczos Lemma*, as can be found in [2], yields

$$\begin{aligned}
\mathcal{L}_N[f](\omega_k) &= \max_u \{f(u) - \omega_k u\} \\
&= \max\Big\{ \max_u \{f(u) - \omega_k u\},\ -\omega_k\epsilon + \max_u \{f(u+\epsilon) - \omega_k u\}\Big\} \\
&= \max\Big\{\mathcal{L}_{\frac{N}{2}}[f](\omega_k),\ -\omega_k\epsilon + \mathcal{L}_{\frac{N}{2}}[f_{-\epsilon}](\omega_k)\Big\} \\
&= \max\Big\{\mathcal{L}_{\frac{N}{2}}[f](\omega_k),\ W_k + \mathcal{L}_{\frac{N}{2}}[f_{-\epsilon}](\omega_k)\Big\} \qquad (3.2)
\end{aligned}$$

with

$$W_k \;=\; -\omega_k\epsilon \;=\; -\frac{\omega_k}{N}$$

In the case of the Fourier transform we need an addition of two terms, where one has a *multiplicative* offset-factor. We see the same sort of thing happening to the Legendre transform when splitting it, but *instead* of an addition we need a maximum-operation on two terms, with one term having an *additive* offset-factor (see eq.3.1 and 3.2).

A more general notation will prove to be quite handy, since pre-processing the function to map it to D will take extra time. Given a discrete function f, take the boundary points to be (p_x, p_y) and (q_x, q_y). Take the x-values of these points such that $p_x < q_x$. Now, we can map D to $[p_x, q_x]$, using [4]

$$f((q_x - p_x)x) \;\longrightarrow\; \mathcal{L}[f](\frac{\omega}{q_x - p_x})$$

such that

$$\begin{aligned}
f_{-p_x}((q_x - p_x)x) &= f((q_x - p_x)x + p_x) = p_y \qquad \textit{for } x = 0 \\
f_{-p_x}((q_x - p_x)x) &= q_y \qquad \textit{for } x = 1
\end{aligned}$$

We will also have to redefine our definition of ϵ

$$\epsilon \;:=\; \frac{q_x - p_x}{N} \qquad (3.3)$$

Now, we can rewrite eq.(3.2) as

$$\max\Big\{\mathcal{L}_{\frac{N}{2}}[f_{-p_x}](\frac{\omega_k}{q_x - p_x}),\ Wk + \mathcal{L}_{\frac{N}{2}}[f_{-p_x-\epsilon}](\frac{\omega_k}{q_x - p_x})\Big\} \qquad (3.4)$$

thus running through N samples from point p to q-ϵ. Note that the first part of eq.(3.2) and eq.(3.4) calculate the samples

$$\underbrace{\{0, 2\epsilon, 4\epsilon, 6\epsilon, 8\epsilon \cdots 1-2\epsilon\}}_{\frac{N}{2}\ \textit{even}\ \text{terms}}$$

and that the second part calculates

$$\underbrace{\{\epsilon, 3\epsilon, 5\epsilon, 7\epsilon, 9\epsilon \cdots 1-\epsilon\}}_{\frac{N}{2}\ \textit{odd}\ \text{terms}}$$

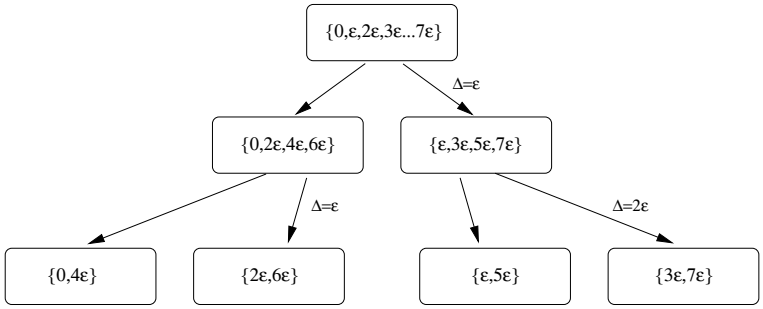
due to the additive ϵ-term in determining $f(u)$. We can even do successive subdivisions using eq.(3.4), thus splitting the 'even' half over N samples into 2 sections of each $N/4$ samples, so an 'even-even' ($\{0, 4\epsilon \cdots\}$) and an 'even-odd' part ($\{2, 6\epsilon \cdots\}$). The same applies for the 'odd' half, yielding an 'odd-even' ($\{\epsilon, 5\epsilon \cdots\}$) and an 'odd-odd' part ($\{3\epsilon, 7\epsilon \cdots\}$). Using the following construct, we can keep splitting each section up to the point where each section only contains 2 values.

Fast Discrete Legendre Transform

fdlt(ω,f,ϵ,Δ,S)
  with
    offset Δ *initialized at 0*
    stopvalue S *set to* $(q_x - p_x)$

  for $2\epsilon < S$
    return *max* $\big($*fdlt*(ω,f, 2ϵ, Δ,S), $-\omega\epsilon$ + *fdlt*(ω,f, 2ϵ, $\Delta + \epsilon$,S)$\big)$
  else
    $temp1 = f(u + \Delta) - \omega u$
    $temp2 = f(u + \Delta + \epsilon) - \omega u$
    return *max(temp1, temp2)*
  end

For example, 8 samples at the top get divided into 2 sections of 4, which in turn can be divided into 4 sections of 2 samples, yielding



This way, we can reduce the computation of an $|\Omega|$-point Legendre transform,

obtained from an N-point sampled function, to $log(N)$ calculations for each slope-value ω. Hence, calculating $|\Omega|$ slope-values would be of the order $|\Omega|$ $log(N)$.

3.2.1 Sampling and its Side-effects

Let's look at the fast Legendre algorithm as proposed in the previous section. When sampling, we're actually using linear interpolation to get data-points with a fixed spacing ϵ based upon the original inputdata. In order to obtain a faster transform, we have to keep in mind that, given a discrete function with n vertices and $|\Omega|$ unique slopes, we need $O(|\Omega|n)$ calculations. This means that we can only get a more efficient algorithm in the number of computations used if we use a number of samples N such that

$$log(N) < n$$

with N being a power of 2. Note that the focus in this setting is mainly based on efficiency; we do not have any qualitative restrictions.

Take discrete function $\mathcal{B}$ as seen in section 2.2, fig.2.7; $\mathcal{B}$ has 4 points and 3 slopes. In this setting, an obvious sample-rate would be

$$N = 4 \ \rightarrow \ \epsilon = 1$$

hence, we would only need $3 \cdot log(4) = 6$ calculations instead of the original 12. Note that the data is already equidistant and that the number of points are a power of 2. When altering either of these two features, we find that there are a number of drawbacks when sampling is applied to discrete functions with a low number of vertices. Since the drawbacks apply for both, we will focus our attention on the last feature.

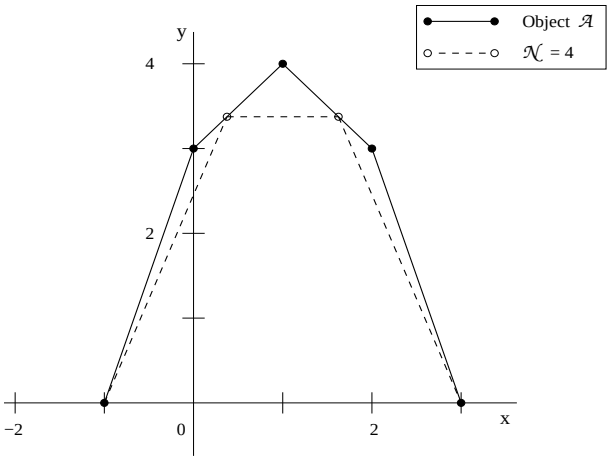


Figure 3.1: Low sample-rate, loss of distinct features.

Discrete function $\mathcal{A}$ has 5 vertices and 4 slopes. When using the same amount of samples, we find that distinct features of $\mathcal{A}$ have been cut, as can be seen in fig.3.1. It is obvious that we do not want to use these rough approximations

in contact computation. To increase the accuracy we would need to use more samples.

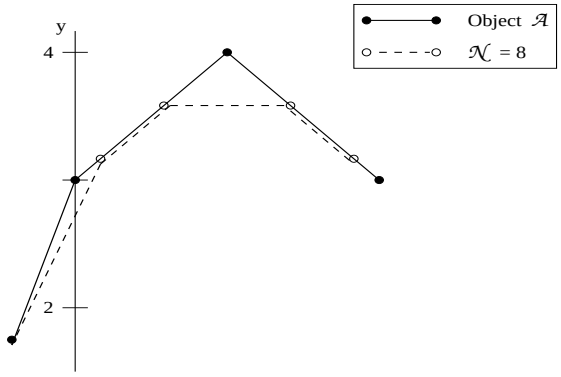


Figure 3.2: Better approximation using higher sample-rate.

In fig.3.2, we have increased the number of samples to 8 and extracted a section of $\mathcal{A}$. The drawback of the sampling-proces is that it yields 3 additional slopes; the sampling does not correspond exactly to the vertices of $\mathcal{A}$. Moreover, in the worst case scenario, $|\Omega|$ unique slopes yield $|\Omega|$*-1* additional slopes, thereby increasing the number of calculations needed to determine the Legendre values. Although being more accurate, the sampling still cuts the original discrete functions' important features as can be clearly seen when looking at the third vertex of $\mathcal{A}$. We have to find a way to determine the number of samples based on the discrete function itself such that the loss of features is minimized.

There are a few ways to use the data of the discrete function, but it is not at all straightforward. In the examples used above, we already noticed that the number of vertices in the original discrete function need not be a good estimate for the number of samples. Conversely, if we were to use the same discrete function $\mathcal{A}$ but with a much greater number of vertices to 'describe' it, we would find that the actual number of samples needed for a satisfactory result would be relatively low compared to the number of vertices.

Another way would be to look at the data-spacing. This way we can determine the smallest edge of the discrete function, hence being able to set the sample-rate in such a way that we include all the edges. But this also leaves us with possible problems; take the same discrete function $\mathcal{A}$ but instead of spreading a large amount of extra points on the existing edges, concentrate them on a single edge. This way, we obtain a very small value for the shortest edge, hence increasing the number of samples. This in turn means that all but one of the edges of $\mathcal{A}$ will be 'oversampled', resulting in the same problem as described previously i.e. only a small amount of values will actually be worthwile.
We can do slightly better by using the average data-spacing but this also leads to possible problems since the longer edges will stretch the average which will in turn set the sampling in such a way that a number of edges, or even worse, potentially interesting features, will be cut.

We can extend this if we want to use sampling for treating non-convex discrete functions like the one in section 2.2.3; 'undersampling' one or several segments would lead to large differences between the original discrete function and its sampled image. This still leaves us with treating two colliding discrete functions; the number of samples used for both discrete functions might potentially be too low (or too high) for one of the discrete functions if we were to use one of the possible solutions mentioned above.

