Evaluering af en skæringsalgoritme for Bezier kurver i planen

Speciale af Thomas Rasmussen

August 2007

Abstract

This thesis contains an evaluation of an algorithm by Chee K. Yap presented in the article *Complete subdivision algorithms, I: intersection of Bezier curves*. The algorithm intersects general Bézier curves, and is based on subdivision, geometric separation bounds, and a complete criterion for detecting non-crossing intersection. The aim of the evaluation is to explore the effectiveness of the algorithm in practice. This is done by confirming that the algoritme actually achieves completeness and exact solutions. Additionally, the algorithm is implemented and tested empirically to establish if efficiency is achieved. The evaluation shows that the geometric separation bounds from which the stop criterion for the subdivision procedures is calculated, leads to a huge amount of iterations resulting in loss of effectiveness in practice.

Resumé

Dette speciale indeholder en evaluering af Chee K. Yaps algoritme beskrevet i artiklen *Complete subdivision algorithms, I: intersection of Bezier curves*. Algoritmen finder skæringspunkterne mellem generelle Bézier kurver og er baseret på opdeling (subdivision) af disse, geometriske separationsgrænser og et kriterium til af- og bekræftelse af et tangentielt, ikke-krydsende skæringspunkt. Evalueringen består i at undersøge, hvorvidt Yaps algoritme opnår effektivitet i praksis. Dette gøres ved først at bekræfte, at algoritmen reelt finder alle skæringspunkter og beregner disse eksakt. Derudover udarbejdes en konkret implementering med henblik på empiriske målinger af udførselstiden for at afdække effektiviteten i praksis. Evalueringen viser, at stop-kriteriet til subdivision procedurerne, som er baseret på de geometriske separationsgrænser, medfører et ualmindeligt stort antal iterationer der ødelægger effektiviteten i praksis.

Indhold

1 Indledning **5**

2 Bézier kurver - en introduktion **8**
2.1 Matematisk baggrund 8
2.2 Paul deCasteljaus algoritme 10
2.3 Bernstein repræsentationen 12
2.4 Den afledte af en Bézier kurve 14
2.5 Opdeling af en Bézier kurve 15
2.6 Notation 16

3 Algebraiske kurver og geometriske separationsgrænser **18**
3.1 Polynomier, algebraiske tal og kurver 18
3.2 Geometriske seperationsgrænser 19
3.3 Vurdering af $\|\cdot\|_2$-normen for Bézier kurver 23

4 Elementære kurver og par **26**
4.1 Elementær kurver 26
4.2 Elementære par 27
4.3 Det komplette kriterium for ikke-krydsende, tangentiel skæring . . . 28

5 Skæringsalgoritmen **30**
5.1 Den overordnede algoritme 31
5.2 Initialiseringsfasen 32
5.3 Makrofasen 33
5.4 Mikrofasen 34
5.4.1 Kritiske punkter og ikke-elementære kurver 34
5.4.2 Kobling processen 40
5.4.3 Skæring mellem Bézier kurver og ret linie 44
5.4.4 Fortegnet på α-vinklen 48

6 Implementering og evaluering **53**
6.1 Komplethed og eksakte løsninger 53
6.2 Effektivitet i praksis 55

7 Konklusion **66**

A Kode - Geometriske primitiver **68**

B Kode - Makrofasen **70**

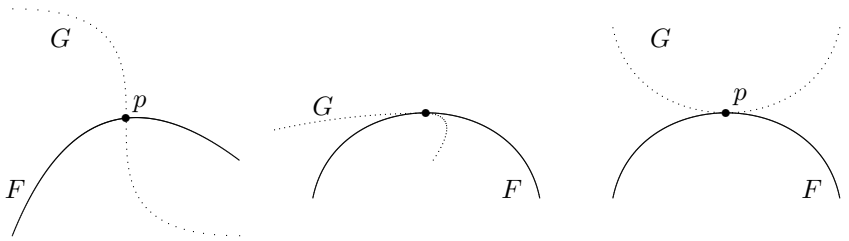
C Kode - Geometriske funktioner **71**

Kapitel 1

Indledning

Skæringsproblemet for kurver dækker over problemet med at beregne skæringspunkter mellem to angivne kurver. En algoritme, der løser dette problem, kaldes herefter en *skæringsalgoritme*. Skæringsproblemet for algebraiske kurver er et fundamentalt problem indenfor områder som *Computer Aided Design and Manufacturing*[1] og indenfor analyse af kurvearrangementer [2]. De fleste skæringsalgoritmer er baseret på såkaldte *fri-form* kurver [3]. I dette speciale betragtes en speciel klasse af fri-form kurver, nemlig Bézier kurver. Mange skæringsalgoritmer for Bézier kurver er ikke eksakte, og lider ofte af robusthedsproblemer. Grunden til disse robusthedsproblemer skitseres nedenfor.
En Bézier kurve F udgør et endeligt kurvesegment, og er repræsenteret ved en endelig sekvens, $P(F) = (p_0, p_1, \ldots, p_m)$, af kontrol punkter [3]. Hvis $CH(F)$ er det konvekse hylster af $P(F)$, betragtet som en lukket mængde, så kaldes et par, (F, G), af Bézier kurver et kandidatpar, hvis der gælder $CH(F) \cap CH(G)$ ikke er tom. Generelt er skæringsalgoritmer for Bézier kurver baseret på to ideer. For det første, en Bézier kurve, F, er indeholdt i $CH(F)$, hvilket kan bruges til at fjerne ikke-kandidatpar fra betragtning i algoritmen. For det andet, det primære algoritmiske skridt består i at opdele en Bézier kurve, F i to kurver F_0 og F_1, ved brug af deCasteljaus algoritme. En generel skæringsalgoritme baseret på opdeling, opretholder en kø, Q, af kandidatpar. Så længe køen ikke er tom, udtages et par (F, G). Hvis diameteren[1] af $CH(F) \cup CH(G)$ er mindre en ϵ, så betragtes (F, G) som et skæringspunkt og rapporteres. Ellers opdeles den kurve med størst diameter, eksempelvis F, i kurverne F_0 og F_1, og parrene (F_0, G) og (F_1, G) indsættes i Q.
Den generelle skæringsalgoritme, beskrevet ovenfor, afhænger af konstanten $\epsilon > 0$. Kurvepar (F, G) med diameter mindre end ϵ betragtes, som om de skærer hinanden, uanset om det er tilfældet eller ej. Det kan også være tilfældet, at et rapporteret kurvepar (F, G) reelt indeholder flere end et skæringspunkt. En sådan adfærd kan i nogle tilfælde være acceptabel, men for eksempel i topologisk analyse af kurvearrangementer er det nødvendigt, at hvert rapporteret par (F, G) kun indeholder et entydigt skæringspunkt.
Lad p være et skæringspunkt mellem Bézier kurverne F og G. Under antagelse af, at F og G ikke ligger helt eller delvist ovenpå hinanden, vil p være et isoleret punkt i $F \cup G$. Skæringspunktet, p, kan enten være tangentielt eller transversalt afhængig af, om tangenterne til henholdsvis F og G er sammenfaldende. Alternativt kan p betegnes som *krydsende* eller *ikke-krydsende* afhængig af, om kurverne krydser hinanden i p. Et transversalt skæringspunkter er altid krydsende, mens et tangentielt skæringspunkt kan være enten krydsende eller ikke-krydsende. Disse tre tilfælde er vist i figur 1.1.

[1] Diameteren af en lukket og begrænset mængde $X \in \mathbb{R}^2$, udgør supremum over alle afstande mellem to forskellige punkter i X.



(a) Transversalt (krydsende)

(b) Tangentielt (krydsende)

(c) Tangentielt (ikke-krydsende)

Figur 1.1: Navngivning af skæringspunkter

Med den ovenstående karakterisering af skæringspunkterne er det nu ikke svært at se, at den generelle skæringsalgoritme ikke kan bekræfte eventuelle tangentielle, ikke-krydsende, skæringspunkter. Grunden til dette er, at hvis afstanden mellem kurverne, F og G, er mindre end ϵ, så vil den generelle skæringsalgoritme altid rapporterer (F, G) som et skæringspunkt, også selvom kurverne ikke skærer hinanden.
Hvis problemerne med den generelle skæringsalgoritme skal løses, er to ting nødvendige. Først og fremmest, er det nødvendigt at kunne opdele kurverne således, at de resulterende kandidatpar højst indeholder et skæringspunkt. Dernæst skal det være muligt at kunne afgøre, om et kandidatpar, som højst indeholder et skæringspunkt, så indeholder skæringspunktet.
Hvis disse to ting kan opfyldes, er det teoretisk muligt at designe en komplet skæringsalgoritme, som under antagelse af eksakt aritmetik også kunne opnå eksakthed i løsningerne.
I dette speciale betragtes en skæringsalgoritme, der forsøger at opnå de to ovenstående ting. Denne skæringsalgoritme er beskrevet i artiklen *Complete subdivision algorithms, I: intersection of Bezier curves* [4], og er baseret på den generelle skæringsalgoritme, beskrevet ovenfor. Denne nye skæringsalgoritme vil fremover blive kaldt Yaps (skærings-)algoritme. I artiklen [4] beskriver Yap, hvordan det i teorien er muligt at opnå komplethed og eksakte løsninger. Han baserer sin algoritme på eksakt aritmetik for at opnå eksakte løsninger. For at opnå komplethed udvikler han blandt andet en grænse, kun afhængig af input kurvernes algebraiske egenskaber, for, hvor lille ϵ skal være for at opnå, at hvert kandidatpar højst indeholder et skæringspunkt. Derudover giver han en beskrivelse af en proces, som kan afgøre, om et kandidatpar reelt indeholder et skæringspunkt eller ej. Denne proces involverer et kriterium, som både kan af- og bekræfte et tangentielt, ikke-krydsende, skæringspunkt.
Spørgsmålet er, om udvidelsen af den generelle skæringsalgoritme til Yaps algoritme, påvirker effektiviteten i praksis. Introduktionen af eksakt aritmetik i en algoritme påvirker ofte effektiviteten i praksis, og idet at størrelsen af ϵ har stor påvirkning på effektiviteten af den generelle skæringsalgoritme, baseret på opdeling, er det ikke oplagt, at Yaps algoritme kan opnå effektivitet i praksis.

Formål med dette speciale er at evaluere Yaps skæringsalgoritme med hensyn til effektivitet i praksis. Denne evaluering består af to dele.

- En gennemgang af teorien bag Yaps skæringsalgoritme.
- En empirisk undersøgelse af en konkret implementering af Yaps algoritme fra beskrivelsen i [4, 5].

En gennemgang af teorien bag Yaps skæringsalgoritme foretages med henblik på at bekræfte, om algoritmen opnår sine mål med hensyn til komplethed og eksakte løsninger. Samtidig giver gennemgangen mulighed for at udfylde de steder i

beskrivelsen af algoritmen, hvor forfatteren har været vag i sin forklaring. En sådan gennemgang hjælper også med til at give en indsigt i, hvor en empirisk undersøgelse skal fokusere for at afdække effektiviteten af en konkret implementering i praksis.

Opbygning

- Kapitel 2 introducerer Bézier kurver, som udgør de kurver, Yaps algoritme finder skæringspunkter for. Samtidig introduceres den matematiske teori for Bézier kurver, som er nødvendig for at forstå Yaps algoritme. Denne teori indeholder blandt andet deCasteljaus algoritme, samt den afledte af en Bézier kurve.
- Kapitel 3 giver det primære værktøj i teorien bag Yaps skæringsalgoritme. Dette værktøj udgør de geometriske separationsgrænser, som udgør stop-kriterier for de numeriske procedurer i Yaps algoritme.
- Kapitel 4 introducerer elementære Bézier kurver og par. Samtidig gives et kriterium for af- og bekræftelse af tangentiel, ikke-krydsende skæring mellem to elementære Bézier kurver. Dette kriterium ligger til grund for Yaps kobling proces.
- Kapitel 5 giver den fulde beskrivelse af teorien bag Yaps skæringsalgoritme og desuden en forklaring af algoritmen selv.
- Kapitel 6 er min evaluering af Yaps algoritme, både den teoretiske gennemgang og den empiriske undersøgelse af en konkret implementering.

Kapitel 2

Bézier kurver - en introduktion

En Bézier kurve er en parametrisk kurve. Bézier kurver blev udbredt i 1962 af den franske ingeniør Pierre Bézier, som benyttede dem til at designe former i den franske bilindustri. Kurverne blev først udviklet i 1959 af Paul de Casteljau, som udviklede en numerisk stabil metode til at evaluere Bézier kurver. Denne algoritme går under navnet *deCasteljaus algoritme*.
Afsnit 2.1 - 2.4 udgør en kort introduktion til Bézier kurver og deres matematiske egenskaber, hvor der primært er lagt vægt på de egenskaber, der er anvendt i forbindelse med Yaps algoritme [4]. Afsnit 2.5 præsenterer en vigtig algoritme, baseret på deCasteljaus algoritme, til opdeling af en Bézier kurve i to delkurver. Denne algoritme er det primære algoritmiske skridt i Yaps algoritme. Det sidste afsnit, 2.6, gør rede for den notation som anvendes i forbindelse med Bézier kurver i resten af specialet. Introduktionen til Bézier kurver i dette kapitel er baseret på Farins bog [3, kap. 3].

2.1 Matematisk baggrund

Det følgende er et kort repetition af de matematiske begreber som er nødvendige i behandlingen af Bézier kurver.

Affine Rum

Det sted, hvor Bézier kurver *lever*, er i de såkaldte affine rum. Følgende definition af affine rum stammer fra den lineære algebra.

Definition 2.1.1. *Et affint underrum $A \subseteq \mathbb{R}^n$ er et underrum, der er lukket under dannelse af linear kombinationer af typen*

$$\sum_{i=0}^{m} \alpha_i b_i, \qquad \alpha_i \in \mathbb{R}, b_i \in A, \qquad \sum_{i=0}^{m} \alpha_i = 1$$

De lineære kombinationer ovenfor kaldes ofte affine kombinationer, og hvis der gælder, at $\alpha_i \geq$ for alle $i = 0, 1, \ldots, m$ så kaldes de konvekse affine kombinationer.

Denne definition er måske en smule abstrakt, men i det følgende gøres rede for, hvilke konsekvenser den har for de Bézier kurver, som dette speciale beskæftiger sig med.

Sammenhængen mellem affine rum og generelle underrum af $\mathbb{R}^n$ er som følgende. Lad $U \subseteq \mathbb{R}^n$ været et underrum og lad $d \in \mathbb{R}^n$, være en vektor som ikke nødvendigvis behøver at ligge i U. Mængden $A = d + U$ bestående af alle elementer, som kan skrives som $d + u, u \in U$ er et affint rum. Dette kan bekræftes ved at betragte den affine kombination $a(d + u) + b(d + v)$, hvor $a, b \in \mathbb{R}, a + b = 1$ og $u, v \in U$.

$$a(d + u) + b(d + v) = (a + b)d + (u + v) = d + (u + v)$$

I det følgende betragtes udelukkende det affine underrum $\mathbb{E}^2 \subseteq \mathbb{R}^2$, som i termer af det ovenstående er defineret som følgende. Lad $d \in R^2$ være en givet vektor, så er $\mathbb{E}^2 = d + u, u \in \mathbb{R}^2$. Der gøres nu kort rede for konsekvenserne af definitionen af affine rum i forhold til hvorledes dette rum udnyttes i forbindelse med Bézier kurver.
Elementerne i det $\mathbb{E}^2$ betegnes som *punkter*. Et punkt betegner en position ofte relativ til andre objekter. Eksempler kan være kontrol punkter til Bézier kurver eller skæringspunkter mellem kurver og linier. På samme måde betegnes elementerne i $\mathbb{R}^2$ som vektorer. Selvom punkter og vektorer begge er beskrevet ved par af reelle tal, bør det understreges, at der er en klar forskel mellem dem. Der gælder nemlig, at for alle par af punkter a og b findes en entydig vektor således, at

$$v = b - a, \qquad a, b \in \mathbb{E}^2, \qquad v \in \mathbb{R}^2$$

Mens de på den anden side gælder, at for en given vektor v findes uendelig mange par af punkter a, b således, at $v = b - a$. Hvis a, b er sådan et par og hvis w er en tilfældig vektor, så gælder også $v = (b + w) - (a + w)$.
Ydermere gælder at hvis $w \in \mathbb{R}^2$ er en givet vektor, så udgør tildelingen af punktet $a + w$ alle punkter $a \in \mathbb{E}^2$ en translation. Som det kan ses fra det ovenstående, så er vektorer invariante overfor translationer, mens punkter ikke er det.
Elementer i $\mathbb{E}^2$ kan kun subtraheres, hvilket giver en vektor, men kan ikke adderes. Det faktum, at punkter fra $\mathbb{E}^2$ ikke kan adderes, kan dog virke lidt underligt i forhold til, at et affint rum er defineret ved hjælp af vægtede summe af punkter - de såkaldte affine kombinationer. Men det viser sig ved følgende omskrivning

$$b = b_0 + \sum_{j=1}^{n} \alpha_j (b_j - b_0)$$

at der i virkeligheden er tale om en sum mellem et punkt og en vektor. En vigtig egenskab ved konvekse affine kombinationer, $b = \sum_{j=0}^{n} \alpha_j b_j$, er, at b altid er indeholdt i det konvekse hylster udspændt af b_j for $j = 0, 1, \ldots, n$. Denne egenskab bruges blandt andet til at redegøre for, hvorfor Bézier kurver altid er indeholdt i det konvekse hylster udspændt af kontrol punkterne.

Affine Afbildninger

Ofte bruges transformationer til at positionere eller orientere objekter (kurver) relativ til et givet koordinatsystem. Disse transformationer udgør affine afbildninger. I det følgende defineres affine afbildninger ved hjælp af affine kombinationer.

Definition 2.1.2. *Afbildningen $\Phi : \mathbb{E}^2 \to \mathbb{E}^2$ kaldes en affin afbildning, hvis den opretholder affine kombinationer. Det vil sige, hvis*

$$x = \sum_{j=0}^{n} \alpha_j a_j, \qquad \sum_{j=0}^{n} \alpha_j = 1, \qquad x, a_j \in \mathbb{E}^2$$

og Φ er en affin afbildning, så gælder

$$\Phi(x) = \sum_{j=0}^{n} \alpha_j \Phi(a_j), \qquad \Phi(x), \Phi(a_j) \in \mathbb{E}^2$$

En affin afbildning har også en mere velkendt form, nemlig $\Phi(x) = Ax + v$, hvor $A \in Mat_{2\times 2}(\mathbb{R})$ og $v \in \mathbb{R}^2$. Den følgende omskrivning viser, hvorfor

$$\begin{aligned}\Phi\left(\sum_{j=0}^{n}\right) &= A\left(\sum_{j=0}^{n}\alpha_j a_j\right) + v \\ &= \sum_{j=0}^{n}\alpha_j A a_j + \sum_{j=0}^{n}\alpha_j v \\ &= \sum_{j=0}^{n}\alpha_j (A a_j + v) \\ &= \sum_{j=0}^{n}\alpha \Phi(a_j)\end{aligned}$$

Eksempler på en affin afbildning kunne være en translation, en rotation eller en skalering. Nedenfor gøres rede for, hvad disse affine afbildninger har med Bézier kurver at gøre.

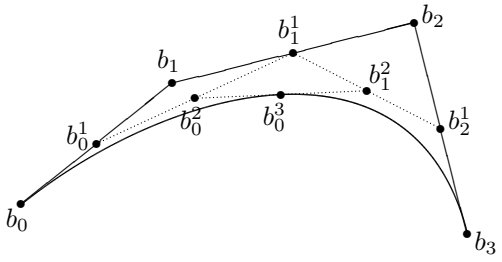
2.2 Paul deCasteljaus algoritme

Definition 2.2.1. *Givet $p_0, p_1, \ldots, p_n \in \mathbb{E}^2, n \geq 2$ og $t \in \mathbb{R}$. Sæt*

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + t b_{i+1}^{r-1}(t) \qquad \begin{cases} r = 1, 2, \ldots, n \\ i = 0, 1, \ldots, n-r \end{cases}$$

hvor $b_i^0(t) = p_i$. Så er $b_0^n(t)$ det punkt med parameter værdien t på Bézier kurven b^n. Punkterne p_i for $i = 0, 1, \ldots, n$ kaldes for kontrol punkter, og polygonen udgjort af $(p_0, p_1, \ldots, p_n)$ kaldes kontrol polygonen.

På figur 2.1 vises deCasteljaus algoritme for den kubiske Bézier kurve, $b^3(t) = b_0^3(t)$, for parameter værdi $t = 1/2$. På figuren vises også alle de mellemliggende punkter $b_j^r(t)$. Bemærk, at (t) er udeladt for overskuelighedens skyld.



Figur 2.1: deCasteljau algoritmen: For det kubiske tilfælde $n = 3$, $t = 1/2$ opnås $b_0^3(t)$ ved gentaget lineær interpolation.

For at give lidt bedre intuition om, hvordan deCasteljaus algoritme fungerer, vises her *deCasteljaus skema* for den kubiske Bézier kurve.

$$\begin{array}{cccc} b_0^0(t) & & & \\ b_1^0(t) & b_0^1(t) & & \\ b_2^0(t) & b_1^1(t) & b_0^2(t) & \\ b_3^0(t) & b_2^1(t) & b_1^2(t) & b_0^3(t) \end{array}$$

Hvor hver indgang i skemaet er beregnet, jævnfør definition 2.2.1, ved lineær interpolation mellem henholdsvis indgangen over og til venstre og indgangen umiddelbart til venstre. Eksempelvis beregnes $b_1^2(t) = (1-t)b_1^1(t) + tb_2^1(t)$. Det vil sige, at for at beregne et punkt på den kubiske Bézier kurve skal alle indgangene i deCasteljaus skema beregnes, hvilket også gælder i det generelle tilfælde. Bemærk dog at den første kolonne i skemaet er givet ved kontrol punkterne $b_j = p_j$ for $j = 0, 1, 2, \ldots, n$. En rekursiv algoritme til beregning af punkterne på en generel Bézier kurve med kontrol punkter $p_0, p_1, \ldots, p_n$ se ud som algoritme 2.2.1.

Algoritme 2.2.1 EvalBezier(b^n, t, j, r)

Input: En Bézier kurve b^n, en parameter værdi t, $j \in \{0, 1, \ldots, n-r\}$, $r \in \{1, 2, \ldots, n\}$
Output: Punktet $b^n(t)$ på Bézier kurven for parameter værdien t.
  if $r = 0$ **then**
    return b_j^0 $\{b_j^0 = p_j\}$
  end if
  return $(1-t)\cdot$EvalBezier$(b^n, t, j, r-1) + t\cdot$EvalBezier$(b^n, t, j+1, r-1)$

Givet Bézier kurven b^n og parameter værdien t så startes algoritmen med EvalBezier(b^n, t, j, r), hvor $j = 0$ og $r = n$. Bemærk, at rekursionsdybden for algoritme 2.2.1 er af samme størrelsesorden som antallet af kontrol punkter i Bézier kurverne. En iterativ algoritme er også mulig, men beskrives ikke her.

Nogle af de vigtigste egenskaber for Bézier kurver, som gøres brug af i dette speciale, er invariante overfor affine afbildninger, invarians overfor affine transformationer af parameter intervallet og konveks hylster egenskaben. Disse egenskaber opsummeres i bemærkning 2.2.2.

Bemærkning 2.2.2. *Lad $b^n, n \geq 2$ være en generel Bézier kurve med kontrol punkter $p_0, p_1, \ldots, p_n$ og antag at $\Phi : \mathbb{E}^2 \to \mathbb{E}^2$ er en affin afbildning.*

- ***Invarians overfor affine afbildninger*** *Affine afbildninger er et vigtigt redskab i de fleste CAD systemer, hvor de anvendes til positionering, orientering og skalering af objekter. At Bézier kurver er invariante overfor affine afbildninger betyder, at følgende to procedurer er ækvivalente: 1) Beregn punkter, $b^n(t)$, på kurven og derefter beregn den affine afbildning af hver beregnet punkt, altså $\Phi(b^n(t))$. 2) Beregn først den affine afbildning af alle kontrol punkterne b_j for $j \in \{0, 1, \ldots, n\}$. Beregn derefter punkterne på kurven med deCasteljaus algoritme med de transformerede kontrol punkter $\Phi(b_j)$.*
  Affine invarians er en direkte konsekvens af deCasteljaus algoritme, da algoritmen består af endelig sekvens af lineær interpolationer. Lineær interpolation udgør en affin afbildning, som selv er affin invariant, og en endelig sekvens af affin invariante afbildninger udgør også en affin afbildning.

- ***Invarians over for affine parameter transformationer*** *En Bézier kurve er ofte defineret på intervallet $[0, 1]$. Det er ikke en nødvendighed, men blot bekvemt at gøre det. Det gælder nemlig angående deCasteljaus algoritme, at den er upåvirket af det interval, som kurven er defineret på. Dette skyldes, at hver punkt beregnet i algoritmen udgør en ratio mellem to tidligere beregnede punkter. Man kan derfor betragte en kurve, som om den er defineret på et tilfældigt interval $a \leq u \leq b$ ved at indføre parameter transformationen $t = (u-a)/(b-a)$. Dette ligger op til følgende generaliserede deCasteljau algoritme*

$$b_i^r(u) = \frac{b-u}{b-a} b_i^{r-1}(u) + \frac{u-a}{b-a} b_{i+1}^{r-1}(u).$$

Afbildningen $t(u) = (u-a)/(b-a)$ udgør en affin afbildning, så generelt gælder, at Bézier kurver er invariante overfor affine parameter transformationer.

- ***Konveks hylster egenskaben*** *For $t \in [0,1]$ så ligger $b^n(t)$ indenfor i det konvekse hylster af sin kontrol polygon. Dette følger fra det faktum, at de mellemliggende punkter b_i^r i deCasteljaus algoritme opstår ved konvekse affine kombinationer af de foregående punkter b_j^{r-1}, hvilket gør, at ligger indenfor det konvekse hylster af kontrol punkterne $p_0, p_1, \ldots, p_n$. Vigtigheden ved denne egenskab er, at den kan bruges til at bekræfte, to Bézier kurver ikke skære hinanden. Hvis de konvekse hylstre af kontrol polygonerne ikke skærer hinanden, så vil Bézier kurverne heller ikke skære hinanden.*

- ***Endepunkt interpolation*** *En Bézier kurve går igennem både b_0 og b_n fordi $b^n(0) = b_0$ og $b^n(1) = b_m$, hvilket kan bekræftes ved at bruge definition 2.2.1.*

2.3 Bernstein repræsentationen

Bézier kurver blev i afsnit 2.2 defineret ved deCasteljaus (rekursive) algoritme. Denne definition udgør en implicit repræsentation af Bézier kurver, men det er ofte bekvemt også at have en eksplicit repræsentation af kurverne. En sådan repræsentation kan gives ved hjælp af Bernstein polynomierne.

Definition 2.3.1. *Det generelle Bernstein polynomium er givet ved*

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \qquad \binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{hvis } 0 \leq i \leq b \\ 0 & \text{ellers} \end{cases}$$

og har grad n.

Før sammenhængen mellem Bernstein polynomierne og Bézier kurver gennemgås, introduceres to nødvendige egenskaberne for Bernstein polynomierne. Disse vigtige egenskaber udgøres af følgende sætninger.

Sætning 2.3.2. *Bernstein polynomierne opfylder*

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$$

hvor $B_0^0(t) = 1$ og $B_j^n(t) = 0$ for $j \notin \{0, 1, \ldots, n\}$.

Bevis.

$$\begin{aligned} B_i^n(t) &= \binom{n}{i} t^i (1-t)^{n-i} \\ &= \binom{n-1}{i} t^i (1-t)^{n-i} + \binom{n-1}{i-1} t^i (1-t)^{n-i} \\ &= (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \end{aligned}$$

□

Sætning 2.3.3. *Bernstein polynomierne opfylder*

$$\sum_{j=0}^{n} B_j^n(t) = 1$$

Bevis.

$$1 = [t + (1-t)]^n = \sum_{j=0}^{n} \binom{n}{j} t^j (1-t)^{n-j} = \sum_{j=0}^{n} B_j^n(t)$$

□

Med disse to egenskaber for Bernstein polynomierne er det nu muligt at gøre rede for sammenhængen med Bézier. De mellemliggende punkter, b_i^r, i deCasteljaus algoritme kan udtrykkes ved hjælp af Bernstein polynomierne.

Sætning 2.3.4. *deCasteljau punkterne b_i^r fra deCasteljaus algoritme kan udtrykkes*

$$b_i^r(t) = \sum_{j=0}^{r} b_{i+j} B_j^r(t), \qquad \begin{cases} r \in \{0, 1, \ldots, n\} \\ i \in \{0, 1, \ldots, n-r\} \end{cases}$$

Bevis.

$$\begin{aligned} b_i^r(t) &= (1-t) b_i^{r-1}(t) + t b_{i+1}^{r-1}(t) \\ &= (1-t) \sum_{j=i}^{i+r-1} b_j B_{j-1}^{r-1}(t) + t \sum_{j=i+1}^{i+r} b_j B_{j-i-1}^{r-1}(t) \\ &= (1-t) \sum_{j=i}^{i+r} b_j \left[B_{j-1}^{r-1}(t) + t B_{j-i-1}^{r-1}(t) \right] \\ &= \sum_{j=0}^{r} b_{i+j} B_j^r(t) \end{aligned}$$

□

Det vigtigste ved sætning 2.3.4 er, at man nu kan udtrykke en Bézier kurve b^n ved Bernstein polynomierne ved at sætte $r = n$. Så er

$$b^n(t) = b_0^n(t) = \sum_{j=0}^{n} b_j B_j^n(t)$$

en eksplicit formel for en Bézier kurve. Denne formel kaldes Bernstein repræsentationen for en Bézier kurve. Før dette afsnit afsluttes, nævnes endnu en vigtig egenskab for Bézier kurver, som er en direkte konsekvens af Bernstein repræsentationen.