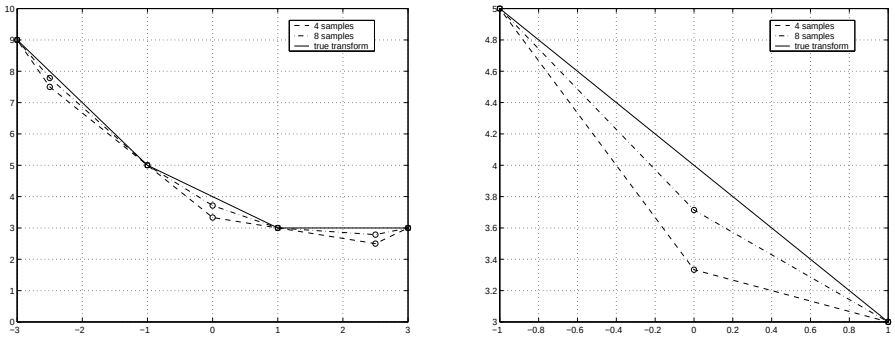
In order to keep the number of samples at an acceptable level *and* increase the accuracy, the best approach is to define a threshold κ; a sampled vertex v_ϵ is acceptable if

$$y_v - y_{v_\epsilon} \;\leq\; \kappa \;\equiv\; c\Big(\frac{x_n - x_1}{N}\Big) \;=\; c\epsilon$$

using some user-defined constant c with

$$0 < c \leq 1$$

and y_{v_ϵ} being the y-value found through interpolation of the x-value closest to the x-value of vertex v using sample-rate ϵ. Once the threshold κ has been set, we need only check the difference of the interpolated data with each point in the original discrete function and update N if needed – increasing the number of samples will decrease the distance of the sample-point nearest to v, thereby decreasing the error in the approximated y-value until $y_v - y_{v_\epsilon} \leq \kappa$.



(a) (b)

Figure 3.3: Accuracy using sampling.

In fig.3.3(a) and 3.3(b)[1] we can clearly see that the resulting approximation of the Legendre transform does indeed approach the actual transform of $\mathcal{A}$, getting closer for an increasing amount of samples. Note that we do however obtain an increasing number of points for each edge in the spatial domain if we increase the sample-rate. This in turn means that we have to run trough all these points for the calculation of the slopes of $\mathcal{A}$ when only a small amount of points will

[1] fig.3.3(b) is an enlarged segment of the transforms found in fig.3.3(a)

actually be of any use; we will find the same slope-value $k-1$ times for an edge with k sample-points, clearly decreasing efficiency.

So, when actually using sampling on a 'simple' discrete function, we find that we have to use a sample-rate which will be sufficiently accurate to retain as much of the original features as possible and that the apparent simplicity of a discrete function can be deceptive, resulting in a increase of the number of calculations that we need to obtain the slopes which in turn adds to the number of calculations needed to compute the (approximated) transform using the proposed algorithm. Hence, the use of equidistant sample-points does not seem to be the ideal approach for handling discrete functions; we will need to find an other way to find the global maximum.

Chapter 4

Linearity using Local Knowledge

In the previous chapters we have always focused our attention on computing the Legendre transform using *global* maximum computation. It turns out that we can actually determine this maximum using only *local* knowledge *without* the use of sampling but by using the vertices and edges that build up the discrete function, reducing the problem to one of a linear computational complexity.

4.1 Slope Support Index

The basic idea behind determining the maximum locally, adapting a method proposed in [8], is to build an index of those points which yield a maximum for a given slope s_j.

Given some arbitrary n-point *concave* discrete function $\mathcal{C}$, we can obtain the m slopes ω using eq.(2.6). Note that

$$\omega_1 > \omega_2 > \cdots > \omega_m$$

Suppose we have a set of slopes $s_{1:p}$ for which we want to compute the Legendre transform; we denote this as Σ_ω. Since the sequence $(\omega_i)_{i=1:m}$ is decreasing, we can easily determine an index of vertices of $\mathcal{C}$ using slopes $s \in \Sigma_\omega$ such that a slope s_j will support $\mathcal{C}$ at vertex k; we will denote this set of indices as *SSI* (Slope Support Index). Formalizing this concept yields

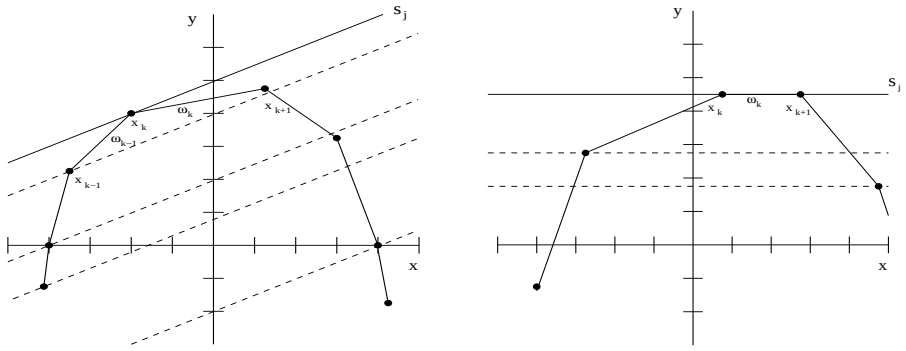
Lemma:

(i) If $s > \omega_1,\ SSI(s) = \{1\}$

(ii) If $\omega_{k-1} > s > \omega_k,\ SSI(s) = \{k\}$

(iii) If $\omega_k = s,\ SSI(s) = \{k, k+1\}$

(iv) If $\omega_m > s,\ SSI(s) = \{n\}$



(a) Slope s_j supports discrete function at (x_k, y_k).

(b) Slope s_j supports discrete function at $\{(x_k, y_k), (x_{k+1}, y_{k+1})\}$.

Figure 4.1: Obtaining a global maximum using local knowledge.

In fact, we have already done something similar when we noted that a vertex of in the spatial domain gets transformed to an edge in the Legendre domain[1], but now we are interested in the vertex itself given some slope s_j i.e. we want to find the vertex x_k for which

$$s_j = \lambda\omega_{k-1} + (1-\lambda)\omega_k \quad \textit{with } 0 \leq \lambda \leq 1 \tag{4.1}$$

is valid, resulting in an index $SSI(s_j) = \{k\}$. Note that this can be the only valid vertex since

$$\omega_1 > \cdots > \omega_{k-1} \geq s_j \geq \omega_k > \cdots > \omega_m$$

Also note that multiple slopes are valid support for $\mathcal{C}$ at (x_i, y_i) using eq.(4.1); we do however still have to prove that $y_k - s_j x_k$ is the maximum for s_j; parts (i) and (iv) of the lemma are straightforward, hence proof will be omited.

Proof Clause (ii) Assume that $\omega_{k-1} > s_j > \omega_k$. The first inequality of (ii) can be written as

$$\frac{y_k - y_{k-1}}{x_k - x_{k-1}} > s_j$$

implying that

$$y_k - s_j x_k > y_{k-1} - s_j x_{k-1}$$

Moreover, since the sequence $(\omega_i)_{i=1,..,m}$ is decreasing, we know that $\omega_{k-2} > \omega_{k-1}$, implying that $y_{k-1} - s_j x_{k-1} > y_{k-2} - s_j x_{k-2}$. Hence we can write the sequence ω_i for $i = 2, \cdots, k-1$ as

$$y_k - s_j x_k > y_{k-1} - s x_{k-1} > \cdots > y_1 - s_j x_1 \tag{4.2}$$

[1]See section 2.2.

The second inequality can be done in a similar fashion:

$$s_j > \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

such that

$$y_k - s_j x_k > y_{k+1} - s_j x_{k+1} > \cdots > y_m - s_j x_m \tag{4.3}$$

for $i = k, \cdots, m$. Using inequalities (4.2) and (4.3) yields $y_k - s_j x_k$ as the strict maximum in the sequence, resulting in an index: $SSI(s_j) = \{k\}$.

□

Proof Clause (iii)

$$\frac{y_{k+1} - y_k}{x_{k+1} - x_k} = s_j$$

implies that

$$y_{k+1} - s_j x_{k+1} = y_k - s_j x_k$$

We can combine this with the previous proof, hence using the notion that the sequence ω_i is decreasing, so,

$$y_{k+1} - s_j x_{k+1} = y_k - s_j x_k > y_{k-1} - s_j x_{k-1} > \cdots > y_1 - s_j x_1$$

and

$$y_k - s_j x_k = y_{k+1} - s_j x_{k+1} > \cdots > y_n - s_j x_n$$

□

In fig.4.1(a) we see a graphical representation of clause (ii) using a positive slope s_j such that $\omega_{k-1} > s_j > \omega_k$. We can clearly see that s_j does indeed give us the strict maximum $y_k - s_j x_k$ and that

$$\forall i \neq k \text{ with } i = 1, \cdots, m: \quad y_k - s_j x_k > y_i - s_j x_i$$

A similar construct holds for fig.4.1(b), depicting clause (iii). It has to be noted that the only index needed in the actual programming for this clause is x_k; x_{k+1} is redundant.

Calculating the index for *convex* discrete functions can be done in much the same way, we only have to alter the 'greater than' in clauses (i), (ii) and (iv) of the lemma and use 'smaller than', hence we will omit any further proof.

Slope Support Index

support*(C,S)*
with
the set of slopes of discrete function C; m elements
the slopes S with $S = \Sigma_\omega$; v elements

note: Take the input to be concave.

$j = 1$;

while $S(j) \geq C(1)$
$SSI(j) = \{1\},\ j = j + 1$;
end

for $i = 2 : m$;
while $S(j) \geq C(i)$ & $j < v$
$SSI(j) = \{i\},\ j = j + 1$;
end
end

% Index any remaining slopes in Σ_ω
for $w = 1 : v - j$;
if $S((j-1)+w) \geq C(m)$
$SSI((j-1)+w) = \{m\}$;
else
$SSI((j-1)+w) = \{m+1\}$;
end
end

4.2 Index-based Legendre Transform

Now we can combine the *Slope Support Index* from the previous section with the concept proposed in section 2.2.2.

4.2.1 A Single Discrete Function

In section 2.1.3 and 2.2.3, it was noted that the inverse of a Legendre transform of a discrete function has to yield the original discrete functions. So, before looking at contact, we have to look at the operations for a single discrete function and show that this does indeed happen. In order to do this, we have to resolve the first issue noted in section 2.2 – the loss of data due to the transform and its inverse.

Handling Infinity

For a possible solution, we have to take another look at section 2.2.2. By proposing to define a 'sense of infinity', we would also 'create' 2 additional

data-points. But we have to be more specific; for concave discrete functions, we can use the construct proposed in that section; for convex discrete functions we simply have to change the sign in front of the infinity-term. In fact, we have to be careful about using the actual infinity-term in the algorithm. It poses a problem when we want to apply the inverse Legendre transform on discrete data; if we choose to use the first and last vertex of the function as a double-valued point, we cannot avoid dividing by 0. A possible solution would be to use a small constant δ in such a way that we 'create' a new first vertex with

$$v_{x_0} = v_{x_1} - \delta$$

and a new last vertex with

$$v_{x_{n+1}} = v_{x_n} + \delta$$

This however does not solve the problem either since using infinity-terms in eq.(2.8) results in infinity, hence crucial data on the first and last vertex are lost.

In fact, the last solution can be used if we simply 'redefine' infinity in such a way that it can be used in the algorithm, hence using some high-value constant. Let's denote this by $\widetilde{\infty}$. It has to be noted that there are a number of factors influencing this constant. Hence, we need to check the dimensions of the input and slope and adjust it accordingly. Note that the 'maximum' Legendre-value possible for any given concave discrete function is

$$\check{\mathcal{L}} = y_{max} - \min\big((\omega_{min} \cdot x_{max}), (\omega_{max} \cdot x_{min})\big)$$

Also note that y_{max} need not be connected to the slopes and extreme x-values in any way. This does however mean that $\check{\mathcal{L}}$ is *at least* the size of the *actual* Legendre-maximum $\mathcal{L}_{max}$ of the discrete function (see fig.4.2).

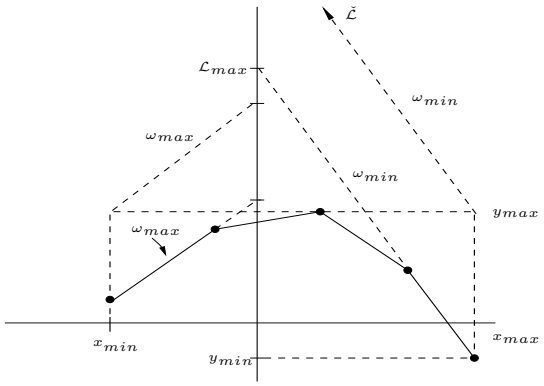


Figure 4.2: Obtaining the 'maximum' Legendre-value.

Convex discrete functions can be done in much the same way for the 'minimum' Legendre-value with some small alterations.

To use this construct as straightforward as possible we can use one single value and define the other two in terms of the first one. For instance

$$\widetilde{\infty} = 1.0^6;\ \delta = \frac{1}{\widetilde{\infty}};\ \omega_{\widetilde{\infty}} = \frac{\widetilde{\infty}}{\delta} = \widetilde{\infty}^2$$

Now we can simply check each of the three variables using the data from the discrete function and its slopes; if one (or more) of these fails, we can easily update the $\widetilde{\infty}$-term, thereby adjusting the other two values as well. We can then use these values in constructing our final transform with an included infinity, resulting in similar transforms as seen in section 2.2.2, so

$$x_0 = x_1 - \delta;\ y_0 = y_1 - \widetilde{\infty}\ \rightarrow\ \mathcal{L}_{\widetilde{\infty}}(left) = y_0 - \omega_{\widetilde{\infty}} \cdot x_0 \tag{4.4}$$

The same can be done for $\{x_{n+1}, y_{n+1}\}$.

$$x_{n+1} = x_n + \delta;\ y_{n+1} = y_n - \widetilde{\infty}\ \rightarrow\ \mathcal{L}_{\widetilde{\infty}}(right) = y_{n+1} - (-\omega_{\widetilde{\infty}}) \cdot x_{n+1} \tag{4.5}$$

For convex discrete functions we only have to alter the subtraction for y_0 and y_{n+1} into an addition and swap $\omega_{\widetilde{\infty}}$ for -$\omega_{\widetilde{\infty}}$ and vice versa. This way, we don't have to worry about the convexity of the function when checking the dimensions.

Now we can focus on proving what we set out to do at the beginning of this section. For instance, take discrete function $\mathcal{A}$ from fig.3.1 – computing the transform using *SSI* yields 4 indices, all resulting from clause (iii) of the lemma in section 4.2; Σ_ω only needs to contain the slopes which are in $\mathcal{A}$ to compute the Legendre transform i.e. $\Sigma_\omega = \omega_{\mathcal{A}}$.

Σ_ω	3	1	-1	-3
SSI	1	2	3	4

The maximum for each slope $s \in \Sigma$ has already been incorporated in the index, therefore computation of the Legendre value is quite straightforward:

$$\forall s_i \in \Sigma_\omega :\quad \mathcal{L}[\mathcal{A}](s) = y_{SSI(s_i)} - s_i \cdot x_{SSI(s_i)} \tag{4.6}$$

for $i = 1, \cdots, h$ with $h = |\Sigma_\omega|$. We can rewrite eq.(2.7) in a similar way to eq.(4.6) to compute the inverse such that

$$\forall x \in \mathcal{X} :\quad \mathcal{L}^{-1}[\mathcal{L}[\mathcal{A}]](x) = \mathcal{L}[\mathcal{A}](s_i) + x_i \cdot s_i \tag{4.7}$$

for $i = 1, \cdots, h-1$. Note that we actually added the left and right $\omega_{\widetilde{\infty}}$ using the relative infinity-method, so we have 6 slopes in Σ_ω. We do however need to apply eq.(2.8) to obtain the x-values, yielding 5 ($= h - 1$) points. Now, using eq.(4.7), we get

x	-1	0	1	2	3
y	0	3	4	3	0

such that we do indeed obtain our original discrete function $\mathcal{A}$.

4.2.2 Colliding Discrete Functions

Before proceeding with 2 contacting discrete functions, we will need to focus our attention on the second issue noted in section 2.2.3 – the possibility of differing slope-values ω.

Combining Discrete Functions

Take the discrete functions $\mathcal{A}$ and $\mathcal{B}$ to have no common slopes i.e.

$$\omega_{\mathcal{A}} \cap \omega_{\mathcal{B}} = \emptyset$$

By simply taking the union of both sets, we obtain a compound and unique set of values Σ_ω. We can then use the slopes in Σ_ω to calculate the Legendre values of both $\mathcal{A}$ and $\mathcal{B}$ and do a simple addition of these values for each slope in Σ_ω. Note that this approach is valid given the properties of a discrete function described in section 2.2; edges get transformed to (a cone on) a vertex and vertices are transformed to edges. This is due to the 'valid' slopes which pass through each of these vertices even though they might not be directly apparent from the discrete function[2]. This means that for all slopes $s \in \Sigma_\omega$, we can always find a vertex i for any discrete function for which

$$s_j = \lambda\omega_{i-1} + (1-\lambda)\omega_i \quad \textit{with} \quad 0 \leq \lambda \leq 1 \tag{4.8}$$

holds. The only two possible exceptions are the first and last vertex of the discrete function. Note that we can however find these vertices for a slope s_j using

$$s_j > \omega_1 \quad or \quad s_j < \omega_n \tag{4.9}$$

for concave discrete functions; the converse holds for the convex version. In section 2.2.3 we already noted that two discrete functions can only 'kiss' when sharing a slope. So, if a slope s_j is present in either function, we will see the two edges with slope s_j 'kiss' and slide alongside each other. If s_j is only directly found in one of them, we will find that the edge of one discrete function will 'kiss' and slide along the vertex v of the other one, given that v is valid for eq.(4.8) or (4.9). Note that we can omit any further occurrences of the same value in Σ_ω for a non-empty intersection; we only need to use a slope once. Simply adding the Legendre values for each slope results in the data needed to compute the tangential dilation of $\mathcal{A}$ and $\mathcal{B}$ using the inverse transform.

Take the discrete functions $\mathcal{A}$ and $\mathcal{B}$ as shown in fig.4.3 be the colliding bodies. Note that the discrete functions used are treated similarly to the examples in section 2.1.3, hence we have to keep in mind that *one* of these functions has its *transpose* in the actual contact; define $\mathcal{B}^*$ to be $\mathcal{B}$'s transpose [3].
In the same section we chose to fix the *reference-point* in the origin; the 'X' used in fig.4.3 has no real purpose other than to enhance readability in the following figures.

[2] See section 2.2

[3] Choice of function is arbitrary; sometimes, choosing one over the other does however add to readability of figures of resulting contact

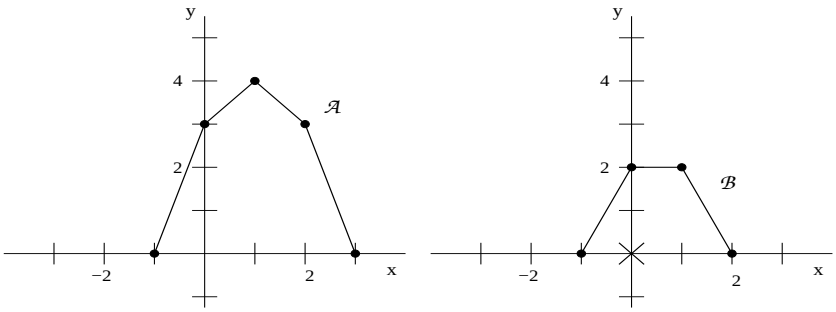
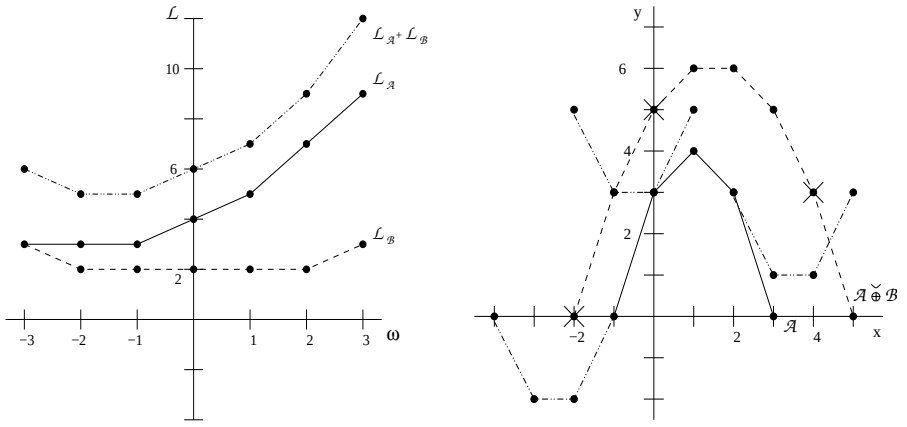


Figure 4.3:

Given the compound, unique set Σ_ω, we need to calculate the index for all slopes $s \in \Sigma_\omega$ using the slopes of the separate discrete functions. Hence, a set of indices is obtained for both $\mathcal{A}$ and $\mathcal{B}$; denote the sets as $SSI_\mathcal{A}$ and $SSI_\mathcal{B}$. Note that, since we use Σ_ω for both functions,

$$|SSI_\mathcal{A}| = |SSI_\mathcal{B}|$$

Using eq.(4.6) yields $\mathcal{L}_\mathcal{A}$ and $\mathcal{L}_\mathcal{B}$ (see fig.4.4(a)).