Sætning 2.3.5. *Lad punkterne $b_0, b_1, \ldots, b_n \in \mathbb{E}^2$ være givet og lad to Bézier kurver med kontrol polygon givet ved henholdsvis $b_0, b_1, \ldots, b_n$ og $b_n, b_{n-1}, \ldots, b_0$. Så repræsenterer disse to ordninger af kontrol punkterne den samme Bézier kurve. Det vil sige kurverne er symmetriske.*

Bevis. Der gælder følgende ligning

$$\sum_{j=0}^{n} b_j B_j^n(t) = \sum_{j=0}^{n} b_{n-j} B_j^n(1-t)$$

som følger fra det faktum, at Bernstein polynomierne er symmetriske med hensyn til t og $1-t$. Der gælder nemlig

$$B_j^n(t) = B_{n-j}^n(1-t)$$

□

2.4 Den afledte af en Bézier kurve

Yaps artikel [4] gør brug af den afledte af en Bézier kurve til at afgøre, om kurven indeholder kritiske punkter. Se eventuelt afsnit 5.4.1. Det gøres blandt andet ved at undersøge *Hodografen* for kurven. I dette afsnit gøres rede for de to begreber afledt og hodograf for Bézier kurver.
Med introduktionen af den eksplicitte repræsentation af Bézier kurver på baggrund af Bernstein polynomierne i afsnit 2.3, er det klart, at $b^n : \mathbb{R} \to \mathbb{E}^2$ givet ved

$$b^n(t) = \sum_{j=0}^{n} b_j B_j^n(t)$$

er en differentiabel afbildning. Grunden er selvfølgelig, at Bernstein polynomierne er differentiable, se definition 2.3.1. Den afledte til Bézier kurven b^n med hensyn til t eksisterer og kan gives ved

$$\frac{d}{dt} b^n(t) = \frac{d}{dt} \sum_{j=0}^{n} b_j B_j^n(t) = \sum_{j=0}^{n} b_j \frac{d}{dt} B_j^n(t)$$

Den følgende sætning giver et udtryk for den afledte til det generelle Bernstein polynomium.

Sætning 2.4.1. *Lad $B_j^n : \mathbb{R} \to \mathbb{E}^2$ være det generelle Bernstein polynomium givet ved*

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \qquad \binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{hvis } 0 \le i \le b \\ 0 & \text{ellers} \end{cases}$$

Så er den afledte med hensyn til t givet ved

$$\frac{d}{dt} B_i^n(t) = n \left[B_{i-1}^{n-1}(t) - B_i^{n-1}(t) \right]$$

Bevis.

$$\begin{aligned} \frac{d}{dt} B_i^n(t) &= \frac{d}{dt} \binom{n}{i} t^i (1-t)^{n-i} \\ &= \frac{i \cdot n!}{i!(n-i)!} t^{i-1}(1-t)^{n-i} - \frac{(n-i)n!}{i!(n-i)!} t^i (1-t)^{n-i-1} \\ &= \frac{n(n-1)!}{(i-1)!(n-i)!} t^{i-1}(1-t)^{n-i} - \frac{n(n-1)!}{i!(n-i-1)!} t^i (1-t)^{n-i-1} \\ &= n \left[B_{i-1}^{n-1}(t) - B_i^{n-1}(t) \right] \end{aligned}$$

□

Så nu er det muligt at give et udtryk for den afledte med hensyn til t for en

Bézier.

$$\begin{aligned}\frac{d}{dt}b^n(t) &= \frac{d}{dt}\sum_{j=0}^{n} b_j B_j^n(t) = \sum_{j=0}^{n} b_j \frac{d}{dt} B_j^n(t) \\ &= \sum_{j=0}^{n} b_j \binom{n}{i} t^i (1-t)^{n-i} \\ &= n\sum_{j=1}^{n} b_j B_{j-1}^{m-1}(t) - n\sum_{j=0}^{n-1} b_j B_j^{n-1} \\ &= n\sum_{j=0} b_{j+1} B_j^{n-1} - n\sum_{j=0}^{n-1} b_j B_j^{n-1}(t) \\ &= n\sum_{j=0}^{n-1} (b_{j+1} - b_j) B_j^{n-1}(t)\end{aligned}$$

Så den afledte af en Bézier kurve med hensyn til t kan altså udtrykkes ved

$$\frac{d}{dt}b^n(t) = n\sum_{j_0}^{n-1} \Delta b_j B_j^{n-1}(t), \qquad \Delta b_j \in \mathbb{R}^2$$

hvor $\Delta b_j = b_{j+1} - b_j$ er den såkaldte *forward difference operator*. Den afledte til $b^n(t)$ ligner næsten en Bézier kurve med kontrol punkter $n\Delta b_0, n\Delta b_1, \ldots, n\Delta b_{n-1}$. Men dette er ikke helt korrekt, fordi $n\Delta b_j \in \mathbb{R}^2$ er nemlig vektorer som derfor ikke ligger i $\mathbb{E}^2$. Vælges $a \in \mathbb{R}^2$ (et oplagt valg er $a = 0$), så gælder $a + n\Delta b_j \in \mathbb{E}^2$ for $j = 0, 1, \ldots, n-1$ og den afledte kurve med kontrol punkter $a + n\Delta b_0, a + n\Delta b_1, \ldots, a + n\Delta b_{n-1}$ er så en Bézier kurve af grad $n-1$. Denne afledte kurve kaldes ofte for en hodograf.

2.5 Opdeling af en Bézier kurve

En Bézier kurve b^n er ofte defineret på intervallet $[0, 1]$, men den kan også defineres på ethvert andet interval $[0, c]$. Billedet $b^n([0, c])$ af b^n på intervallet $[0, c]$ kan også defineres ved en kontrol polygon. Processen af finde denne kontrol polygon kaldes for opdeling (subdivision) af Bézier kurver. De kontrol punkter c_i som der ledes efter, kan beregnes ved hjælp af deCasteljaus algoritme med respekt til $t = c$ og kan udtrykkes ved følgende formel

$$c_j = b_0^j(c) \tag{2.1}$$

som kaldes *opdelingsformlen (subdivision formula)* for Bézier kurver. Men hvad så med den del af b^n som er defineret på intervallet $[c, 1]$? Fra symmetri egenskaben for Bézier kurver, sætning 2.3.5, følger, at kontrol punkterne d_i til delkurven $[c, 1]$ er givet

$$d_j = b_j^{n-j}(c) \tag{2.2}$$

For at illustrere opdelingen af en Bézier kurve betragtes figur 2.1 ovenfor. På figuren er den kubiske Bézier kurve b^n givet ved kontrol punkterne b_0, b_1, b_2, b_3. På figurerne er også angivet de mellemliggende deCasteljau punkter, og det er netop fra disse, at de to kontrol polygoner, for henholdsvis delkurve c^n og delkurve d^n, stammer. Kontrol punkterne c_j er givet ved (2.1) og svarer overens med $b_0^0, b_0^1, b_0^2, b_0^3$ på figuren. På tilsvarende måde svarer d_j givet ved (2.2) overens med $b_0^3, b_1^2, b_2^1, b_3^0$.

2.6 Notation

I forbindelse med redegørelsen for Yaps algoritme gøres brug af en speciel notation for Bézier kurver. Blandt andet udregnes diameteren af en Bézier kurve, samt det konvekse hylster af kontrol polygoner. I det følgende beskrives den sprogbrug som benyttes i resten af specialet.

For en givet lukket og begrænset delmængde $S \subseteq \mathbb{E}^2$ defineres diameteren som følger.

Definition 2.6.1. *Lad $S \subseteq \mathbb{E}^2$ være en lukket og begrænset mængde. Så defineres diameteren af S til*

$$diam(S) = \sup\{\|p - q\| \mid p, q \in S\}$$

hvor $\|\cdot\|$ er den sædvanlige afstandsfunktion i $\mathbb{R}^2$. Det vil sige at $diam(S)$ udgør supremum over alle afstande mellem to punkter i S.

Det konvekse hylster af en lukket og begrænset mængde $S \subseteq \mathbb{E}^2$ er givet i [6, kap. 1.1] ved følgende.

Definition 2.6.2. *Lad $S \in \mathbb{E}^2$ være lukket og begrænset mængde. Så defineres det konvekse hylster af S som den mindste konvekse mængde der indeholder S. Det konvekse hylster for S betegnes $CH(S)$.*

Generelt vil Bézier kurver b^n blive betegnet med store bogstaver, startende med F. Kontrol polygonen for en Bézier kurve F vil så blive betegnet $P(F) = (p_0, p_1, \ldots, p_n)$ for en kurve med $n + 1$ kontrol punkter. $P(F)$ betragtes som en lukket (og begrænset) mængde i $\mathbb{E}^2$. Når intervallet, som Bézier kurven G er defineret på, er af særlig betydning, skrives ofte $G = G[a, b]$ eller blot $G[a, b]$. Bemærk at der gælder overalt i dette speciale, at $[a, b] \subseteq [0, 1]$. Når Bézier kurven F opdeles, betegnes delkurverne henholdsvis F_0 og F_1. Definitioner 2.6.1 er tiltænkt at gøre følgende notation kortere. Hvis F er en Bézier kurve i $\mathbb{E}^2$, så er $diam(F)$ supremum over alle afstande mellem punkter på kurven F. Funktionen $diam$ er defineret for alle lukkede og begrænsede mængder, så hvis F og G er Bézier kurver, gælder også, at $diam(F \cup G)$ udgør supremum af alle afstande mellem punkter på enten F eller G. Definition 2.6.2 benyttes på Bézier kurver på følgende måde. Hvis F er en Bézier kurve med tilhørende kontrol polygon $P(F)$, så er det konvekse hylster af F givet ved det konvekse hylster af kontrol polygonen, altså $CH(F) = CH(P(F))$.

Bigfloat tal og repræsentationer

Som nævnt i indledningen bliver de numeriske beregninger, i Yaps algoritme, reduceret til ring operationerne $(+, -, \times)$ på eksakte bigfloat tal. I dette afsnit gøres rede for, hvad bigfloat tal egentlig er, samt for hvilke tal der kan repræsenteres med bigfloat tal.

Definition 2.6.3. *Et tal $x \in \mathbb{Q} \subseteq \mathbb{R}$ kaldes et bigfloat tal, hvis der gælder*

$$x = n2^m, \qquad n, m \in \mathbb{Z}$$

Som det kan ses af følgende lille omskrivning af definition 2.6.3, så udgør bigfloat tal de rationale tal, som er brøker af typen

$$x = \frac{n}{2^i}, \qquad n \in \mathbb{Z}, \qquad i \in \mathbb{Z} \setminus \{0\}.$$

Så der er ikke tale om generelle rationale tal. For eksempel $y = 1/3$ kan ikke repræsenteres ved et bigfloat tal, fordi der for ethvert $i \in \mathbb{Z}$ gælder $3 \neq 2^i$. Men der gælder derimod, at for ethvert rationalt tal z og fejlmargen $\epsilon > 0$ findes et bigfloat tal z', således at $|z - z'| < \epsilon$. Dette følger fra sætning 2.6.4 nedenfor.

Sætning 2.6.4. *Ethvert åbent interval $(a, b) \subseteq \mathbb{R}$ $(a < b)$ indeholder et rationalt tal af formen $x = n2^m$.*

Bevis. Vælg et tal $m' \in \mathbb{N}$ således at $1/2^{m'} < (b - a)$. Tallet m' findes, fordi for ethvert $x \in \mathbb{R}, x > 0$ gælder at $2^{\lfloor \log_2 x \rfloor} \leq x$, så sæt $m = 2^{\lfloor \log_2 x \rfloor}$.
Da $m < (b - a)$ må der findes et $n \in \mathbb{Z}$ således at $n/m \in (a, b)$. Så der gælder at $n2^m \in (a, b)$. □

Ring operationerne $(+, -, \times)$ (og division med 2) er eksakte i den forstand, at henholdsvis sum, differens, og produkt mellem bigfloat tal resulterer i nye bigfloat tal. Følgende sætning præciserer denne påstand.

Sætning 2.6.5. *Hvis $\mathbb{B} = \{n2^m \mid n, m \in \mathbb{Z}\} \subset \mathbb{Q}$ er mængden af alle bigfloat tal. Så gælder $x + y, x - y, x \times y, x/2 \in \mathbb{B}$.*

Bevis. Lad $x = n_1 2^{m_1} \in \mathbb{B}$ og $y = n_2 2^{m_2} \in \mathbb{B}$. Antag $m_1 \leq m2$. Så gælder

$$\begin{aligned} x \pm y &= n_1 2^{m_1} \pm n_2 2^{m_2} = (n_1 \pm n_2 2^{m2-m1}) 2^{m_1} \in \mathbb{B} \\ x \times y &= n_1 2^{m_1} \times n_2 2^{m_2} = (n_1 n_2) 2^{m1+m1} \in \mathbb{B} \\ x/2 &= n2^{m-1} \end{aligned}$$

□

Så hvis et givet udtryk $E(x, y, z)$ kun afhænger af $x, y, z \in \mathbb{B}$ og regne operationerne $(+, -, \times)$ (eller division med 2), så gælder ifølge sætning 2.6.5 at $E(x, y, z) \in \mathbb{B}$. Denne egenskab ligger til grund for Yaps anvendelse af bigfloat tal i sin algoritme beskrevet i kapitel 5.

I forbindelse med definitionen af geometriske separationsgrænser, blandt andet i kapitel 4, er det nødvendigt at indføre et udtryk for (bit-)kvaliteten af bigfloat repræsentationen. Så derfor indføres de såkaldte (L, l)-bit floating point tal.

Definition 2.6.6. *Lad $x = m2^{k-L} \in \mathbb{B}$, $n, k \in \mathbb{Z}$ og $L \in \mathbb{N}$, hvor der gælder $|m| < 2^L$ og $0 \leq k \leq l$. Så kaldes x for et (L, l)-bit floating point tal. Bemærk, at der gælder at $-L \leq \log_2 |x| < l$.*

Direkte og indirekte objekter

For at understrege anvendelsen af bigfloat tal i resten af specialet indføres følgende notation. Antag at punkter, linier og Bézier kurver er repræsenteret ved standard parametre. Et punkt p er givet ved koordinaterne $p = (x, y)$, en linie l er givet ved koefficienterne til dens ligning $l : ax + by + c$ og en Bézier kurve er givet ved dens kontrol punkter (som er givet ved koordinater). Når parametre, såsom x,y,a,b,c osv., er givet ved bigfloat tal, kaldes punkter, linier og kurver for *direkte* objekter. I kontrast til *indirekte* objekter, som ikke nødvendigvis er repræsenteret ved bigfloat tal, men derimod generelle rationale eller algebraiske tal (se afsnit 3.1 for en nærmere beskrivelse af algebraiske tal). For eksempel kan skæringspunktet p mellem en direkte linie l og en direkte Bézier kurve F være et punkt, hvis koordinater ikke er bigfloat tal, så p er derfor et indirekte punkt. Det er så nødvendigt at finde en metode til at repræsentere og estimere indirekte objekter på. Den generelle løsning er at indføre udtryk over direkte objekter. For eksempel, hvis F og l skærer hinanden i et entydigt punkt p, så introduceres følgende implicitte repræsentation $Point[F, l]$ for $p = F \cap l$. For at estimere værdien af p antages, at $F = \{F(t) \mid t \in [0, 1]\}$. Så kan to bigfloat tal t_0, t_1 gives således at, $p = F(t)$ og $0 \leq t_0 \leq t \leq t_1 \leq 1$. Så udgør udtrykket $Point[F, l, t_0, t_1]$ en estimering af skæringspunktet p. Ydermere kan estimatet $Point[F, l, t_0, t_1]$ forfines vilkårligt meget ved at indsnævre intervallet $[t_0, t_1]$.

Kapitel 3

Algebraiske kurver og geometriske separationsgrænser

Algebraiske kurver er en generel klasse af kurver baseret på polynomier i to variable. Bézier kurver udgør en underklasse af algebraiske kurver, hvilket vil sige, at der findes algebraiske kurver, som ikke er Bézier kurver, men alle Bézier kurver er algebraiske kurver. I dette kapitel indføres et vigtigt analytisk redskab, som bruges i stor udstrækning i Yaps algoritme [4]. Dette redskab er de såkaldte *geometriske separationsgrænser* for algebraiske kurver. Disse grænser kan blandt andet besvare spørgsmål som "*Hvor tæt kan to kurver være uden at skære hinanden?*" eller "*Hvor tæt kan et punkt være på en kurve, når punktet ikke ligger på kurven?*".
Afsnit 3.1 introducerer polynomier, algebraiske tal og kurver, mens afsnit 3.2 indfører de geometriske separationsgrænser. Afsnit 3.3 beskriver, hvorledes grænserne, fra afsnit 3.2, kan beregnes, når kurverne er Bézier kurver.

3.1 Polynomier, algebraiske tal og kurver

Før de algebraiske tal og kurver kan defineres formelt, er det nødvendigt at introducere et par definitioner og lidt terminologi fra algebra. Dette afsnit indeholder alle de informationer omkring polynomier, algebraiske tal og kurver, som der gøres brug af i specialet.

Definition 3.1.1. *Lad R være en ring. Så er $R[X]$ en ring over polynomier med koefficienter fra ringen R, hvor $X = (x_1, \ldots, x_d)$, for $d \geq 1$.*

Polynomiet $P \in R[X]$ kaldes henholdsvis et heltals-, rationelt, reelt eller komplekst polynomium afhængig af, om R er $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, eller $\mathbb{C}$.

Definition 3.1.2. *Lad $R_0 \subseteq R_1$ være ringe og lad polynomiet $P \in R_0[X]$ være givet. Et element, $\alpha \in R_1$, der løser ligningen*

$$P(X) = 0$$

kaldes et nulpunkt til polynomiet P og mængden $Zero(P) = \{\alpha \in R_1 \mid P(\alpha) = 0\}$ er mængden af alle sådanne nulpunkter.

Sætning 3.1.3. *Hvis $P \in \mathbb{Q}[X]$ er et rationelt polynomium, så findes der et tal $c \in \mathbb{Z}$, således at polynomiet $Q(X) = c \cdot P(X) \in \mathbb{Z}[X]$ er et heltalspolynomium, og der gælder*

$$Zero(Q) = Zero(P)$$

Bevis. Lad $c \in \mathbb{Z}$ være produktet af koefficienterne til polynomiet P. Antag $\alpha \in Zero(P)$ være et nulpunkt for polynomiet P. Så gælder

$$Q(\alpha) = c \cdot P(\alpha) = c \cdot 0 = 0$$

og dermed er α også et nulpunkt i polynomiet Q. □

Definition 3.1.4 (**Algebraiske tal**). *Lad $R_0 \subseteq R_1$ være ringe. Så kaldes tallet $\alpha \in R_1$ et algebraisk tal, hvis der gælder, at α er et nulpunkt i den polynomielle ligning*

$$P(x) = \sum_{i=0}^{n} x^i = a_0 + a_1 x + \cdots + a_n x^n = 0, \;\; a_n \neq 0$$

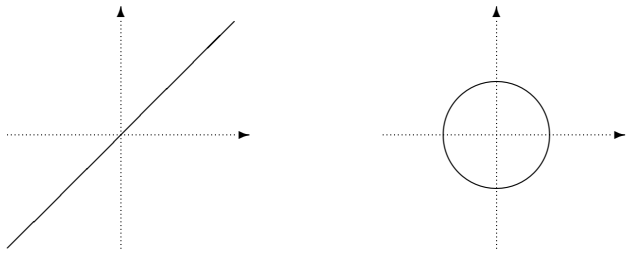
hvor $a_i \in R_0$ for alle $i = 0, 1, 2, \ldots, n$.

Definition 3.1.5 (**Algebraisk kurve**). *Lad $R_0 \subseteq R_1$ være ringe. Lad $K = \{(\alpha, \beta) \in R_1^2 \mid P(\alpha, \beta) = 0\}$ være løsningsmængden til den polynomielle ligning*

$$\begin{aligned} P(x, y) &= \sum_{i=0}^{n} \sum_{j=0}^{m} a_{ij} x^i y^j \\ &= a_{mn} x^n y^m + \cdots + a_{11} xy + a_{10} x + a_{01} y + a_{00} \end{aligned}$$

hvor $a_{ij} \in R_0$ for $i = 0, 1, 2, \ldots, n$, $j = 0, 1, 2, \ldots, m$. Så kaldes K for en algebraisk kurve.

For polynomium $P \in R_0[x, y]$ udgør den algebraiske kurve, $K = \{(\alpha, \beta) \in R_1^2 \mid P(\alpha, \beta) = 0\}$, ofte en uendelig mængde. For eksemepel, når $R_0 = \mathbb{Z}$ og $R_1 = \mathbb{R}$ så er K en kurve i planen, $\mathbb{R}^2$.
På figur 3.1 nedenfor vises to konkrete eksempler på algebraiske kurver i planen. Figur 3.1(a) viser en ret linie gennem $(0, 0)$ med hældning 1. Punkterne på denne linie opfylder den polynomielle ligning $A(x, y) = x - y = 0$. På tilsvarende måde er cirklen, på figur 3.1(b), med centrum i $(0, 0)$ med radius r løsningsmængden til den polynomielle ligning $B(x, y) = x^2 + y^2 - r^2 = 0$.



(a) $A(x, y) = x - y$ (b) $B(x, y) = x^2 + y^2 - r^2$

Figur 3.1: Kendte algebraiske kurver

3.2 Geometriske seperationsgrænser

Lad $A(x, y), B(x, y) \in \mathbb{Z}[x, y]$ være polynomier af to variable og betragt kurverne F og G, som henholdsvis er defineret ved ligningerne $A(x, y) = 0$ og $B(x, y) = 0$. Men før de enkelte geometriske separationsgrænser kan indføres, behøves et par definitioner.

Definition 3.2.1. *Lad $A, B \in \mathbb{Z}[X], X = (x_0, x_1, \ldots, x_d), d \geq 1$ være to polynomier. Så kaldes A og B primske, hvis der gælder at $gcd(A, B) = 1$.*

Definition 3.2.2. *Lad F og G være to algebraiske kurver defineret ved ligningerne $A(x, y) = 0$ og $B(x, y) = 0$. Et par (p, q), hvor linien gennem $p \in F$ og $q \in G$ er normal til F i punktet p og normal til G i q, kaldes et (F, G)-antipodalpar.*

Specielt gælder, at hvis kurverne F og G skærer hinanden tangentielt i punktet p, vil parret (p, p) være et (F, G)-antipodalpar.
I det følgende gøres brug af antagelsen, at der for et par af kurver, F og G, kun findes endeligt mange (F, G)-antipodalpar. Dette er ensbetydende med, at der for polynomierne A og B gælder, at de ikke har nogle fælles komponenter, i.e. de er primske.

I beviserne for de geometriske separationsgrænser nedenfor gøres der stærkt brug af en sætning fra bogen [7, Sætning 11.45], så den nævnes her, dog uden bevis.

Sætning 3.2.3. *Lad $\Sigma = \{A_1, \ldots, A_n\} \subseteq \mathbb{Z}[x_1, x_2, \ldots, x_n]$ være et system af polynomier, i n variable, som ikke nødvendigvis er homogene. Antag, at Σ har endelig mange komplekse nulpunkter, og lad $(\xi_1, \ldots, \xi_n) \in \mathbb{C}^n$ være et af disse nulpunkter. Lad også $d_i = \deg(A_i)$ for $i = 1, 2, \ldots, n$ og lad*

$$K = \max\{\sqrt{n+1}, \max\{\|A_i\|_2 \mid i = 1, 2, \ldots, n\}\}$$

Hvis $|\xi_i| \neq 0$ så gælder at

$$|\xi_i| > (2^{2/3} NK)^{-D} 2^{-(n+1)d_1 \cdots d_n}$$

hvor

$$N = \binom{1 + \sum_{i=1}^{n} d_i}{n}, \qquad D = \left(1 + \sum_{i=1}^{n} \frac{1}{d_i}\right) \prod_{i=1}^{n} d_i$$

Sætningen 3.2.3 gør brug af $\|\cdot\|_2$-normen til et polynomium i to variable. Denne norm er defineret således.

Definition 3.2.4. *Lad $A(x, y) \in \mathbb{Z}[x, y]$ være et polynomium i to variable og lad*

$$v = (a_{00}, a_{01}, \ldots, a_{10}, a_{11}, \ldots, a_{mn})$$

være vektoren i $\mathbb{R}^{(n+1)(m+1)}$ af koefficienterne til A. Så er k-normen af A defineret som

$$\|A\|_k = \|v\|_k$$

hvor $\|\cdot\|_k$ på højresiden er den sædvanlige k-norm fra $\mathbb{R}^{(n+1)(m+1)}$.

Med sætning 3.2.3 er det nu muligt at bevise de specifikke geometriske separationsgrænser, som der blandt andet gøres brug af i algoritmen i kapitel 5. Den første grænse giver den minimale afstand mellem to forskellige punkter, som udgør et (F, G)-antipodalpar.

Sætning 3.2.5. *Lad $A(x, y), B(x, y) \in \mathbb{Z}[x, y]$ og lad F og G være kurverne defineret ved henholdsvis $A(x, y) = 0$ og $B(x, y) = 0$. Antag, at F og G har endelig mange (F, G)-antipodalpar, og lad $m = \deg(A), n = \deg(B), \|A\|_2 = a, \|B\|_2 = b$. Hvis (p, q) er et (F, G)-antipodalpar og $p \neq q$, så er*

$$\|p - q\| \geq \Delta_1(m, n, a, b) = (2^{\frac{3}{2}} NK)^{-D} 2^{-12m^2n^2})$$

hvor

$$K = \max\{\sqrt{13}, 4ma, 4nb\}, \qquad N = \binom{3 + 2m + 2n}{5}, \qquad D = m^2n^2(3 + \frac{4}{m} + \frac{4}{n})$$

Bevis. Fra definitionen af (F,G)-antipodalpar opfylder $p = (p.x, p.y) \in F$, $q = (q.x, q.y) \in G$ og $h = \|p-q\|$ det følgende system af polynomielle ligninger af fem variable $(p.x, p.y, q.x, q.y, h)$

$$\begin{aligned}
&\mathcal{A}_1 : A(p) = 0\\
&\mathcal{A}_2 : B(q) = 0\\
&\mathcal{A}_3 : h^2 - \|p-q\|^2 = 0\\
&\mathcal{A}_4 : \left\langle (\frac{dA}{dy}(p), -\frac{dA}{dx}(p)), (p-q) \right\rangle = 0\\
&\mathcal{A}_5 : \left\langle (\frac{dB}{dy}(q), -\frac{dB}{dx}(q)), (p-q) \right\rangle = 0
\end{aligned}$$

hvor $\langle\cdot\rangle$ er et indre produkt i $\mathbb{R}^2$. Da F og G har endeligt mange (F,G)-antipodalpar, er systemet ovenfor 0-dimensionalt. For at bruge sætning 3.2.3 skal 2-normen af de fem ovenstående polynomier beregnes. $\|\mathcal{A}_1\|_2 = a$ og $\|\mathcal{A}_2\|_2 = b$ og $\|\mathcal{A}_3\|_2 = \sqrt{13}$. Lad $A_x(p) = \frac{dA}{dx}(p)$ og $A_y(p) = \frac{dA}{dy}(p)$ så gælder $\|A_x\|_2 \le ma$ og $\|A_y\|_2 \le ma$. For polynomium $\mathcal{A}_4$ gælder derfor

$$\begin{aligned}
\|\mathcal{A}_4\|_2 &= \|A_y(p)\cdot(p.x-q.x) - A_x(p)\cdot(p.y-q.y)\|_2\\
&\le \|A_y(p)\cdot p.x\| + \|A_y(p)\cdot q.x\| + \|A_x(p)\cdot p.y\| + \|A_x(p)\cdot q.y\|_2\\
&\le 2ma + 2ma\\
&= 4ma
\end{aligned}$$

På tilsvarende måde gælder

$$\|\mathcal{A}_5\|_2 = \|B_y(p)\cdot(p.x-q.x) - B_x(p)\cdot(p.y-q.y)\|_2 \le 2na + 2na = 4na$$

Til sidst bemærkes, at $d_1 = d_3 = m$, $d_2 = d_4 = n$ og $d_5 = 2$, og så kan normerne umiddelbart sættes ind i sætning 3.2.3. □

Den næste separationsgrænse giver den minimale afstand mellem to forskellige skæringspunkter for et par af kurver.

Sætning 3.2.6. *Lad $A(x,y), B(x,y) \in \mathbb{Z}[x,y]$ være to primske polynomier, hvor $m = \deg(A), n = \deg(B), \|A\|_2 = a, \|B\|_2 = b$.*
Hvis p og q, $p \neq q$, er to forskellige punkter på henholdsvis $A(x,y) = 0$ og $B(x,y) = 0$, så gælder

$$\|p-q\| \ge \Delta_2(m,n,a,b) = (2^{\frac{3}{2}}NK)^{-D}2^{-12m^2n^2})$$

hvor

$$K = \max\{\sqrt{13}, m, n\}, \qquad N = \binom{3+2m+2n}{5}, \qquad D = m^2n^2(3+\frac{4}{m}+\frac{4}{n})$$

Bevis. Punkter $p = (p.x, p.y)$, $q = (q.x, q.y)$ og afstanden $h = \|p-q\|$ opfylder følgende system af polynomielle ligninger af fem variable $(p.x, p.y, q.x, q.y, h)$

$$\begin{aligned}
&\mathcal{A}_1 : A(p) = 0\\
&\mathcal{A}_2 : B(q) = 0\\
&\mathcal{A}_3 : h^2 - \|p-q\|^2 = 0\\
&\mathcal{A}_4 : A(q) = 0\\
&\mathcal{A}_5 : B(p) = 0
\end{aligned}$$

Da A og B er antaget primske, bliver systemet overfor 0-dimensionalt. Som ovenfor gælder, at $\|\mathcal{A}_1\|_2 = \|\mathcal{A}_4\|_2 = a$, $\|\mathcal{A}_2\|_2 = \|\mathcal{A}_5\|_2 = b$ og $\|\mathcal{A}_3\|_2 = \sqrt{13}$, så anvendelsen af sætning 3.2.3 er derfor ligefrem. □

For at give den næste grænse er det nødvendig at vide hvordan, eller mere præcist med hvilken præcision, er punktet og kurven givet. Dette leder frem til følgende definition.

Definition 3.2.7. *Et tal $x \in \mathbb{R}$ kaldes for et (L,l)-bit flydende kommatal, hvis der gælder følgende.*

$$x = m2^{k-L}, \;\; |m| < 2^L, \;\; 0 \leq k \leq l$$

hvor $L, l \in \mathbb{N}$ og $m, k \in \mathbb{Z}$. Hvis $L = l$ kaldes x for et L-bit flydende kommatal. Bemærk, at der gælder $-L \leq log_2|x| < l$.