(a) (b)

Figure 4.4: Legendre transforms and the resulting contact $\mathcal{A}\breve{\oplus}\mathcal{B}$.

Now, we can simply add up the Legendre values obtained using eq.(4.6), yielding $\mathcal{L}_\mathcal{A}+\mathcal{L}_\mathcal{B}$. The resulting contact $\mathcal{A}\breve{\oplus}\mathcal{B}$ in fig.4.4(b)[4] clearly shows all of the slopes in Σ_ω and is similar to the last example in the section on concave functions, being tighter on one side and wider on the other due to the functions' position in the frame relative to the reference-point ('X'). In fact, we find that if we compute the contact of a convex and a concave discrete function, as seen in the

[4] $\mathcal{A}\breve{\oplus}\mathcal{B}$ represented by dashed line.

previous example, we always find a result which is single-valued (see [4]), so for every valid x in the domain, we only find 1 corresponding y-value; we're simply sliding a convex discrete body, approaching from above, along the outside of a concave discrete body or vice versa, with the concave body approaching from below.

This however does not strictly hold when combining two discrete functions with the *same* 'shape'. Using 2 other, simple functions (fig.4.5), we obtain a contact for which the statement made above does not hold.

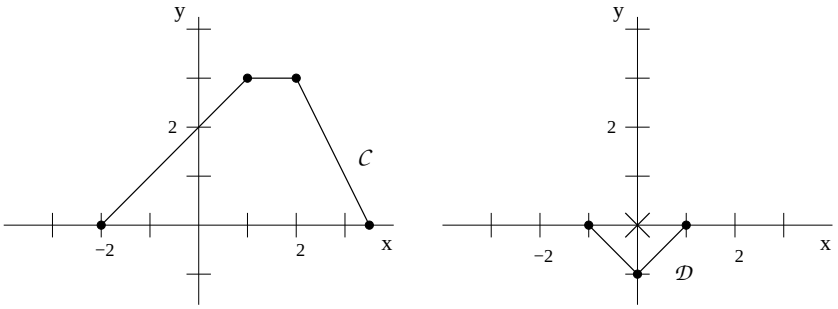


Figure 4.5:

$\mathcal{D}^*$ slides nicely along the first edge of $\mathcal{C}$, both having the same first slope, but when sliding along the second one, $\mathcal{D}$ also intersects with the third edge; to slide back along the third edge, $\mathcal{C}\breve{\oplus}\mathcal{D}$ has to fold back; the last vertex of $\mathcal{D}^*$ is the only point at which $\mathcal{D}^*$ can 'kiss' $\mathcal{C}$ due to clause (iv) of the lemma.

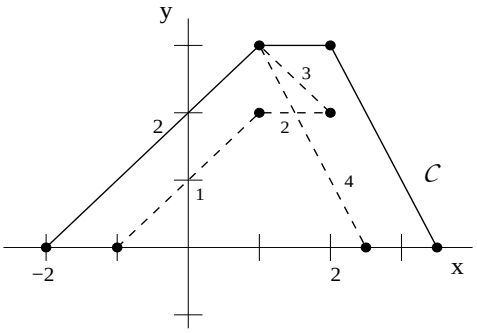


Figure 4.6: $\mathcal{C}\breve{\oplus}\mathcal{D}$: Kissing with penetration.

Note that the point of intersection of edge 2 and 4 corresponds to a multiple contact of $\mathcal{D}^*$ with the second and third edge of $\mathcal{C}$.

Another feature which we will only find for certain combinations of 2 concave/ convex discrete functions is the appearance of an edge (or edges) which at first does not seem to belong in the resulting contact.

Take function $\mathcal{C}$ as used in the previous example and take $\mathcal{E}$ to be quite similar

to $\mathcal{D}$ but with higher slope-values[5]. If we were to omit the first and last edge of $\mathcal{C}\breve{\oplus}\mathcal{E}$, the resulting contact would seem to fit much better. But if we were to use this altered version, denoted by $\mathcal{I}$, in combination with $\mathcal{C}$, we find that

$$\mathcal{L}^{-1}\big[\mathcal{L}[\mathcal{C}+\mathcal{I}](\omega)\big](x) \neq \mathcal{E}$$

but a single point (0,3), confirming that $\mathcal{I}$ is indeed a simple linear translation of $\mathcal{C}$. Hence, we find that these edges do indeed belong to the actual contact, so sliding the first edge of $\mathcal{E}^*$ along the first vertex of $\mathcal{C}$ and the second edge along the last vertex of $\mathcal{C}$.

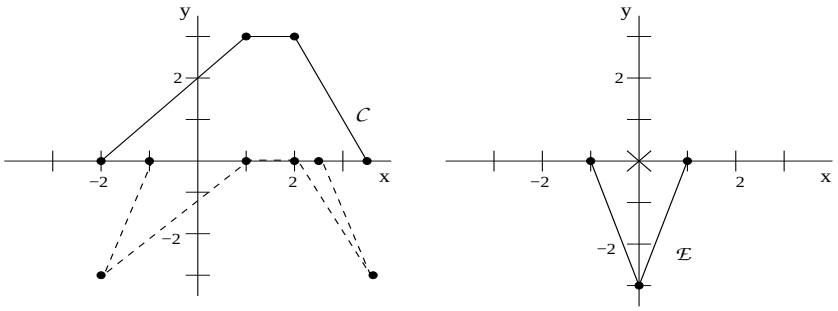


Figure 4.7: $\mathcal{C}\breve{\oplus}\mathcal{E}$: Kissing with penetration(2).

4.3 Complexity

The linear complexity of the algorithm using on this *Slope Support Index* is mainly based on the sorting of two sequences.

Take some discrete function $\mathcal{C}$ to have n points and m unique slopes with $m \approx n$. We require $\mathrm{O}(n)$ operations for calculating the slopes and to determine the convexity of the function. Since we know that Σ_ω and the sequence of slopes $(\omega_i)_{i=1,\ldots,m}$ for $\mathcal{C}$ are the same, we need $\mathrm{O}(m)$ operations for calculating the *SSI* of $\mathcal{C}$ since both sequences are either increasing or decreasing, depending on the convexity. This way, the indices will always be sorted in ascending order and makes the actual programming much more straightforward. This leaves us with $\mathrm{O}(m)$ calculations for the actual Legendre transform, yielding an $\mathrm{O}(n+m)$ complexity at the end.

To show that it is indeed a worst-case linear algorithm, we can use an increasingly accurate discrete representation of a concave/convex second degree polynomial.

[5]See fig.4.7

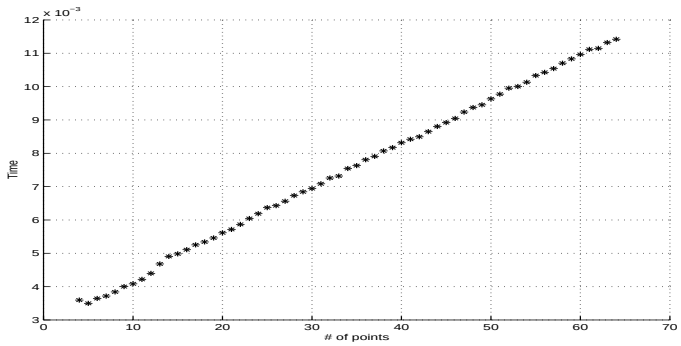


Figure 4.8: Computing Legendre and its inverse for a single discrete function.

The complexity for computing the contact of two discrete functions can be proven in a similar fashion to the Legendre and inverse transform of a single discrete function.

Again, presume a worst-case scenario. This means that the two functions must be (roughly) of the same order in the number of points (n) *and* slopes (m), $m \approx n$. Moreover, the intersection of both sets of slopes should be empty. Hence, Σ_ω will be twice the size of either functions' slope-set. We still need $\mathrm{O}(n)$ calculations to obtain the slopes and the convexity of the discrete functions. To calculate the *SSI* of each discrete function, we need to run through Σ_ω twice. Suppose that Σ_ω has h ($\approx 2m$) elements. This leaves us with $\mathrm{O}(h+m)$ operations needed for indexing each function. At the end, we need a further $\mathrm{O}(h)$ calculations to get the Legendre transform, yielding an $\mathrm{O}(n+m)$ complexity overall[6].

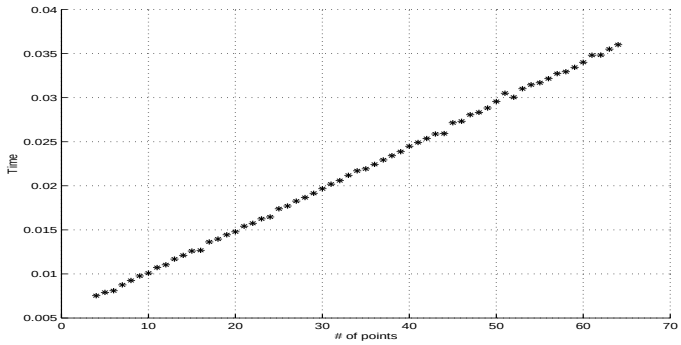


Figure 4.9: Computing contact-boundary of 2 discrete functions.

[6]Benchmark used for fig 4.8 and 4.9: Sun Enterprise 450 (4 X UltraSPARC-II 296MHz)' (4 X UltraSPARC-II cpus) @ 296.0 MHz 1024 MB.

Chapter 5

Non-Convex Discrete Functions

We also want to be able to handle more complex discrete functions in contact computation like the one shown below. We already noted that a non-convex discrete function can be perceived as a set of concave and convex segments pasted together[1]. In fact, one can determine whether a discrete function is concave or convex given the second order derivative σ; we know that the discrete first order derivative can be found using eq.(2.6), hence, the values σ_i of the second order derivative can be obtained using

$$\sigma_i = \Big(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\Big)' = \Big(\frac{\omega_{i+1} - \omega_i}{x_{i+1} - x_i}\Big) \tag{5.1}$$

So, for every discrete function we can determine its convexity:

$$\underline{\text{Def}} : \begin{cases} \text{'concave' for } [x_p, x_q] & \text{if } \sigma_{p,\cdots,q} \leq 0 \\ \text{'convex' for } [x_p, x_q] & \text{if } \sigma_{p,\cdots,q} \geq 0 \end{cases}$$

If we were to use the global maximum approach on a complete non-convex discrete function, global maxima would be found for every slope ω.

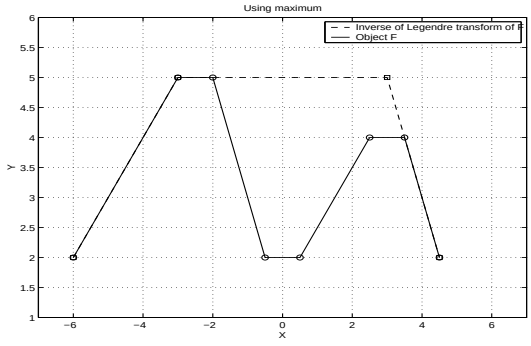


Figure 5.1: Non-convex discrete function $\mathcal{M}$.

[1] See section 2.2.3.

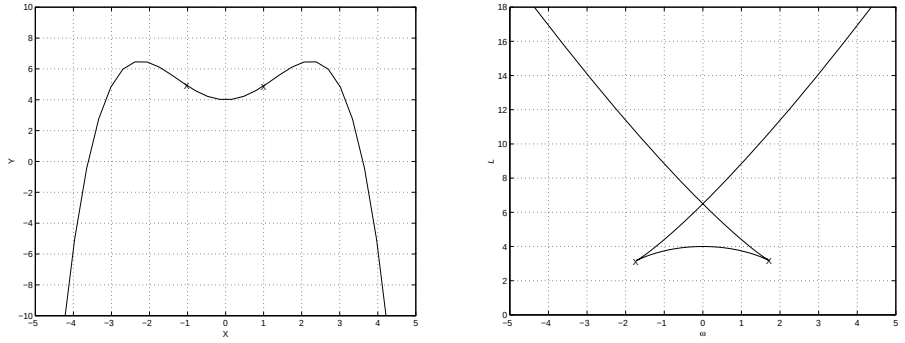
The result of this can also be found in fig.5.1; the inverse of the Legendre transform of $\mathcal{M}$ without using any segmentation yields a *concave hull*, 'destroying' the specific features.

To use the notion of a compound discrete function we need not only segment and re-attach the segments correctly but we also have to see to it that the separate segments, when used in an actual contact with some other body, yield a correct result. But, before we can start looking at colliding discrete functions we need to be able to handle a function like $\mathcal{M}$ in fig.5.1 and be able to show that

$$\mathcal{L}^{-1}\big[\mathcal{L}[\mathcal{M}](\omega)\big](x) = \mathcal{M} \tag{5.2}$$

5.1 Segmentation

In order to obtain an acceptable segmentation of the non-convex discrete functions into a set of concave and convex segments, we have to examine the Legendre transform of continuous functions with similar features.



(a) (b)

Figure 5.2: Transforming a non-convex function.

If we examine the function f in fig.5.2(a) in combination with fig.5.2(b), we find that the points where 5.2(b) folds back ('cusps') are actually the points where the second order derivative of f equals zero. This means that the transform for a similar non-convex discrete function should yield a similar result.

One of the major problems with discrete data when using the second order derivative is that we do not have a continuous change of slopes, hence, the 'switchpoints' in functions like f need not be found for all discrete functions. We have however noted that we can use the second order derivative to determine a discrete functions' convexity. Moreover, if we determine the second order derivative σ using eq.(5.1) of $\mathcal{M}$, found in this chapters' introduction, we find that there are several interesting points.

$\sigma_{\mathcal{M}}$	$-\frac{1}{3}$	-2	$1\frac{1}{3}$	1	$-\frac{1}{2}$	-2

Notice that there are 2 sign-changes in the sequence. The continuous function f actually has the same features; it swaps over from a negative to a positive value for the first 'switchpoint' and vice versa for the second one. Hence, we have found a way to locate the crossing-over of a concave segment to a convex one and vice versa, so $sign(\sigma_i) \neq sign(\sigma_{i+1})$.

An other way to find this cross-over, as noted previously, is when encountering a zero in the second order derivative of a non-convex discrete functions. The problem with actually finding a zero at σ_i, is to be absolutely sure that we do indeed see a sign-change i.e.

$$sign(\sigma_{i-1}) \neq sign(\sigma_{i+1}) \;\&\; (\sigma_{i-1}) \neq 0 \;\&\; (\sigma_{i-1}) \neq 0 \tag{5.3}$$

The problem can be easily illustrated using the concave discrete function $\mathcal{A}$ from chapter 4.

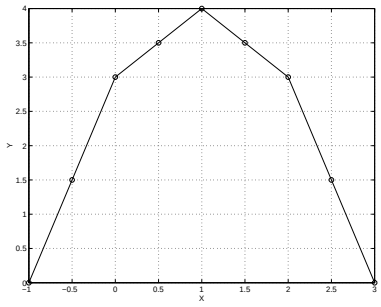


Figure 5.3:

We just added a few vertices to it; note that the placing of the extra vertices, although spaced at equal intervals, do not influence the results given the index-method explained in the previous section; they can be placed at any point, along any edge. The only restrictions are that they should be placed on an edge, thereby not altering the actual discrete function, and that we do not want vertices to occur more than once, hence avoiding division by zero. When looking at the second order derivative of $\mathcal{A}$, we find

$\sigma_{\mathcal{A}}$	0	-4	0	-4	0	-4	0

But looking at $\mathcal{A}$, we can easily see that there's not apparent reason to actually segment it; $\mathcal{A}$ is concave since

$$\forall i: \; \sigma_{\mathcal{A}_i} \leq 0$$

with $i = 1, \cdots, |\omega_{\mathcal{A}}| - 1$. When we elect to use a similar method for $\mathcal{M}$, for instance by adding a vertex somewhere along the second edge, we would see

$\sigma_{\mathcal{M}}$	-$\frac{1}{3}$	0	-4	$1\frac{1}{3}$	1	-$\frac{1}{2}$	-2

hence, we have to keep track of certain features in the sequence. Since multiple zeros might occur when we find several vertices which are valid for a single slope-value, we have to be certain that we actually find the last one. One way to this

is by keeping track of the sign of the last non-zero second order derivative-value; denote this by $\sigma_{\neq 0}$. Now, when we find a zero at σ_i, we only segment $\mathcal{M}$ if

(i) $\sigma_{i+1} \neq 0$

(ii) $sign(\sigma_{\neq 0}) \neq sign(\sigma_{i+1})$

hence specifying eq.(5.3). Note that the procedures presented need to be recursive to correctly segment a non-convex discrete function where needed i.e. we have check the remainder of the function when a segment has been found.

When we look closer at the original version of $\mathcal{M}$ we can actually find more that one way to do the segmentation. One way would be to cut it up at a vertex; the sequence (σ_i) has a sign-change for $i = 3$ and $i = 5$. Let's look at the first one. When examining $\mathcal{M}$ in fig.5.1, we find that the third vertex seems to be good place for segmentation using definition posted at the beginning of this chapter. The problem with segmenting a non-convex discrete function at a *mutual* vertex is the resulting transform that we obtain; consider segmenting $\mathcal{M}$ at the third and fifth vertex; hence we obtain three segments: $\{x_1, \ldots, x_3\}$, $\{x_3, \cdots, x_5\}$ and $\{x_5, \cdots, x_8\}$. We can clearly see that when we compute the Legendre transform for each of these segments, we will end up with three separate transforms which are not connected like the transform of f in fig.5.2(b).

To stay close to the actual transform in the continuous setting, we have find a way to segment a non-convex discrete function in such a way that the combination of the separate segments yields a result similar to that of fig.5.2(b). When re-examining $\mathcal{M}$, we might also have elected to segment the function at the fourth vertex instead of the third; the first segment would still be convex and would not influence the convexity of the second segment; this would require us to alter our definition. We have however already noted that an edge gets transformed to a vertex in section 2.2. We can actually combine these two notions; if we were to segment using *mutual* edges, we would obtain the same point in the Legendre domain for neighbouring segments given their mutual edge. So, for $\mathcal{M}$, given the sign-change for $i = 3$, we get a mutual edge $\{x_3, x_4\}$ for the first and second segment and $\{x_5, x_6\}$ for the second and third segment.

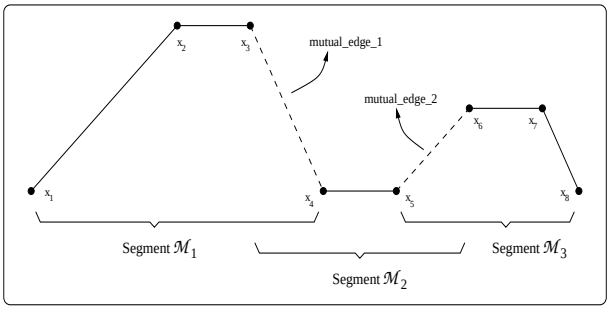
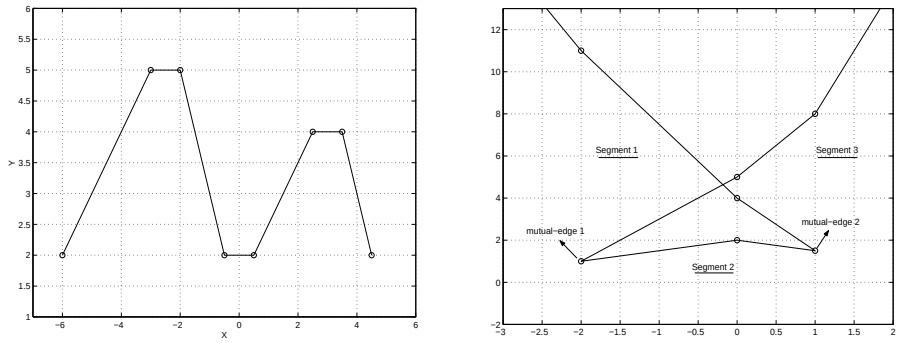


Figure 5.4: Segmenting a non-covex discrete function.

Now that we have our segments $\mathcal{M}_{1,\cdots,3}$, we can apply the index-based Legendre

transform on each segment separately. If we then re-attach these segments using the mutual edge-approach and apply eq.(4.6), we obtain the Legendre transform of $\mathcal{M}$.



(a) (b)

Figure 5.5: Transforming a non-convex discrete function.

If we then apply eq.(4.7) on $\mathcal{L}[\mathcal{M}](\omega)$ [2], we re-obtain $\mathcal{M}$, thereby proving that eq.(5.2) does indeed apply.

5.2 Fusion

When we want to actually use these non-convex discrete functions in a collision with a concave/convex discrete function, we can not simply treat every separate segment of the non-convex function as being a separate discrete function colliding with a concave/convex discrete function and then re-attach the separate transforms at the end. This is mainly due to the use of the mutual edges that we use to do the segmentation, hence, we have to re-examine the construction of our compound, unique set Σ_ω as presented in section 4.3. In fact, we can distinguish 3 separate cases; the two outer segments, so the left and right one and the possible segment(s) situated in between. We will examine the outer segments first.