Den sidste separationsgrænse beskriver den mindste afstand fra en algebraisk kurve, $A(x,y) = 0$, til et punkt, p, hvorom der gælder, at $A(p) \neq 0$, hvilket vil sige, at p ikke ligger på kurven.

Sætning 3.2.8. *Lad $A(x,y) = 0$ være en algebraisk kurve og $q = (u,v) \in \mathbb{R}^2$ være et punkt, hvis koordinater er givet med (L,l)-bit flydende kommatal, hvor $l \geq 2$. Antag yderligere, at $A(u,v) \neq 0$.*
Hvis kurven $A(x,y) = 0$ ikke udgør en cirkel med centrum i q, og p er et punkt på kurven, så gælder

$$\|p - q\| \geq \Delta_3(m,a,L,l) = (2^{\frac{3}{2}}NK)^{-D}2^{-8m^2}$$

hvor

$$K = 2^{L+l+1}\max\{2^L, 2ma\}, \qquad N = \binom{3+2m}{5}, \qquad D = m^2(3 + \frac{4}{m}).$$

Bevis. Antag umiddelbart, at p er det punkt på kurven, som ligger tættest på q, og lad $h = \|p-q\|$. Så de tre variable $p.x, p.y, h$ opfylder følgende system af polynomielle ligninger af tre variable.

$$\begin{aligned}
&\mathcal{A}_1 : A(p) = 0 \\
&\mathcal{A}_2 : h^2 - \|p-q\|^2 = 0 \\
&\mathcal{A}_3 : \left\langle (\frac{dA}{dy}(p), -\frac{dA}{dx}(p)), (p-q) \right\rangle = 0
\end{aligned}$$

Dette system er 0-dimensionalt, da $A(x,y) = 0$ ikke indeholder en cirkel omkring q. For at bruge sætning 3.2.3 skal 2-normen for de tre polynomier, $\mathcal{A}_i$, vurderes. $\|\mathcal{A}_1\| = a$ er givet fra $A(x,y) = 0$. $\|\mathcal{A}_2\| \leq \sqrt{1 + 2(2^{2l} + 2^{l+1} + 1)} < 2^{1+l}$ under antagelse af at $l \geq 2$.
For at vurdere $\|\mathcal{A}_3\|$ lad $A_x(p) = \frac{dA}{dx}(p)$ og $A_y(p) = \frac{dA}{dy}(p)$. Så gælder

$$\begin{aligned}
\|\mathcal{A}_3\|_2 &= \|A_y(p) \cdot (p.x - q.x) - A_x(p) \cdot (p.y - q.y)\|_2 \\
&\leq \|A_y(p) \cdot (p.x - q.x)\|_2 + \|A_x(p) \cdot (p.y - q.y)\|_2 \\
&\leq \|A_y(p) \cdot p.x\|_2 + \|A_y(p) \cdot q.x\|_2 + \|A_x(p) \cdot p.y\|_2 + \|A_x(p) \cdot q.y)\|_2 \\
&\leq ma + 2^l ma + ma + 2^l ma \\
&= 2ma(1 + 2^l) \\
&< 2ma2^{l+1}
\end{aligned}$$

Men istedet for polynomierne $\mathcal{A}_2$ og $\mathcal{A}_3$ bruges $2^{2L}\mathcal{A}_2$ og $2^L\mathcal{A}_3$ så $\|2^{2L}\mathcal{A}_2\|_2 < 2^{2L+l+1}$ og $\|2^L\mathcal{A}_3\|_2 = 2ma(1 + 2^l)2^L < 2ma2^{L+l+1}$. Herefter kan $\|\mathcal{A}_i\|_2$ umiddelbart sættes ind i sætning 3.2.3. □

Det bemærkes her, at der kan ses bort fra kravet, i sætning 3.2.8, om, at kurven ikke udgør en cirkel med center i punktet q, da Bézier kurver ikke kan være cirkulære.

Definition 3.2.9 (**Δ-separationsegenskaben**). *Lad F, G være Bézier kurver. Et par $(p, q) \in F \times G$ kaldes et kritisk par for (F, G), hvis der gælder mindst en af følgende betingelser*

- *$(p, q) \in F \times G$ er et (F, G)-antipodalpar*
- *$p \in F \cap G$ og $q \in F \cap G$*

Givet $\Delta > 0$ siges kurverne, F og G, at have Δ-separationsegenskaben, hvis ethvert kritisk par (p, q), hvor $p \neq q$, opfylder $\|p - q\|_2 > \Delta$.

3.3 Vurdering af $\|\cdot\|_2$-normen for Bézier kurver

Bézier kurver er, som nævnt tidligere, en speciel klasse af algebraiske kurver, så de geometriske separationsgrænser for algebraiske kurver, udledt i sektion 3.2, gælder også for Bézier kurver. Men de kurver, som dette speciale beskæftiger sig med, er givet ved en kontrol polygon, hvorimod algebraiske kurver generelt er givet ved en polynomiel ligning i to variable. Så for at bruge sætningerne fra afsnit 3.2 er det nødvendigt at kunne vurdere den øvre grænse for $\|\cdot\|_2$-normen for en Bézier kurve givet ved en kontrol polygon. Dette gribes an ved at udlede den implicitte algebraiske ligning for en givet Bézier kurve og så vurdere $\|\cdot\|_2$-normen for denne algebraiske kurve.

Lad i det følgende F være en Bézier kurve givet ved kontrol polygonen $P(F) = (p_0, p_1, \ldots, p_m)$. Så kan koordinatfunktionerne til F udregnes på følgende måde

$$\begin{aligned}
F(t) &= \sum_{i=0}^{m} p_i \binom{m}{i} t^i (1-t)^{m-i} \\
&= \sum_{i=0}^{m} p_i \binom{m}{i} t^i \left(\sum_{j=0}^{m-i} t^j (-1)^j \binom{m-i}{j} \right) \\
&= \sum_{i=0}^{m} \sum_{j=0}^{m-i} \binom{m}{i,j} t^{i+j} (-1)^j p_i \qquad (3.1) \\
&= \sum_{k=0}^{m} t^k \sum_{l=0}^{k} p_l \binom{m}{l, m-k} (-1)^{k-l} \\
&= \sum_{k=0}^{m} t^k \begin{bmatrix} a_k \\ b_k \end{bmatrix}
\end{aligned}$$

hvor $\binom{m}{i,j} = \frac{m!}{i!j!(m-i-j)!}$. Bemærk, at den sidste ligning blot definerer konstanterne a_k og b_k for $k = 0, 1, 2, \cdots, m$. Kurven kan nu skrives $F(t) = (F_x(t), F_y(t))$, hvor

$$F_x(t) = \sum_{k=0}^{m} a_k t^k, \qquad F_y(t) = \sum_{k=0}^{m} b_k t^k$$

For koefficienterne a_k (og b_k) for $k = 0, 1, \ldots, m$ i koordinatfunktionerne til F gælder følgende sætning angående koefficienternes størrelse.

Lemma 3.3.1. *Lad $F(t) = (F_x(t), F_y(t))$ være en Bézier kurve og antag at koordinaterne til kontrol polygonen $P(F) = (p_0, \cdots, p_m)$ er givet ved (L,l)-bit floating-point værdier. Så er $2^L a_k$ er et heltal og $|a_k| \leq 2^{L+k}\binom{m}{k}$ for $k = 0, \cdots, m$. En tilsvarende grænse findes for b_k.*

Bevis. At $2^L a_k \in \mathbb{Z}$ følger direkte fra definitionen af (L,l)-bit flydende kommatal. Hvis x er et (L,l)-bit flydende kommatal, gælder at $x = m2^{k-L}$, hvor $m, k \in \mathbb{Z}$ og $|m| < 2^L$ for $0 \leq k \leq l$. Så er $2^L x = m2^k$ et heltal.

$$|a_k| = \left|\sum_{l=0}^{k}(-1)^{k-l}p_l.x\binom{m}{l, m-k}\right| \leq 2^L \sum_{l=0}^{k}\binom{m}{l, m-k} = 2^L\binom{m}{k}\sum_{l=0}^{k}\binom{k}{l} = 2^{L+k}\binom{m}{k}$$

□

Lad nu $F(t) = (F_x(t), F_y(t))$ være en givet Bézier kurve, så er kurvens implicitte algebraiske ligning givet ved $B(x, y) = 0$, hvor $B(x, y) = res_t(F_x(t) - x, F_y(t) - y)$. Da $F_x(t) = \sum_{k=0}^{m} a_k t^k$ og $F_y(t) = \sum_{k=0}^{m} b_k t^k$ gælder det at $B(x, y) = res_t(F_x(t) - x, F_y(t) - y)$, jævnfør [7, afsnit 3.3], udgør determinanten af følgende matrix.

$$M = \begin{bmatrix} a_m & a_{m-1} & \cdots & a_1 & a_0 - x & \cdots & \\ & a_m & \cdots & a_2 & a_1 & \cdots & \\ & & \ddots & & & \cdots & \\ & & & a_m & a_{m-1} & \cdots & a_0 - x \\ b_m & b_{m-1} & \cdots & b_1 & b_0 - y & \cdots & \\ & b_m & \cdots & b_2 & b_1 & \cdots & \\ & & \ddots & & & \cdots & \\ & & & b_m & b_{m-1} & \cdots & b_0 - y \end{bmatrix}$$

Da a_i og b_i for $i = 0, 1, \ldots, m$ er konstanter og x, y er variable, vil determinanten til matrix M være et polynomielt udtryk, $B(x, y)$, i to variable, og $B(x, y) = det(M) = 0$ vil derfor være en algebraisk kurve. Bemærk, at $B(x, y) \in \mathbb{Q}$ er muligvis et rationelt polynomium, men sætning 3.1.3 gør det mulig at finde et heltalspolynomium, $A(x, y)$, med samme løsningsmængde som $B(x, y)$. Den følgende sætning giver en øvre grænse for $\|\cdot\|_2$-normen af den implicitte algebraiske kurve for en givet Bézier kurve.

Sætning 3.3.2. *Lad F være en Bézier kurve givet ved kontrol polygonen, $P(F) = (p_0, p_1, \ldots, p_m)$, hvor koordinaterne til punkterne p_i er givet L-bit flydende kommatal. Så opfylder F en polynomiel ligning $A(x, y) = 0$, hvor $A \in \mathbb{Z}[x, y]$ og $\deg(A) = m$, og hvor der gælder*

$$\|A\|_2 \leq (16^L 9^m)^m$$

Bevis. Kurven $F(t) = (F_x(t), F_y(t))$ opfylder ligningen $B(x, y) = 0$, hvor $B(x, y) = det(M)$ og M er givet i (3.3). Tallene i matrix M er givet ved L-bit flydende kommatal, så $2^L M$ er en matrix med heltalsindgange. Lad $A(x, y) = det(2^L M)$ være et heltalspolynomium. Den generaliserede Hadamard-grænse, fra [7, afsnit 6.8], fortæller, at $\|A\|_2$ er begrænset af produktet af $\|\cdot\|_2$-normerne til hver af rækkerne i matrix $2^L M_1$. Matrix M_1 er givet ved at betragte alle indgangene i matrix M som polynomier, $P_{ij} \in \mathbb{Z}[x, t]$, og derefter udskifte hver P_{ij} med en øvre grænse til $\|P_{ij}\|_1$. Hvis grænserne for koefficienterne i sætning 3.3.1 benyttes, bliver M_1 som nedenfor.

$$M_1 = \begin{bmatrix} 2^{L+m}\binom{m}{m} & 2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L+1}\binom{m}{1} & 2^{L}\binom{m}{0}+1 & \cdots & \\ & 2^{L+m}\binom{m}{m} & \cdots & 2^{L+2}\binom{m}{2} & 2^{L+1}\binom{m}{1} & \cdots & \\ & & \ddots & & & \cdots & \\ & & & 2^{L+m}\binom{m}{m} & 2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L}\binom{m}{0}+1 \\ 2^{L+m}\binom{m}{m} & 2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L+1}\binom{m}{1} & 2^{L}\binom{m}{0}+1 & \cdots & \\ & 2^{L+m}\binom{m}{m} & \cdots & 2^{L+2}\binom{m}{2} & 2^{L+1}\binom{m}{1} & \cdots & \\ & & \ddots & & & \cdots & \\ & & & 2^{L+m}\binom{m}{m} & 2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L}\binom{m}{0}+1 \end{bmatrix}$$

$\|\cdot\|_2$-normen til hver række i matrix $2^L M_1$ kan nu vurderes til $2^{2L}3^m$ (se sætning 3.3.3 nedenfor), og da der er $2m$ rækker i matrix $2^L M_1$, og der gælder

$$\|A\|_2 \leq (2^{2L}3^m)^{2m} = (16^L 9^m)^m$$

□

Lemma 3.3.3. *Lad matrix M_1 være givet som ovenfor og lad r_i være en række i denne matrix. Så gælder*

(i) $\|r_i\|_1 \leq 1 + 2^L 3^m$

(ii) $\|r_i\|_2 \leq 2^L 3^m$

Bevis. Beviserne nedenfor udnytter følgende ligning $\sum_{k=0}^{m} 2^k\binom{m}{k} = 3^m$.
(i)

$$\|r_i\|_1 = \left|1 + 2^L\right| + \sum_{k=1}^{m} \left|2^{L+k}\binom{m}{k}\right| = 1 + 2^L \sum_{k=0}^{m} 2^k \binom{m}{k} = 1 + 2^L 3^m$$

(ii) Beviset nedenfor udnytter også uligheden $\sqrt{x^2+y^2} \leq x + y - 1$ for $x \geq 2$ og $y \geq 2$ i anden linie.

$$\begin{aligned} \|r_i\|_2 &= \sqrt{(1+2^L)^2 + \sum_{k=1}^{m} \left(2^{L+k}\binom{m}{k}\right)^2} \\ &\leq (1+2^L) + \sqrt{\sum_{k=1}^{m} \left(2^{L+k}\binom{m}{k}\right)^2} - 1 \\ &\leq 2^L + \sum_{k=1}^{m} \sqrt{\left(2^{L+k}\binom{m}{k}\right)^2} \\ &= 2^L \sum_{k=0}^{m} 2^k \binom{m}{k} \\ &= 2^L 3^m \end{aligned}$$

□

Kapitel 4

Elementære kurver og par

Lad i det følgende $f : [c,d] \mapsto \mathbb{R}$ være en begrænset og kontinuert differentiabel funktion fra et interval ind i de reelle tal. Grafen for f er den parametriserede kurve $F = \{F(t) : t \in [c,d]\}$, hvor $F(t) = (t, f(t)) \in \mathbb{R}^2$. Lad desuden $\mathcal{G}[c,d]$ betegne mængden af alle grafer for begrænsede, kontinuert differentiable funktioner $g : [c,d] \mapsto \mathbb{R}$. Et element i $\mathcal{G}[c,d]$ kaldes herefter for en *graf parametrisering*. For at simplificere notationen nedenfor, antages at $[c,d] = [0,1]$.