Consider using the non-convex discrete function $\mathcal{M}$ in combination with some discrete function $\mathcal{C}$. When looking at the first, concave segment of $\mathcal{M}$ we find 3 slopes. We can denote these slopes for segment $\mathcal{M}_1$ as $\omega_{\mathcal{M}_{1:1}}$, $\omega_{\mathcal{M}_{1:2}}$ etc. Using eq.(4.8) and (4.9), we know that we can find a vertex in $\mathcal{M}_1$ such that $\mathcal{C}^*$ 'kisses' $\mathcal{M}_1$. Note that the last edge of segment $\mathcal{M}_1$ is also the first edge of segment $\mathcal{M}_2$ – given the notion that two discrete functions can only 'kiss' when sharing a slope immediately results in the restriction that the only slopes in $\omega_\mathcal{C}$ which are valid for collision with $\mathcal{M}_2$ are slopes greater or equal to the first slope of $\mathcal{M}_2$, so $\omega_{\mathcal{M}_{2:1}}$ $(= \omega_{\mathcal{M}_{1:3}})$. This means that we can only use slopes

[2]Remember that on left the slope $\omega_{\widetilde{\infty}}$ has been added to the first segment and on the right slope $-\omega_{\widetilde{\infty}}$ has been added to last segment.

from $\mathcal{C}$ for which

$$\omega_{\mathcal{C}_i} \geq \omega_{\mathcal{M}_{1:3}}$$

holds when colliding with segment $\mathcal{M}_1$. We can then simply take the unique union of this set with the slopes of $\mathcal{M}_1$ to find Σ_ω for computing the first contact-boundary.

A similar setup can be used for the last segment. The only valid slopes for colliding $\mathcal{C}^*$ with segment $\mathcal{M}_3$ due to segment $\mathcal{M}_2$ are slopes that are smaller or equal to $\omega_{\mathcal{M}_{2:3}}$. This in turn results in the restrictions for valid slopes for segment 2; the only valid slopes in $\mathcal{C}$ are slopes greater or equal to $\omega_{\mathcal{M}_{2:1}}$ and smaller or equal to $\omega_{\mathcal{M}_{2:3}}$.

When formalizing this, we find that valid slopes $(\omega_{\mathcal{C}_i})_{i=1\cdots n}$ of a discrete function $\mathcal{C}$ for constructing Σ_ω with a non-convex discrete function $\mathcal{M}$, consisting of v segments, are those for which the following holds:

- segment $\mathcal{M}_1$ with $\omega_{1,\cdots,r}$
  - 'concave': $\omega_{\mathcal{C}_i} \geq \omega_{\mathcal{M}_{1:r}}$
  - 'convex': $\omega_{\mathcal{C}_i} \leq \omega_{\mathcal{M}_{1:r}}$
- segment $\mathcal{M}_j$ with $\omega_{1,\cdots,s}$ and $1 < j < v$
  - 'concave': $\omega_{\mathcal{M}_{j:1}} \geq \omega_{\mathcal{C}_i} \geq \omega_{\mathcal{M}_{j:s}}$
  - 'convex': $\omega_{\mathcal{M}_{j:1}} \leq \omega_{\mathcal{C}_i} \leq \omega_{\mathcal{M}_{j:s}}$
- segment $\mathcal{M}_v$ with $\omega_{1,\cdots,t}$
  - 'concave': $\omega_{\mathcal{C}_i} \leq \omega_{\mathcal{M}_{v:1}}$
  - 'convex': $\omega_{\mathcal{C}_i} \geq \omega_{\mathcal{M}_{v:1}}$

5.3 Complexity

We can derive the computational complexity for a non-convex discrete function in a similar way to that of a concave/convex discrete function in section 4.3. In fact, we can easily extend the proof in that section for some non-convex function $\mathcal{C}$ with N vertices and M slopes: we need $O(N)$ to determine slopes and segmentation. Assume that $\mathcal{C}$ has k segments, each containing n vertices and m slopes ($m \approx n$), such that

$$\sum_{i=1}^{k} |C_i| = N + 2(k-1); \ \sum_{i=1}^{k} |\omega_{C_i}| = M + (k-1)$$

Note that the additional terms $2(k-1)$ and $(k-1)$ are due to the use of the $(k-1)$ mutual edges. We already know that we need $O(n+m)$ calculations for a concave/convex discrete functions i.e. a segment of $\mathcal{C}$. Hence, using all k segments results in

$$\sum_{i=1}^{k} O(|C_i| + |\omega_{C_i}|) \ = \ O(\sum_{i=1}^{k} |C_i| + \sum_{i=1}^{k} |\omega_{C_i}|)$$

yielding an $O(N + M + 3(k-1))$ worst-case complexity.

The complexity for computing the contact of $\mathcal{C}$ and a concave/convex discrete function $\mathcal{D}$ can be done along the same lines. Given the proof in section 4.3, we know that we have an $O(n+m)$ worst-case complexity for computing two contacting concave/convex discrete functions i.e. the contact of a segment of $\mathcal{C}$ and $\mathcal{D}$. In order to obtain an upper bound for computing the contact of $\mathcal{C}$ and $\mathcal{D}$, the intersection of the slopes of *each* segment of $\mathcal{C}$ and $\mathcal{D}$ has to be empty i.e.

$$\forall i : \ \omega_{\mathcal{C}_i} \cap \omega_{\mathcal{D}} = \emptyset$$

with $i = 1, \cdots, k$, combined with the demand that no slopes are lost in fusion[3]. When using all k segments with N points and M slopes, we simply need to sum over the k separate contacts, yielding an $O(N + M + 3(k-1))$ computational complexity.

[3]See section 5.2.

Chapter 6

Polygons

Now that we can handle concave, convex and non-convex discrete functions, we can extend this to $2D$ objects like polygons. A polygon is assumed have a circular data-structure

A polygon $\mathcal{P}$ is an ordered list of N vertices of which it is assumed that each pair of vertices $\{v_i, v_{(i \ MOD \ N)+1}\}$ with $1 \leq i \leq N$ determines an edge of $\mathcal{P}$.

$\mathcal{P}$ is assumed to be closed **and** not self-intersecting. Furthermore, we assume that a polygon, representing some geometrical object/body, has a clockwise orientation – if we traverse along the edges of the polygon, we will always find the polygon on the right-hand side. Note that, in order to use a polygon $\mathcal{P}$ using the discrete linear Legendre transform, we need to split it into an upper and a lower part; the transform only applies for $\Omega \rightarrow \mathbb{R}$. In doing so, we obtain similar objects to those used in previous chapters.

6.1 Splitting Polygons

One way to split $\mathcal{P}$ is by first acquiring the left-most and right-most vertex – using these two vertices, we can split the list of vertices V into two smaller lists. Assume that we have a polygon $\mathcal{P}$ with N vertices described by an ordered list V such that the left-most vertex is the first vertex in the list[1]. Take v_k to be the right-most vertex of $\mathcal{P}$, hence, we obtain the two parts of $\mathcal{P}:\ \{v_1, \cdots, v_k\}$ and $\{v_k, \cdots, v_N, v_1\}$.

Note that we restrict the data of a polygon such that we exclude self-intersections and 'fold-backs' – it is built up out of an upper and a lower discrete function, hence the x-values need to be increasing i.e. $x_i < x_{i+1}$. This does however mean that we have to flip the sequence $\{v_k, \cdots, v_N, v_1\}$ around such that the restriction holds, resulting in $\{v_1, v_N, \cdots, v_k\}$.

Since we made the assumptions concerning clockwise orientation and self-intersection, we have a trivial solution for determining the upper and lower

[1] In the algorithm the general case has been implemented.

half of the polygon; denote these parts as $\mathcal{P}_{up}$ and $\mathcal{P}_{down}$. This results in $\mathcal{P}_{up} = \{v_1, \cdots, v_k\}$ and $\mathcal{P}_{down} = \{v_1, v_N, \cdots, v_k\}$.

6.2 Colliding Polygons

Consider some polygon $\mathcal{P}$ and $\mathcal{Q}$ representing two rigid bodies with $\mathcal{Q}$ moving *around* (obstacle) $\mathcal{P}$. In order to the compute the contact-boundary $\mathcal{P}\breve{\oplus}\mathcal{Q}$ we have to combine the right parts of each polygon in the Legendre transform i.e. $\mathcal{Q}_{up}$ can only kiss $\mathcal{P}_{down}$ when $\mathcal{Q}$ approaches $\mathcal{P}$ from below and $\mathcal{Q}_{down}$ can only kiss $\mathcal{P}_{up}$ when $\mathcal{Q}$ approaches $\mathcal{P}$ from above.

But before we can proceed, we have to examine the possible restrictions and problems which did not surface when treating discrete functions. We have already noticed in chapter 4 that we are limited in the control of the contact – we are not able to influence wether a discrete function $\mathcal{B}^*$ approaches a discrete function $\mathcal{A}$ from above or below – it is simply based on the Legendre transform in combination with the convexity of the functions used.

Take $\mathcal{A}$ to be concave; if $\mathcal{B}^*$ is *convex*, we find that the resulting contact-boundary, like the one in fig.4.4(b), is concave such that $\mathcal{B}^*$ seems to be approaching $\mathcal{A}$ from above. If $\mathcal{B}^*$ is *concave* however, we obtain a contact-boundary similar to fig.4.6 or 4.7 where $\mathcal{B}^*$ seems to be approaching $\mathcal{A}$ from below. Hence, if we want to want to slide a polygon $\mathcal{Q}$ along a polygon $\mathcal{P}$, we have to be sure that if $\mathcal{Q}_{up}$ is *concave*, $\mathcal{P}_{down}$ needs to be convex or non-convex such that the resulting contact-boundary 'suggests' that $\mathcal{Q}$ slides along the outside of the bottom of $\mathcal{P}$. The converse is true when $\mathcal{Q}_{up}$ is *convex* i.e. $\mathcal{P}_{down}$ needs to be concave or non-convex. A similar combination-restriction can be formulated for $\mathcal{Q}_{down}$ and $\mathcal{P}_{up}$.

This leaves us with tackling the problem which was last noted in section 4.3: the appearance of seemingly invalid edges. Note that this problem only arises for certain cases of *concave vs. concave* or *convex vs. convex*: for two concave discrete functions it occurs when the first slope(s) of the 'moving' function is/are greater than the first slope of the 'fixed' function that it collides with and/or when the last slope(s) of the first function is/are smaller then the last slope of the second function. The converse holds for two convex discrete functions.

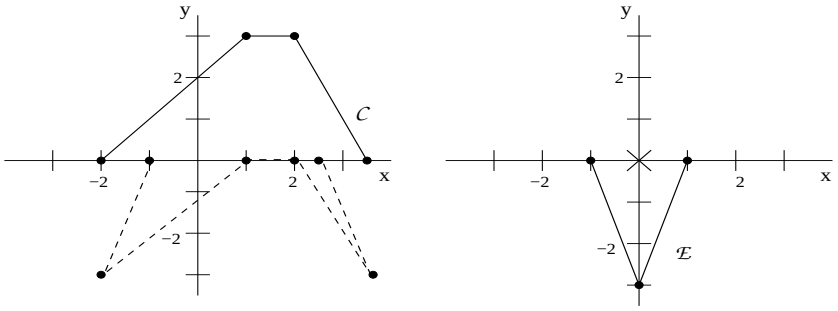


Figure 6.1: $\mathcal{C}\breve{\oplus}\mathcal{E}$: Kissing with penetration(2).