4.1 Elementær kurver

En kurve $F \in \mathcal{G}[c,d]$ kaldes en *elementær kurve*, hvis den udgør grafen for en *konveks* eller en *konkav* funktion i $\mathbb{R}$. Liniestykket, som forbinder $F(c)$ og $F(d)$ kaldes for *basesegmentet* af F. Hvis F er en elementær kurve, kan den yderligere klassificeres som enten *A-elementær* eller *B-elementær* afhængig af, om den udgør grafen for en konkav eller en konveks funktion. Bemærk, at hvis $F \in \mathcal{G}[c,d]$, og F samtidig er konkav, så vil $F(t)$ ligge over (*Above*) basesegmentet for alle $t \in [c,d]$. Tilsvarende vil $F(t)$ ligge under (*Below*) basesegmentet, hvis $F \in \mathcal{G}[c,d]$ er konveks.

Tangentvektoren til F i punktet $F(t)$ er $(1, f'(t)) \in \mathbb{R}^2$, og derfor er enhedsnormalvektoren, $u_F : [c,d] \mapsto \mathbb{R}^2$, i samme punkt givet ved

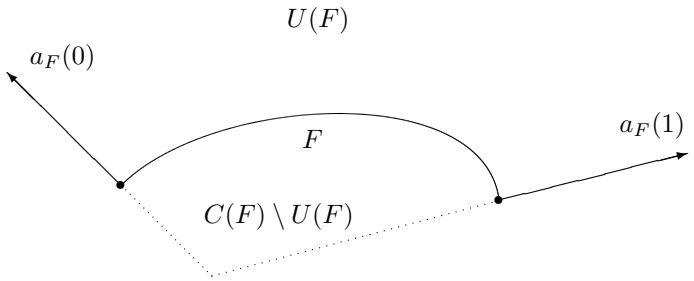
$$\begin{aligned} u_F(t) &= \overbrace{\left(\frac{1}{\sqrt{1+f'(t)^2}}, \frac{f'(t)}{\sqrt{1+f'(t)^2}} \right)} \\ &= \left(\frac{-f'(t)}{\sqrt{1+f'(t)^2}}, \frac{1}{\sqrt{1+f'(t)^2}} \right) \\ &= (\cos\theta_F(t), sin\theta_F(t)) \end{aligned} \tag{4.1}$$

hvor funktionen $\theta_F : [c,d] \mapsto \mathbb{R}$ kaldes for normalvinklen i punktet $F(t)$. Det følger fra (4.1), at $\|u_F(t)\| = 1$ for alle $t \in [c,d]$ og at $\theta_F(t) \in (0,\pi)$, hvor $c < t < d$. Elementære kurver kan nu karakteriseres på følgende måde.

- F er A-elementær, hvis og kun hvis, $\theta_F(t)$ er aftagende.
- F er B-elementær, hvis og kun hvis, $\theta_F(t)$ er voksende.

Lad afbildningen $n_F(t)$ være linien gennem $F(t)$ langs retningen $u_F(t)$. Kald $n_F(0)$ og $n_F(1)$ for *endepunktsnormaler* for $F = F[0,1]$. Opdel linien, $n_F(t)$, i en øvre- og en nedre- halvnormal, som hver især udspringer fra $F(t)$ og løber retning af henholdsvis $u_F(t)$ og $-u_F(t)$. De øvre halvnormaler overdækker et område begrænset af

$a(0)$, F, og $a(1)$, kald dette område $U(F)$. Området $U(F)$ betragtes som en lukket mængde, og det gælder derfor, at $F \in U(F)$. Udvides $a_F(0)$ og $a_F(1)$ til de mødes fremkommer keglen $C(F)$. Se figur 4.1 nedenfor.



Figur 4.1: KEGLEN $C(F)$ UDSPÆNDT AF NORMALERNE $n_F(0)$ OG $n_F(1)$

Lemma 4.1.1. *Hvis $F \in \mathcal{G}[0,1]$ er A-elementær, så er området $U(F)$ overdækket af en familie af halvnormaler $\{a_F(t) : t \in [0,1]\}$. Det vil sige, at ethvert punkt i $U(F)$ tilhører en entydig halvnormal $a_F(t)$ og enhver halvnormal er indeholdt i $U(F)$.*

Bevis. Det er givet umiddelbart fra definitionen, at $a_F(t) \subseteq U(F)$. For at vise entydigheden, altså at ethvert $p \in U(F)$ er indeholdt i en entydig halvnormal, antag, at $p = a_F(t) \cap a_F(s)$ for givne $t < s$. Idet vektoren $F(s) - F(t)$ har en positiv x-koordinat og $\theta_F(t) > \theta_F(s)$, må normalerne $n_F(t)$ og $n_F(s)$ skære hinanden under linien gennem $F(t)$ og $F(s)$. Så $p = b_F(t) \cap b_F(s)$, hvilket strider mod antagelsen, at $p \in a_F(t) \cap a_F(s)$. □

4.2 Elementære par

Lad $F \in \mathcal{G}[0,1]$ være en A-elementær kurve og lad $G \in \mathcal{G}[c,d]$ være en anden elementær kurve. Bemærk, at G kan være enten A- eller B-elementær.

Definition 4.2.1. *Kurvepar (F,G) kaldes et elementært par, hvis der gælder*

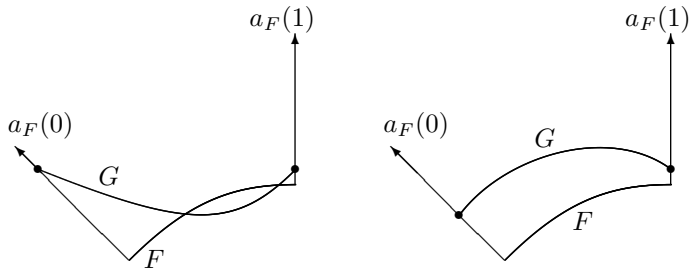
- *$G(c) \in a_F(0) \setminus \{F(0)\}$ og $G(d) \in a_F(1) \setminus \{F(1)\}$*
- *Hele G ligger indenfor keglen C(F).*

På figur 4.2 kan ses eksempler på henholdsvis AA- og AB-elementære par. (F,G) kan også udgøre BA- eller BB-elementære par, men da disse tilfælde er symmetriske behandles kun AA- og AB-par.

Lad $p \in F$ og $q \in G$. Hvis L er normallinien for F i punktet p, så kaldes ethvert punkt i $G \cap L$ en G-projektion af punktet p. Generelt kan et punkt p have ingen eller flere G-projektioner.

Lemma 4.2.2. *Lad $(F,G) \in \mathcal{G}[0,1] \times \mathcal{G}[c,d]$ være et elementært par. Hvis $G \subseteq U(F)$, så findes der en entydig kontinuert funktion $s : [0,1] \mapsto [c,d]$ således, at ethvert punkt $G(s(t))$ er en G-projektion af $F(t)$.*

Bevis. Når $G \in U(F)$ sikrer lemma 4.1.1 at $n_F(t)$ skærer G for alle $t \in [0,1]$. Så der mangler kun at blive vist, at $n_F(t)$ skærer G nøjagtig en gang. Det er oplagt, at $n_F(t)$ skærer G et ulige antal gange. Området afgrænset af G, og basesegmentet



(a) AB-elementære par (b) AA-elementære par

Figur 4.2: Elementære par

for G, er konvekst så en linie kan højst skære G to gange. Derfor skærer normalen, $n_F(t)$, linien, G, præcis en gang, nemlig i punktet $G(s(t))$. Kontinuiteten af s(t) følger fra kontinuiteten af parametriseringen af $n_F(t)$ og fra kontinuiteten af kurven G. □

Korollar 4.2.3. *Lad $(F, G) \in \mathcal{G}[0,1] \times \mathcal{G}[c,d]$ være et elementært par. Hvis $G \subseteq U(F)$, så er vinkelfunktionen $\alpha : [0,1] \mapsto (-\pi, \pi)$ givet ved*

$$\alpha(t) = \theta_F(t) - \theta_G(s(t)) \tag{4.2}$$

veldefineret og kontinuert.

Bevis. Både $\theta_F(t)$, $\theta_G(t)$ og $s(t)$ er kontinuerte funktioner, så $\alpha(t)$ er også kontinuert. At værdimængden til α-funktionen er givet ved $Vm(\alpha) = (-\pi, \pi)$ ses ved at $Vm(\theta_F) = Vm(\theta_G(s(t)) = (0, \pi)$. □

4.3 Det komplette kriterium for ikke-krydsende, tangentiel skæring

Sætning 4.3.1. *Lad $(F, G) \in \mathcal{G}[0,1] \times \mathcal{G}[c,d]$ være et elementært par. Antag, at F og G har Δ-separationsegenskaben og at diameteren af $F \cup G$ er mindre end Δ. Så findes en afbildning $\alpha : [0,1] \mapsto (-\pi, \pi)$, hvorom der gælder*

(i) Hvis $\alpha(0)\alpha(1) \leq 0$, så skærer F og G hinanden tangentielt i et entydigt punkt.

(ii) Hvis $\alpha(0)\alpha(1) > 0$, så skærer F og G ikke hinanden.

Bevis. Hvis $F \cap G$ indeholder to forskellige punkter, p og q, så gælder $\|p - q\| \leq diam(F \cup G) \leq \Delta$. Men dette modsiger antagelsen om Δ-separationsegenskaben. Det kan derfor konkluderes, at $F \cap G$ er enten tom eller indeholder præcist et punkt. Hvis $F \cap G \neq \emptyset$, så skærer kurverne F og G hinanden tangentielt i et entydigt punkt. Afbildningen α er givet i korollar 4.2.3.
(i) Antag, $\alpha(0)\alpha(1) \leq 0$. Der findes $0 < t < 1$ således, at $\alpha(t) = 0$. Parret $(p, q) = (F(t), G(s(t)))$ er derfor et antipodalpar. Hvis $p \neq q$, så medfører Δ-separationsegenskaben, at $\|p - q\| > \Delta$. Men dette modsiger antagelsen om, at $F \cup G$ har diameter mindre end Δ. Derfor gælder det $p = q$, og F og G skærer hinanden tangentielt.
(ii) Antag, at F og G skærer hinanden i et (tangentielt) punkt p og vis, at dette medfører en modstrid. Lad $p = F(t_0)$ for et $t_0 \in [0,1]$ så $\alpha(t_0) = 0$. Da $\alpha(0)\alpha(1) > 0$,

antag, uden tab af generelitet, at $\alpha(0) > 0$ og $\alpha(1) > 0$. Siden p er et tangentielt skæringspunkt, og det er antaget, at F ligger under G, må $\alpha(t_0^-) > 0$ og $\alpha(t_0^+) < 0$. Da $\alpha(t)$ er en kontinuert funktion, findes et $t_1 \in (t_0, 1)$ således, at $\alpha(t_1) = 0$ og $(F(t_1), G(s(t_1)))$ er derfor et antipodalpar. Men $F(t_1) = G(s(t_1))$ er så endnu et tangentielt skæringspunkt mellem F og G, hvilket strider mod Δ-separationsegenskaben. □

Kapitel 5

Skæringsalgoritmen

En formel (matematisk) specifikation af en skæringsalgoritme for Bézier kurver i planen kunne se ud som følgende.

Algoritme 5.0.1 YapAlgorithm(F, G)

Input: $P(F) = (p_0, p_1, \ldots, p_m)$ og $P(G) = (q_0, q_1, \ldots, q_m)$
Output: $\{r \in \mathbb{E}^2 \mid r \in F \cap G\}$

Problemet med denne specifikation er, at den ikke dækker de besværligheder involveret i håndteringen af sådanne matematiske problemer under en beregningsmodel, som ikke har eksakt aritmetik til rådighed. En oplagt løsning til dette problem er at indføre eksakt aritmetik og så løse skæringsproblemet ved hjælp af algebraisk manipulation af de involverede ligninger. Denne tilgang er ofte forbundet med ringe effektivitet i praksis, så en mere effektiv tilgang er nødvendig. De skæringsalgoritmer, som opnår effektivitet i praksis, er ofte baseret på numeriske metoder, hvilket gør det svært at opnå kravene om eksakte løsninger. Den fundamentale motivation for Yaps skæringsalgoritme er, ifølge [4, 5], at designe en eksakt skæringsalgoritme, baseret på opdeling (subdivision), af Bézier kurver og samtidig opnå effektivitet i praksis. I dette kapitel beskrives Yaps skæringsalgoritme i detaljer, hvor der gøres rede for, hvordan algoritmen opnår sine mål om eksakte løsninger. En opsummering af de generelle teknikker, som algoritmen udnytter, følger.

- **Subdivision med geometriske separationsgrænser.** Det primære algoritmiske skridt i Yaps algoritme er opdeling (subdivision), se afsnit 2.5, af Bézier kurverne. Stop kriterierne for denne iterative opdeling udgøres af de geometriske separationsgrænser Δ_i, blandt andet introduceret i afsnit 3.2. Disse grænser afhænger kun af egenskaberne for input kurverne og beregnes derfor i initialiseringsfasen, hvilket betyder, at deres værdier ikke påvirker logikken i algoritmen. Hvis der findes bedre grænser i fremtiden, kan disse umiddelbart indføres i algoritmen uden ændringer til den algoritmiske logik.

- **BigFloat tal.** En eksakt skæringsalgoritme må nødvendigvis benytte eksakt aritmetik, hvilket vil sige rational aritmetik. Alle numeriske beregninger i algoritmen benytter derfor kun regneoperationerne $(+, -, \times)$ på (binære) bigfloat tal, se afsnit 2.6. Disse regneoperationer er eksakte, hvilket for eksempel vil medføre, at værdien af følgende udtryk er eksakt, hvis $a_{ij} \in \mathbb{B}$, altså er bigfloat tal.

$$E(a_{11}, a_{12}, a_{21}, a_{22}) = \det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11} \times a_{22} - a_{12} \times a_{21}$$

Grunden til, at der ikke anvendes generelle rationale tal, er ønsket om effektivitet i praksis. Regneoperationerne på bigfloat tal kan håndteres mere effektivt end generelle rationale tal. Se sætning 2.6.5.

- **Adaptivitet.** Algoritmen er adaptiv. Det vil sige, at under de iterative processer nås separationsgrænserne, Δ_i, kun i værste tilfælde. For "pæne"input stopper den iterative process længe før, separationsgrænserne er nået. Sådan tidlige stop af den iterative process er baseret på partielle kriterier. Partielle kriterier kan enten være *accept-* eller *reject*-kriterier, som henholdsvis kan bekræfte og afkræfte skæring mellem to Bézier kurver. Selvom partielle kriterier ikke er fremhævet i dette speciale, så har de stadig indflydelse på den generelle struktur i Yaps algoritme. Det forventes, at de fleste partielle kriterier kan indføres i algoritmen uden større vanskeligheder.

Fra det ovenstående er det ikke oplagt, at det er muligt at opnå en eksakt skæringsalgoritme. Hvordan håndteres eksempelvis tilfældet, hvis input kurverne har skæringspunkter, som har irrationale koordinater. Det følgende vil forsøge at belyse disse problemer.

5.1 Den overordnede algoritme

Lad Bézier kurverne F og G være givet ved deres respektive kontrol polygoner, henholdsvis $P(F) = (p_0, p_1, \ldots, p_m)$ og $P(G) = (q_0, q_1, \ldots, q_n)$. Kald et par af kurver (F', G') for et *kandidatpar*, hvis der om kurverne gælder, at deres respektive konvekse hylstre skærer hinanden, altså $CH(F') \cap CH(G') \neq \emptyset$. Et kandidatpar (F', G') kaldes også henholdsvis *mikropar* eller *makropar* afhængig af, om $diam(CH(F') \cup CH(G')) < \Delta^*$.
Givet to Bézier kurver, F og G, så finder YapAlgorithm, se algoritme 5.1.1, alle skæringspunkter mellem F og G. Overordnet set arbejder algoritmen på tre køer Q_0, Q_1 og S, som kaldes henholdsvis makro-, mikro- og løsningskøen. Køerne indeholder under algoritmens forløb kandidatpar, som henholdsvis udgør makropar, mikropar eller løsninger (skæringspunkter). YapAlgorithm kan inddeles i tre faser, nemlig initialiseringsfasen, makrofasen og mikrofasen.
Initialiseringsfasen udgør den oplagte initialisering af køerne Q_0, Q_1 og S, samt beregning af de enkelte geometriske separationsgrænser.
Makrofasen arbejder på makropar fra makrokøen og gør følgende. Den udtager et makropar (F', G') fra makrokøen og forsøger at afgøre skæring mellem F' og G' ved hjælp af partielle kriterier. Hvis det lykkes, indsættes (F', G') i løsningskøen S. Ellers deles en af kurverne, eksempelvis F, således, at følgende par fremkommer (F'_0, G'), (F'_1, G'). Hvis (F'_i, G') udgør et kandidatpar indsættes parret i henholdsvis makro- eller mikrokøen afhængig af, om det udgør et makro- eller mikropar. Herefter udtages et nyt makropar fra makrokøen, og processen gentager sig. Dette forsætter indtil makrokøen er tom. Hvis mikrokøen ikke er tom, fortsætter Yaps algoritme med mikrofasen, som for hver mikropar (F', G') i mikrokøen gør følgende. Hvis F' indeholder et kritisk punkt, indsættes (F', G') i løsningskøen S. Ellers udføres kobling processen på (F', G').

Resten af dette kapitel beskriver de tre ovennævnte faser i detaljer. Afsnit 5.2 uddyber hvorledes de geometriske separationsgrænser beregnes, mens afsnittene 5.3 og 5.4 beskriver detaljerne for henholdsvis makrofasen og mikrofasen.

Algoritme 5.1.1 YapAlgorithm(F, G)

Input: $P(F) = (p_0, p_1, \ldots, p_m)$ og $P(G) = (q_0, q_1, \ldots, q_n)$.
Output: En kø S som indeholder kandidatpar (F_i, G_j), hvorom der gælder $F_i \subseteq F$, $G_j \subseteq G$ og $F_i \cap G_j \neq \emptyset$.

```
1: {Initialiseringsfasen}
2: Q_0 ← ∅    Q_1 ← ∅    S ← ∅
3: Δ* ← CalculateΔ
4: {Makro- og Mikrofasen}
5: Q_0 ← (F, G) {(F, G) indsættes i Q_0}
6: MacroPhase(Q_0, Q_1, S, Δ*)
7: if (Q_1 ≠ ∅) then
8:    MicroPhase(Q_1, S, Δ*)
9: end if
10: return S
```

5.2 Initialiseringsfasen

Initialiseringsfasen er givet i pseudo-kode i algoritme 5.2.1. Udover den oplagte initialisering af makro-, mikro- og løsningskøerne går initialiseringsfasen i alt sin enkelthed ud på at beregne de enkelte geometriske separationsgrænser. Specielt beregnes Δ^*, som benyttes til at afgøre, om et kandidatpar udgør et makro- eller mikropar. Δ^* er en størrelse, som udgør et minimum af nogle specifikke separationsgrænser, nemlig $\Delta_1(m, n, a, b)$, $\Delta_2(m, n, a, b)$ (afsnit 3.2), $\Delta_4(m, n, a, b, L)$ og $\Delta_6(m, n, L)$ (afsnit 5.4.1). Som det kan ses, er de enkelte grænser udtrykt ved $\Delta_i(m, n, a, b, L, l)$. Parametrene repræsenterer forskellige størrelser, som kun afhænger af input kurverne til algoritmen.

Algoritme 5.2.1 InitPhase$(F, G, Q_0, Q_1, S, \Delta^*)$

```
Q_0 ← ∅    Q_1 ← ∅    S ← ∅
a = (16^L 9^m)^m {Sætning 3.3.2}
b = (16^L 9^n)^n {Sætning 3.3.2}
Δ_1 ← Δ_1(m, n, a, b) {Sætning 3.2.5}
Δ_2 ← Δ_2(m, n, a, b) {Sætning 3.2.6}
Δ_4 ← Δ_4(m, n, a, b, L) {Korollar 5.4.4}
Δ_6 ← Δ_6(m, n, L) {Sætning 5.4.5 og lemma 5.4.2}
Δ* ← min{Δ_1, Δ_2, Δ_4, Δ_6}
```

Størrelserne m og n udgør graden af hver af input kurverne F og G. Deres værdier er enten givet eksplicit til algoritmen, eller også kan de beregnes ved at tælle kontrol punkterne til hver af input kurverne.
Værdierne L og l udtrykker den såkaldte (bit-)kvalitet af de (bigfloat) tal, som udgør koordinaterne i kontrol polygonerne for input kurverne F og G. Se afsnit 2.6. Disse værdier forventes givet til algoritmen sammen med input kurverne.
Størrelserne a og b udgør $\|\cdot\|_2$-normerne for de algebraiske kurver, som ligger bag input kurverne. Husk, at enhver Bézier kurve kan udtrykkes som en algebraisk kurve $C(x, y) = 0$. Se eventuelt afsnit 3.1. Så hvis $A(x, y) = 0$ og $B(x, y) = 0$ er de bagvedliggende algebraiske kurver til henholdsvis F og G, så er $a = \|A\|_2$ og $b = \|B\|_2$. De algebraiske kurver er ikke umiddelbart tilgængelige, da input kurverne er givet ved kontrol polygoner. Men sætning 3.3.2 giver en eksplicit øvre grænse for størrelserne a og b, der kun afhænger af værdierne m, n og L, så denne øvre grænse bruges i stedet for de specifikke normer.

5.3 Makrofasen

Makrofasen i Yaps algoritme udgør en generel skæringsalgoritme baseret på opdeling (subdivision). En sådan algoritme er baseret på en række kriterier til at finde de enkelte skæringspunkter. Disse kriterier er typisk *partielle* kriterier, hvilket skal forstås på den måde, at de enten kan bekræfte skæring (*accept* kriterier) eller afkræfte (*reject* kriterier). Et kriterium, som både er et accept- og reject-kriterium, kaldes et *komplet* kriterium, da det både kan bekræfte og afkræfte skæring.
Et velkendt reject-kriterium for Bézier kurver er *Konveks hylster* kriteriet, som benytter det faktum, at en Bézier kurve er indeholdt i sit kontrol polygon. Se afsnit 2.2. Dette faktum kan så bruges på følgende måde. Hvis det konvekse hylster for kurven F ikke skærer det konvekse hylster for kurven G, så kan kurverne nødvendigvis ikke skære hinanden.
I den nedenstående forklaring af makrofasen lægges ikke stor vægt på de individuelle partielle kriterier, da disse ikke har indvirkning på det generelle design af makrofasen. Jo mere effektive partielle kriterier desto mere effektive bliver hele Yaps algoritme. Makrofasen kan ses i pseudo-kode i algoritme 5.3.1.

Algoritme 5.3.1 MacroPhase(Q_0, Q_1, S, Δ^*)

```
1:  while Q_0 ≠ ∅ do
2:    (F', G') ← Q_0
3:    if CH(F') ∩ CH(G') ≠ ∅ then
4:      ...
5:      if AcceptCriteria_l(F', G') = true then
6:        S ← (F', G')
7:        ...
8:      else if RejectCriteria_k(F', G') = true then
9:        continue {Skip to next iteration}
10:     else if diam(CH(F') ∪ CH(G')) < Δ* then
11:       Q_1 ← (F', G')
12:     else
13:       if diam(CH(F')) > diam(CH(G')) then
14:         (F_0, F_1) ← Subdivide(F')
15:         Q_0 ← (F_0, G')
16:         Q_0 ← (F_1, G')
17:       else
18:         (G_0, G_1) ← Subdivide(G')
19:         Q_0 ← (F', G_0)
20:         Q_0 ← (F', G_1)
21:       end if
22:     end if
23:   end if
24: end while
```

Makrofasen består af en løkke, som stopper, når makrokøen er tom. Under hver iteration udtages et makropar (F', G') fra makrokøen. Hvis dette makropar udgør et kandidatpar, hvilket afgøres med konveks hylster kriteriet, så udsættes det for de enkelte partielle kriterier. Hvis det lykkedes for et accept-kriterium at bekræfter, et entydigt skæringspunkt mellem F' og G', så indsættes (F', G') i løsningskøen S, og makrofasen forsætter med næste iteration. Hvis derimod reject-kriteriet afkræfter skæring mellem F' og G', så springes umiddelbart til næste iteration. Hvis ingen af de partielle kriterier kan afkræfte eller bekræfte skæring, kontrolleres om diameteren af foreningen af de to kontrol polygoner er mindre end Δ^*. Hvis dette er tilfældet, indsættes (F', G') i mikrokøen. Ellers opdeles (subdivision) den kurve,

hvis kontrol polygon har den største diameter, og de fremkomne kurvepar indsættes i makrokøen. Herefter forsættes til næste iteration.

Konsekvenserne af dette design af makrofasen er som følger. Når makrofasen afslutter efter et endeligt antal iterationer, vil makrokøen være tom, og mikrokøen vil indeholde kandidatpar, hvorom der gælder, at $diam(CH(F') \cup CH(G')) < \Delta^*$. Løsningskøen kan også indeholde bekræftede, via accept-kriterium, skæringspunkter, som udgør en del af hele algoritmens resultat. At makrofasen stopper, hænger på det faktum, at efter hver iteration fjernes et kurvepar fra makrokøen. I de tilfælde, hvor der indsættes to kurvepar i makrokøen, har disse kurvepar mindre diameter end det kurvepar, som de erstatter. Dette vil i sidste ende føre til, at de fjernes fra makrokøen og indsættes i mikrokøen.

5.4 Mikrofasen

Mikrofasen arbejder udelukkende på mikropar og er bygget op omkring følgende sætning for mikropar.

Sætning 5.4.1. *Lad (F', G') være et mikropar. Så gælder følgende*

(i) Et mikropar kan højst have et skæringspunkt, og det skæringspunkt er isoleret og entydigt.

(ii) Kurven F' i mikropar (F', G') kan højst have et kritisk punkt, og hvis den har det, udgør det kritiske punkt et entydigt skæringspunkt mellem F' og G'.

(iii) Hvis F' ikke indeholder et kritisk punkt, kan kobling processen afgøre, om kurverne skærer hinanden.

Bevis. Per definition opfylder mikropar, at $diam(CH(F') \cup CH(G')) < \Delta^*$.
(i) Sætning 3.2.6 beskriver den mindste afstand, Δ_2, mellem to forskellige skæringspunkter mellem F' og G'. Denne grænse er større end Δ^*, hvilket fuldender beviset.
(ii) Korollar 5.4.4 beskriver den mindste afstand, Δ_4, mellem to forskellige kritiske punkter på kurven F' og samtidig gælder, at $\Delta_4 > \Delta^*$, hvilket færdiggør første del af beviset. Sætning 5.4.5 og lemma 5.4.2 beskriver tilsammen en mindste afstand, Δ_6, mellem kurven G' og et kritisk punkt på F', og da $\Delta_6 > \Delta^*$, så er beviset fuldendt.
(iii) Se afsnit 5.4.2. □

Sætning 5.4.1 giver anledning til et design af mikrofasen, som er givet i algoritme 5.4.1. Mikrofasen forløber som følgende. For hvert mikropar (F', G') kontrolleres først, om F' indeholder et kritisk punkt, hvordan denne kontrol laves i praksis, er beskrevet i afsnit 5.4.1, derefter udføres kobling processen. Kobling processen er inspireret af sætning 4.3.1, som udgør et kriterium for ikke-krydsende tangentiel skæring mellem elementære kurver. Hvordan kobling processen fungerer i praksis, kan findes i afsnit 5.4.2.

5.4.1 Kritiske punkter og ikke-elementære kurver

En general Bézier kurve er ikke nødvendigvis en elementær kurve. Her mindes om, at en elementær kurve udgør grafen for en konkav/konveks funktion af en variabel i de reelle tal. Se afsnit 4 for mere information om elementære kurver. En general Bézier kurve, $F[0, 1]$, kan dog opbrydes i endeligt mange elementære kurver. Punkterne $F(t)$, hvor denne opbrydning foretages, kaldes *kritiske punkter*. Det skal dog vise sig, at opbrydningen af en Bézier kurve i de kritiske punkter er en optimal strategi.

Algoritme 5.4.1 MicroPhase(Q_1, S, Δ^*)

```
1: while Q ≠ ∅ do
2:    (F', G') ← Q
3:    if HasCriticalPoints(F', S, Δ*) then
4:       S ← (F', G')
5:    else
6:       CouplingProcess(F', G', S, Δ*)
7:    end if
8: end while
```

Lad $F_x(t)$ og $F_y(t)$ være koordinatfunktionerne for Bézier kurven F således, at $F(t) = (F_x(t), F_y(t))$. Så er $F'(t) = (F'_x(t), F'_y(t))$ den afledte med hensyn til t. Et punkt $F(t)$, $t \in (0,1)$, på kurven er et kritisk punkt, jævnfør [8, Lemma 2.1], hvis et af de følgende forhold er opfyldt.

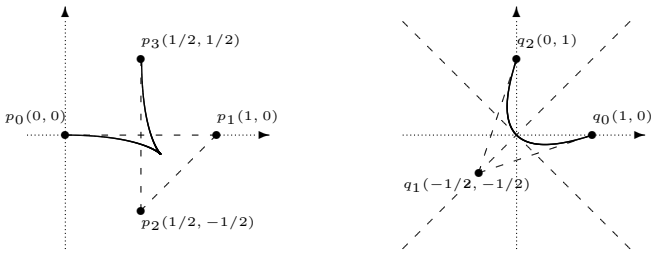
(i) $F(t)$ er et **stationært** punkt: $F'_x(t) = F'_y(t) = 0$

(ii) $F(t)$ er et **x-ekstremt** punkt: $F'_x(t) = 0, F'_x(t^-)F'_x(t^+) < 0$ og $F'_y(t) \neq 0$

(iii) $F(t)$ er et **inflexionspunkt** punkt : $H_F(t) = 0$ og $H_F(t^+)H_F(t^-) < 0$, hvor

$$H_F(t) = F'_x(t)F''_y(t) - F'_y(t)F''_x(t)$$

Figure 5.1 viser et eksempel på en kubisk Bézier kurve med et kritisk punkt, samt kurvens hodograf. Generelt gælder, at hvis hodografen for en Bézier kurve går gennem punktet $(0,0)$, så har kurven et kritisk punkt. Se afsnit 2.4 for mere information omkring hodografen for en Bézier kurve.