If we want to obtain a contact-boundary of $\mathcal{Q}$ sliding along the outside of $\mathcal{P}$, we have to find that

$$v_1(\mathcal{P}_{up} \breve{\oplus} \mathcal{Q}_{down}) = v_1(\mathcal{P}_{down} \breve{\oplus} \mathcal{Q}_{up})$$
$$v_s(\mathcal{P}_{up} \breve{\oplus} \mathcal{Q}_{down}) = v_t(\mathcal{P}_{down} \breve{\oplus} \mathcal{Q}_{up})$$

with v_s being the last vertex of the upper contact and v_t being the last vertex of the lower contact. Of course we do want to be able to use the objects in fig.6.1 as parts of a polygon. One way to resolve the problem is by applying a similar 'sense of infinity' to the upper and lower parts of one of the polygons, as seen in fig.6.2, and use the updated parts as input for the actual Legendre algorithm.

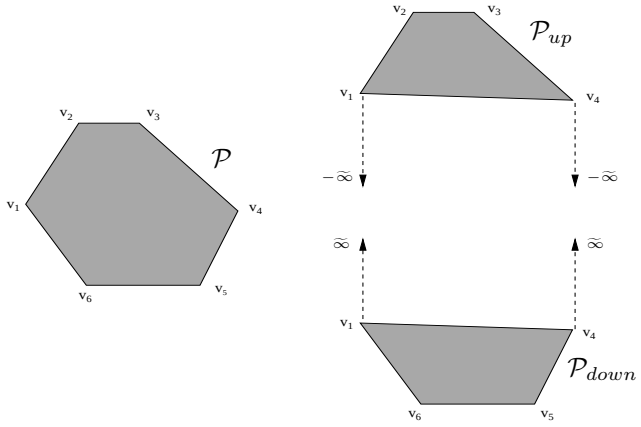


Figure 6.2: Splitting a polygon.

Consider $\mathcal{C}$ to be the lower part of $\mathcal{P}$, so $\mathcal{C} = \mathcal{P}_{down}$ and $\mathcal{E}$ to be the transposed upper part of $\mathcal{Q}$. If we then apply the same concept as described above to update $\mathcal{C}$, we obtain

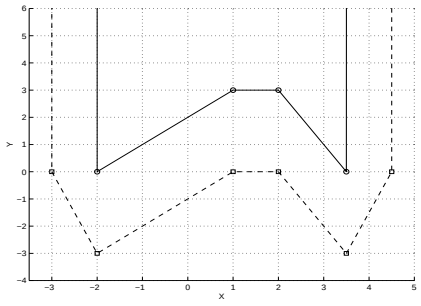
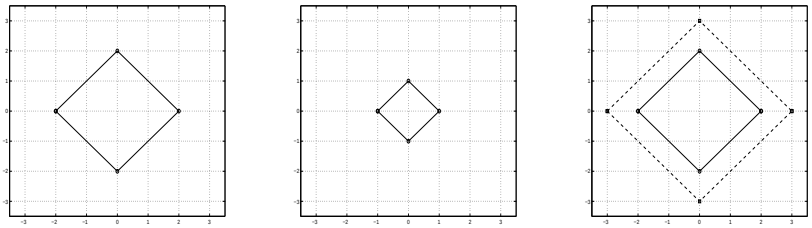


Figure 6.3: Resolving 'fold backs'.

If we then omit the first and last edge of $\mathcal{C} \breve{\oplus} \mathcal{E}$, we find that we can slide $\mathcal{E}^*$ along $\mathcal{C}$ i.e. $\mathcal{Q}_{up}$ slides along $\mathcal{P}_{down}$, with $\mathcal{Q}_{up}$ approaching $\mathcal{P}_{down}$ from below. Note that we no longer have to use convex upper parts and concave lower parts for the 'fixed' polygon, in this case polygon $\mathcal{P}$, due to this update, they will be treated as non-convex.

Now we are at the point where we can look at the bigger picture. To simplify the use and readability of the resulting contact-boundaries, we have built up the algorithm in such a way that the first polygon, $\mathcal{P}$ is considered to be the representation of the 'fixed' object in the real world and the second one, $\mathcal{Q}$, to represent the 'moving' object. After both polygons have been spit into their upper and lower parts, we can apply the translation noted in section 2.1.3 on $\mathcal{Q}_{up}$ and $\mathcal{Q}_{down}$, so point-mirroring both parts in the origin. This way, we need not worry about any translations that we have to do at the end (WYSIWYG)[2].

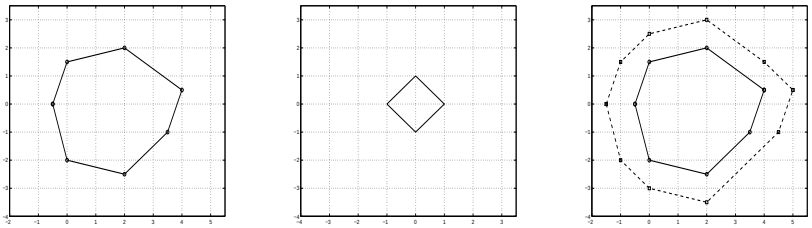
For the time being we will use polygons representing the 'moving' object with a concave upper and convex lower part. Consider using two simple diamond-shaped polygons $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{B}$;



(a) $\mathcal{P}_\mathcal{A}$ (b) $\mathcal{P}_\mathcal{B}$ (c) $\mathcal{P}_\mathcal{A} \breve{\oplus} \mathcal{P}_\mathcal{B}$

Figure 6.4: Two kissing diamonds.

The result is quite straightforward and needs no further explanation. If we alter the shape of $\mathcal{P}_\mathcal{A}$ and add some additional vertices we get a polygon $\mathcal{P}_\mathcal{C}$ which still has a concave upper and a convex lower part but is more complex than $\mathcal{P}_\mathcal{A}$. Computing the contact-boundary of $\mathcal{P}_\mathcal{C}$ and $\mathcal{P}_\mathcal{B}$ yields fig.6.5(c).



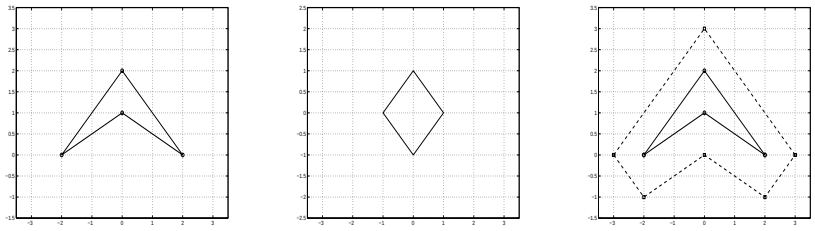
(a) $\mathcal{P}_\mathcal{C}$ (b) $\mathcal{P}_\mathcal{B}$ (c) $\mathcal{P}_\mathcal{C} \breve{\oplus} \mathcal{P}_\mathcal{B}$

Figure 6.5: $\mathcal{P}_\mathcal{B}$ kissing $\mathcal{P}_\mathcal{C}$.

Altering the position of $\mathcal{P}_\mathcal{B}$ using some linear translation T will effectively have the same result as seen in section 2.1: it will simply translate $\mathcal{P}_\mathcal{C} \breve{\oplus} \mathcal{P}_\mathcal{B}$ using the inverse linear translation $-T$.

[2]What You See Is What You Get.

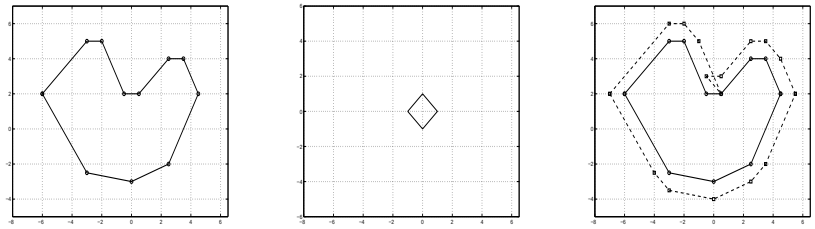
We already noted that we can also use the 'fixed' polygon with a concave lower (or convex upper) part using the added 'sense of infinity'. So if we compute the contact of polygon $\mathcal{P}_\mathcal{B}$ with $\mathcal{P}_\mathcal{D}$ we get fig.6.6(c).



(a) $\mathcal{P}_\mathcal{D}$ (b) $\mathcal{P}_\mathcal{B}$ (c) $\mathcal{P}_\mathcal{D} \breve{\oplus} \mathcal{P}_\mathcal{B}$

Figure 6.6: $\mathcal{P}_\mathcal{B}$ kissing $\mathcal{P}_\mathcal{D}$.

This leaves us with treating polygons with a non-convex upper (and/or lower) part like polygon $\mathcal{P}_\mathcal{E}$ in fig.6.2.



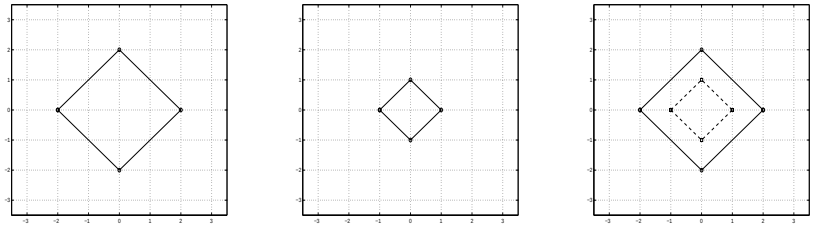
(a) $\mathcal{P}_\mathcal{E}$ (b) $\mathcal{P}_\mathcal{B}$ (c) $\mathcal{P}_\mathcal{E} \breve{\oplus} \mathcal{P}_\mathcal{B}$

Figure 6.7: $\mathcal{P}_\mathcal{B}$ kissing $\mathcal{P}_\mathcal{E}$ with penetration.

When $\mathcal{P}_\mathcal{B}$ slides along the third edge of the upper part of $\mathcal{P}_\mathcal{E}$ it also intersects with the fourth and fifth edge; $\mathcal{P}_\mathcal{E} \breve{\oplus} \mathcal{P}_\mathcal{B}$ has to fold back in order to slide $\mathcal{P}_\mathcal{B}$ along the fourth and fifth edge and proceed along the remainder of $\mathcal{P}_\mathcal{E}$.

The other way to interpret the collision of two polygons was already hinted at at the beginning of this section – colliding polygons $\mathcal{P}$ and $\mathcal{Q}$ with $\mathcal{Q}$ moving along the *inside* of $\mathcal{P}$. One possible interpretation of the resulting contact-boundary is by viewing $\mathcal{P}$ as an environment in which $\mathcal{Q}$ operates. Note that we can introduce a similar approach to do this; the only difference lies in the combination of the contacting parts of both polygons i.e. $\mathcal{Q}_{up}$ can only contact $\mathcal{P}_{up}$ and $\mathcal{Q}_{down}$ can only contact $\mathcal{P}_{down}$.

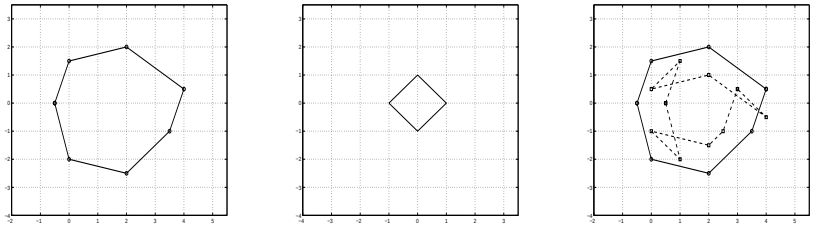
If we take the polygons from fig.6.4(a) and 6.4(b) we get a straightforward contact-boundary (see fig.6.8(c)) in much the same way as in fig.6.4(c).



(a) $\mathcal{P}_\mathcal{A}$ (b) $\mathcal{P}_\mathcal{B}$ (c) $\mathcal{P}_\mathcal{A} \breve{\oplus} \mathcal{P}_\mathcal{B}$

Figure 6.8: $\mathcal{P}_\mathcal{B}$ moving inside $\mathcal{P}_\mathcal{A}$.

We can do the same for $\mathcal{P}_\mathcal{C}$ and $\mathcal{P}_\mathcal{B}$, resulting in fig.6.9(c).



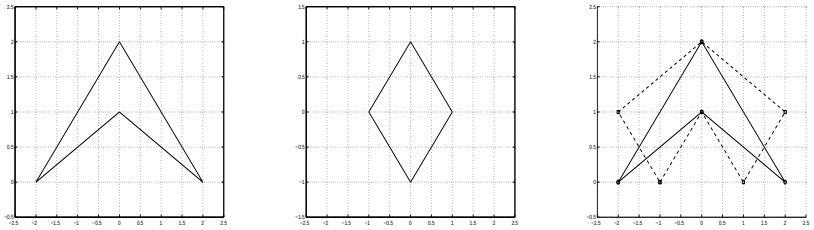
(a) $\mathcal{P}_\mathcal{C}$ (b) $\mathcal{P}_\mathcal{B}$ (c) $\mathcal{P}_\mathcal{C} \breve{\oplus} \mathcal{P}_\mathcal{B}$

Figure 6.9: $\mathcal{P}_\mathcal{B}$ moving inside $\mathcal{P}_\mathcal{C}$.

$\mathcal{P}_\mathcal{C} \breve{\oplus} \mathcal{P}_\mathcal{B}$ seems to be quite complex but can for the largest part be explained in a similar fashion to $\mathcal{P}_\mathcal{E} \breve{\oplus} \mathcal{P}_\mathcal{B}$ in fig.6.7(c); $\mathcal{P}_\mathcal{C} \breve{\oplus} \mathcal{P}_\mathcal{B}$ has to fold back when sliding $\mathcal{P}_\mathcal{B}$ from the last edge of the upper part to the last edge of the lower part i.e. the lower part of $\mathcal{P}_\mathcal{B}$ intersects the lower part of $\mathcal{P}_\mathcal{C}$ when the upper part of $\mathcal{P}_\mathcal{B}$ kisses the upper part of $\mathcal{P}_\mathcal{C}$.

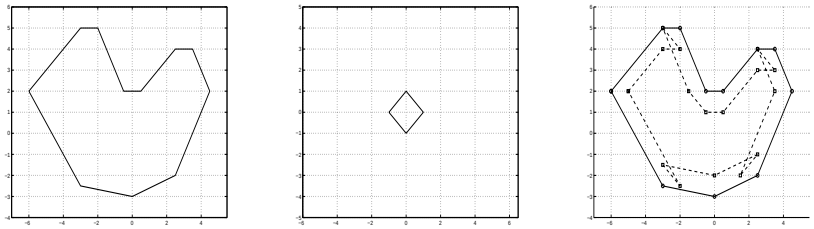
Even though we can easily see that $\mathcal{P}_\mathcal{B}$ will in no way fit within $\mathcal{P}_\mathcal{D}$ in such a way that the resulting contact-boundary can be interpreted as $\mathcal{P}_\mathcal{B}$ moving around 'in' $\mathcal{P}_\mathcal{D}$, we can compute $\mathcal{P}_\mathcal{D} \breve{\oplus} \mathcal{P}_\mathcal{B}$, as can be seen in fig.6.10(c).

Finally, we can slide $\mathcal{P}_\mathcal{B}$ within $\mathcal{P}_\mathcal{E}$, resulting in the contact-boundary shown in fig.6.11(c).



(a) $\mathcal{P}_{\mathcal{D}}$ (b) $\mathcal{P}_{\mathcal{B}}$ (c) $\mathcal{P}_{\mathcal{D}} \breve{\oplus} \mathcal{P}_{\mathcal{B}}$

Figure 6.10: $\mathcal{P}_{\mathcal{B}}$ moving inside $\mathcal{P}_{\mathcal{D}}$ with penetration.



(a) $\mathcal{P}_{\mathcal{E}}$ (b) $\mathcal{P}_{\mathcal{B}}$ (c) $\mathcal{P}_{\mathcal{E}} \breve{\oplus} \mathcal{P}_{\mathcal{B}}$

Figure 6.11: $\mathcal{P}_{\mathcal{B}}$ moving inside $\mathcal{P}_{\mathcal{E}}$ with penetration.

When looking at the complexity for computing the contact-boundary for polygons like the ones we have used in this section, we can extend the proofs provided in sections 4.3 and 5.3 for computing the contact of the separate parts of the polygons used, yielding an $O(N + M)$ computational complexity for two N-point polygons with a possible additive factor for the segmentation of a non-convex part of one of the polygons.

Chapter 7

Conclusion

The spectral decomposition of the Legendre transform for boundaries of discrete objects i.e. $2D$ polygons representing some 'real world' object, has provided us with a simplification for computing the contact-boundary of two potential colliding bodies; a complicated dilation operation based on tangent planes becomes a straightforward addition in the Legendre domain. This provides us with a tractable and potentially powerful tool for mathematical morphology and with possible implications in the areas of robot-object-interaction (collision-avoidance) and computer graphics.

We have shown that a discrete implementation is indeed possible based upon the theory on the Legendre transform and the tangential dilation as presented by Dorst and Van den Boomgaard. Moreover, the algebraic parallels between the Legendre and Fourier transform, as found in [6], and the decomposition proposed by Corrias [2], have indeed proven their potential to convert a discrete Legendre transform into a faster transform, analogous to that of the Fast Fourier transform, hence compressing the computational complexity of the transform itself and, in turn, of the tangential dilation operation to $O(|\Omega|logN)$.
We have however found a number of problems when applying this approach to (parts of) polygonal objects concerning the accuracy of describing objects using equidistant sampling and (possible) redundant computation, implying that this approach, although usable, is not most suited for handling this kind of data[1].

In constructing the algorithm we found that, in order to use finite discrete data, we had to find a way to mimic the behaviour of continuous data such that we would stay very close to the mathematical theory and still obtain valid contact-boundaries given the data. This meant finding a way to counter the loss of data when using a discretized version of the Legendre transform combined with a valid theoretical foundation. We proposed to introduce a constant dependent on the dimension and slopes of the objects used, resolving both data-loss and theoretical difficulties[2].
Combining this with the alternative approach for *local* maximum-computation, proposed by Lucet in [8], provided us with a solid foundation for developing a *linear* complexity algorithm suited for polygonal data. We extended the pos-

[1] See section 3.2.1.
[2] See section 4.2.1.

sibilities of the algorithm by introducing a method to handle polygons with non-convex parts – we found that we could segment non-convex parts of polygons into its concave and convex components using *mutual edges* and re-attach them after applying the transform on the components separately (see chapter 5). The result is a $O(N)$ worst-case complexity algorithm able to handle numerous polygon-collisions which could be of value in the areas of robotics and computer-graphics.

Future work

The algorithm in its present form, even though being able to proces numerous types of polygons and polygon-collisions, still lacks the possibility of handling 'fold backs' in polygons and the contact of two non-convex polygons.

Fold backs posed a problem since there was no way to use the segmentation that was introduced in this thesis. A possible alternative might be constructed using an adapted version of a polygon partitioning algorithm [1].

Handling the contact of a non-convex part $\mathcal{P}$ and a concave/convex part $\mathcal{Q}$ requires us to keep track of the slopes bounded by the slopes of the individual segments of the non-convex part which in turn are based upon the position of the segment within $\mathcal{P}$ i.e. keeping track of valid slopes for each contact of a segment of $\mathcal{P}$ with $\mathcal{Q}$[3]. The possibility of an extension for two non-convex parts therefore seems to be applicable but does need some additional research to obtain valid transforms in the Legendre domain and, in turn, to yield valid contact-boundaries.

[3] See section 5.2.

Bibliography

[1] M. de Berg, M. van Krefeld, M. Overmars, O. Schwarzkopf: *Computational Geometry – Algorithms and Applications* [2nd ed.], Springer-Verlag Berlin Heidelberg *(2000)*

[2] L. Corrias: *Fast Legendre-Fenchel Transform and Applications to Hamilton-Jacobi Equations and Conservation Laws*, Siam Journal of Numerical Analysis, vol.33/nr.4, pp.1534-1558 *(1996)*

[3] L. Dorst & R. van den Boomgaard: *An Analytical Theory of Mathematical Morphology*, Mathematical Morphology and its Applications to Signal Processing, pp.245-250 *(1993)*

[4] L. Dorst & R. van den Boomgaard: *Morphological Signal Processing and the Slope Transform*, invited paper for Signal Processing, vol.38, pp.79-98 *(1993)*

[5] L. Dorst & R. van den Boomgaard: *Orientation-Based Representations for Mathematical Morphology*, Shape, Structure and Pattern Recognition, World Scientific, pp.13-22. *(1995)*

[6] L. Dorst & R. van den Boomgaard: *The Systems Theory of Contact*, Invited presentation for workshop Algebraic Frames for the Perception-Action Cycle, Kiel *(2000)*

[7] P. Keuning: *2D Shape from touching*, AI Masters' Thesis, University of Amsterdam *(1995)*

[8] Y. Lucet: *Faster than the Fast Legendre Transform, the Linear-time Legendre Transform*, Numerical Algorithms 16, pp.171-185 *(1997)*

[9] Numerical Recipes: Section 12.2; Fast Fourier Transform – http://lib-www.lanl.gov/numerical/bookcpdf.html