(a) Kubisk Bézier kurve (b) Hodograf

Figur 5.1: En kubisk Bézier kurve med tilhørende hodograf

Hvis en Bézier kurve, F, givet ved $P(F) = (p_0, \ldots, p_n)$ ikke har nogle kritiske punkter i sit indre, er F en elementær kurve. Omvendt, hvis F er en elementær kurve, har den ingen x-ekstreme eller inflexionspunkter i sit indre. Dette betyder, at en elementær kurve godt kan have stationære punkter. F.eks har kurven $P(F) = ((-1,0),(0,0),(0,0),(1,0))$ et stationært punkt i $F(1/2)$.
På grund af antagelsen, om kun at bruge eksakte bigfloat beregninger, er det ikke muligt at opbryde Bézier kurver i de kritiske punkter. Grunden er, at værdien af parameteren, $t \in (0,1)$, hvor $F(t)$ er et kritisk punkt, ikke kan garanteres at kunne repræsenteres som et bigfloat tal, så opdeling af en Bézier kurve kan ikke umiddelbart foretages i kritiske punkter. I stedet opbrydes F i punkter, hvor t kan repræsenteres som et bigfloat tal, og hvor delkurverne højst indeholder et kritisk punkt. Kald en Bézier kurve, der indeholder præcis et kritisk punkt for en *kritisk*

elementær kurve. Der findes så tre typer af kritiske elementær kurver, nemlig K-, C- og S-kurver[1], afhængig af, om det kritiske punkt er et stationært-, et x-ekstremt- eller et inflexionspunkt.

Lad F og G være to Bézier kurver med kontrol polygoner givet ved henholdsvis $P(F) = (p_0, \ldots, p_m)$ og $P(G) = (q_0, \ldots, p_n)$. Koordinaterne til punkterne p_i og q_j, for $i = 1, 2, \ldots, m$ og $j = 1, 2, \ldots, n$ er givet ved eksakte bigfloats.

Lemma 5.4.2. *Lad $F(t_0) = (F_x(t_0), F_y(t_0))$, for $t_0 \in (0, 1)$, være et kritisk punkt for Bézier kurven F.*

(i) Hvis $F(t_0)$ er et stationært eller x-extremt punkt, så har $F_x(t_0)$ og $F_y(t_0)$ grad højst $m-1$ og $\|\cdot\|_2$-norm højst $(4^L 9^m m)^m$.

(ii) Hvis $F(t_0)$ er et inflexionspunkt, så har $F(t)$ grad højst $2m-3$ og $\|\cdot\|_2$-norm højst $(2m^4 16^L 81^m)^m$.

Bevis. (i) Hvis $F(t) = (X, Y)$ er et stationært eller x-extremt punkt, så opfylder X ligningen

$$R(X) = res_t(F_x(t) - X, F'_X(t))$$

Når $F_x(t) = \sum_{i=0}^{m} a_i t^i$, så er $R(X)$ determinaten af matrix M_1 nedenfor.

$$M_1 = \begin{bmatrix} a_m & a_{m-1} & \cdots & a_1 & a_0 - X & \cdots & \\ & a_m & \cdots & a_2 & a_1 & \cdots & \\ & & \ddots & & & \cdots & \\ & & & a_m & a_{m-1} & \cdots & a_0 - X \\ m a_m & (m-1)a_{m-1} & \cdots & 2a_2 & a_1 & & \\ & m a_m & \cdots & 3a_3 & 2a_2 & \cdots & \\ & & \ddots & & & \cdots & \\ & & & m a_m & a_{m-1} & \cdots & a_1 \end{bmatrix}$$

På samme måde som i sætning 3.3.2 betragtes matricen, M_2, af $\|\cdot\|_1$-normen af hver indgang (betragtet som et polynomium) i matrix M_1. Når grænserne fra sætning 3.3.1 benyttes, fremkommer matricen nedenfor.

$$M_2 = \begin{bmatrix} 2^{L+m}\binom{m}{m} & 2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L+1}\binom{m}{1} & 2^L\binom{m}{0}+1 & \cdots & \\ & 2^{L+m}\binom{m}{m} & \cdots & 2^{L+2}\binom{m}{2} & 2^{L+1}\binom{m}{1} & \cdots & \\ & & \ddots & & & \cdots & \\ & & & 2^{L+m}\binom{m}{m} & 2^{L+m-1}\binom{m}{m-1} & \cdots & 2^L\binom{m}{0}+1 \\ m2^{L+m}\binom{m}{m} & (m-1)2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L+1}\binom{m}{1} & & & \\ & m2^{L+m}\binom{m}{m} & \cdots & 22^{L+2}\binom{m}{2} & 2^{L+1}\binom{m}{1} & \cdots & \\ & & \ddots & & & \cdots & \\ & & & m2^{L+m}\binom{m}{m} & (m-1)2^{L+m-1}\binom{m}{m-1} & \cdots & 2^{L+1}\binom{m}{1} \end{bmatrix}$$

De første $m-1$ rækker af M_2 har $\|\cdot\|_2$-norm $2^L 3^m$. De sidste m rækker af M_2 har $\|\cdot\|_2$-norm mindre eller lig $m2^L 3^m$. Bruges Hadamard-grænsen [7, afsnit 6.8] gælder følgende

$$\|R(X)\|_2 \leq (2^L 3^m)^{m-1}(m2^L 3^m)^m = 2^{(2m-1)L} 3^{2m^2-m} m^m < (4^L 9^m m)^m$$

[1] Bogstaverne "C" og "S" indikerer groft formen på kurven

(ii) Som i beviset for (i) betragtes matricen M_1', som har $2m-3$ rækker med koefficienterne til $F_x(t_0) - X$, og m rækker med koefficienterne til $F_x'(t_0)F_y''(t_0) - F_x''(t_0)F_y'(t_0)$. Så graden af $R(X) = \det(M_1')$ er $2m-3$. Størrelsen

$$\begin{aligned}\|F_x'(t_0)F_y''(t_0) - F_x''(t_0)F_y'(t_0)\|_2 &\leq \|F_x'(t_0)F_y''(t_0)\|_2 + \|F_x''(t_0)F_y'(t_0)\|_2 \\ &\leq \|F_x'(t_0)\|_2\|F_y''(t_0)\|_2 + \|F_x''(t_0)\|_2\|F_y'(t_0)\|_2 \\ &\leq 2m^4 4^L 9^m\end{aligned}$$

da $\|F_x'(t_0)\|_2 \leq m2^L3^m$, $\|F_x''(t_0)\|_2 \leq m^2 2^L 3^m$, $\|F_y'(t_0)\|_2 \leq m2^L3^m$ og $\|F_y''(t_0)\|_2 \leq m^2 2^L 3^m$. Det er nu mulig at vurdere $\|R(X)\|_2$.

$$\|R(X)\|_2 \leq (2^L 3^m)^{2m-3}(2m^4 4^L 9^m)^m < (2m^4 16^L 81^m)^m$$

□

Lemma 5.4.3. *Lad nu α, β være algebraiske tal, henholdsvis af grad højst m og n og $\|\cdot\|_2$-normer højst a, b. Hvis $m \leq n$ så gælder, jvf. [7, Lemma 6.32],*

$$|\alpha - \beta| > (2^{mn+m} a^n b^m)^{-1}$$

Bevis. Se [7, Afsnit 6.8.1]. □

Korollar 5.4.4. *Lad p, q være to forskellige kritiske punkter for Bézier kurven F. Hvis $2 \leq m \leq n$ så gælder*

$$|p.x - q.x| \geq \Delta_4 = (16^{m+2} 256^L 81^{2m} m^5)^{-m}$$

En tilsvarende grænse gælder for $|p.y - q.y|$.

Bevis. Når p og q er kritiske punkter for F, så gælder det, at $p.x$ og $q.x$ er løsninger til $F_x(t)$, $p.x$ og $q.x$ er algebraiske tal. Hvis lemma 5.4.3 anvendes med $\alpha = p.x$ og $\beta = q.x$, hvor $\deg(\alpha) = 2m-1$, $deg(\beta) = m-1$, $\|\alpha\|_2 = (2m^4 16^L 81^m)^m$ og $\|\beta\|_2 = (4^L 9^m)^m$ er grænserne fra lemma 5.4.2, så gælder (ved indsættelse i lemma 5.4.3)

$$|p.x - q.x| \geq (16^{m+2} 256^L 81^{2m} m^5)^{-m}$$

□

Den næste sætning er en generalisering af sætning 3.2.8 (og dermed af separationsgrænsen Δ_3).

Sætning 5.4.5. *Antag $q = (q.x, q.y)$, hvor $q.x$ og $q.y$ er algebraiske tal med $\|\cdot\|_2$-norm mindre end c og grad mindre end d. Lad om polynomiet $A(x,y) \in \mathbb{Z}[x,y]$ gælde, at $deg(A) = m$ og $\|A\|_2 = a$. Hvis kurven $A(x,y) = 0$ ikke udgør en cirkel med centrum i q, så er afstanden mellem q og kurven $A(x,y) = 0$ mindst*

$$\Delta_5(m, a, L, c, d) = (2^{3/2} NK)^{-D} 2^{-12m^2 d^2}$$

hvor

$$K = \max\{\sqrt{13}, 4ma, c\}, \qquad N = \binom{5+2m+2d}{5}, \qquad D = m^2 d^2 (3 + \frac{4}{m} + \frac{4}{d}).$$

Bevis. Det kan umiddelbart antages, at punktet p, på kurven $A(x,y)=0$, er det punkt på kurven, som ligger tættest på q, og lad $h=\|p-q\|$. Så opfylder q, p og h følgende system af polynomier.

$$\begin{aligned}
&\mathcal{A}_1 : A(p) = 0\\
&\mathcal{A}_2 : h^2 - \|p-q\|^2 = 0\\
&\mathcal{A}_3 : \left\langle (\frac{dA}{dy}(p), -\frac{dA}{dx}(p)), (p-q) \right\rangle = 0\\
&\mathcal{A}_4 : P(q.x) = 0\\
&\mathcal{A}_5 : Q(q.y) = 0
\end{aligned}$$

hvor $P(x), Q(x) \in \mathbb{Z}[x]$ er polynomier, af grad højst d og med $\|\cdot\|_2$-norm højst c, og hvor ligningerne $P(x)=0$ og $Q(x)=0$ har henholdsvis $q.x$ og $q.y$ som løsninger. Dette system af polynomielle ligninger er 0-dimensionalt, da $A(x,y)=0$ ikke udgør en cirkel omkring punktet q. Når $\|\mathcal{A}_1\|_2 = a$, $\|\mathcal{A}_2\|_2 \leq \sqrt{13}$, $\|\mathcal{A}_3\|_2 \leq 4ma$, $\|\mathcal{A}_4\|_2 \leq c$ og $\|\mathcal{A}_5\|_2 \leq c$ indsættes i sætning 3.2.3, er beviset færdigt. □

Hvis grænserne fra lemma 5.4.2 indsættes i sætning 5.4.5, opnås en ny separationsgrænse, nemlig $\Delta_6 = \Delta(m,n,L)$ som udgør minimumafstanden mellem en kurve G og et kritisk punkt p på kurven F, når p ikke også ligger på kurven G. Sættes $\Delta_* = \min\{\Delta_1, \Delta_2, \Delta_4, \Delta_6\}$, hvor Δ_1 og Δ_2 er fra afsnit 3.2, er det nu mulig at afgøre, om kurve, F, hvis diameter er mindre end Δ_*, udgør en kritisk elementær kurve.

Lad i det følgende F være en Bézier kurve givet ved kontrol polygon $P(F) = (p_0, \ldots, p_m)$. Kurven kan så, jævnfør 2.2, udtrykkes

$$F(t) = \sum_{i=0}^{m} p_i B_i^m(t)$$

hvor $B_i^m = \binom{m}{i} t^i (1-t)^{m-i}$. Den afledte med hensyn til t kan så, jævnfør 2.4, gives ved

$$\begin{aligned}
F'(t) &= m \sum_{j=0}^{m-1} \left[p_{j+1} - p_j\right] B_j^{m-1}(t)\\
&= m \sum_{j=0}^{m-1} \Delta p_j B_j^{m-1}(t)
\end{aligned}$$

hvor $\Delta p_j = p_{j+1} - p_j$ er den såkaldte *forward difference operator*. Så kan hodografen, $m\Delta P(F) = (m\Delta p_0, \ldots, m\Delta p_{m-1})$, opfattes som et kontrol polygon for kurven $F'(t)$.

Stationære punkter

Sætning 5.4.6. *Lad F være en Bézier kurve givet ved kontrol polygon $P(F) = (p_0, \ldots, p_m)$. Antag yderligere, at diameteren af F er mindre end $\Delta_3(m-1, ma, 2, 2)/m$. Så indeholder F et stationært punkt hvis, og kun hvis, det konvekse hylster af $\Delta P(F)$ indeholder punktet $(0,0)$.*

Bevis. Lad $F(t_0)$ være et stationært punkt på Bézier kurven F. Så gælder det, at $F'(t_0) = (0,0)$, ifølge definitionen af stationære punkter, og da $m\Delta P(F)$ er kontrol polygonen til kurven $F'(t)$, så indeholder det konvekse hylster af $m\Delta P(F)$ blandt

andet punktet $(0,0)$.
Antag nu, at Bézier kurven F ikke indeholder nogle stationære punkter. F' har grad $m-1$ (se ovenfor) og $\|F'\|_2 \leq ma$. Ifølge sætning 3.2.8 er afstanden fra $(0,0)$ til kurven F' mindst $\Delta_3(m-1, ma, 2, 2)$, da $(0,0)$ er et 2-bit flydende kommatal. Så afstanden fra $(0,0)$ til kurven F'/m er mindst $\Delta_3(m-1, ma, 2, 2)/m$. På den anden side, er diameteren af $\Delta P(F)$ er højst diameteren af F, det vil sige, mindre end $\Delta_3(m-1, ma, 2, 2)/m$. Dette betyder, at punktet $(0,0)$ ikke kan ligge indenfor det konvekse hylster af $\Delta P(F)$. □

Bemærk, at separationsgrænsen $\Delta_3(m-1, ma, 2, 2)$ er betydeligt større end Δ^*. Så for en kurve, F, med diameter mindre end Δ^* kan sætning 5.4.6 umiddelbart bruges til at afgøre, om F indeholder et stationært punkt ved at beregne det konvekse hylster for $\Delta P(F)$ og tjekke, om dette hylster indeholder punktet $(0,0)$.

X-ekstreme punkter

Sætning 5.4.7. *Lad F være en Bézier kurve givet ved kontrol polygon $P(F) = (p_0, \ldots, p_m)$. Antag, at diameteren af F er mindre end Δ^*. Så har F et x-ekstremt punkt hvis, og kun hvis, $(\Delta p_0).x(\Delta p_{m-1}).x \leq 0$ og $(\Delta p_0).y(\Delta p_{m-1}).y > 0$*

Bevis. Hvis $(\Delta p_0).x(\Delta p_{m-1}).x \leq 0$ og $(\Delta p_0).y(\Delta p_{m-1}).y > 0$ gælder så indeholder F præcist et x-ekstremt punkt.
Antag nu, at $F(t_0)$ er et x-ekstremt punkt, så ligger både $F[0,t_0]$ og $F[t_0,1]$ på samme side af den lodrette linie gennem punktet $F(t_0)$. Da diameteren af F er mindre end Δ^*, findes der ikke andre kritiske punkter på F, så delkurverne $F[0,t_0]$ og $F[t_0,1]$ er elementære kurver. Det er nu klart, at den første og den sidste kant, henholdsvis Δp_0 og Δp_{m-1}, opfylder $(\Delta p_0).x(\Delta p_{m-1}).x \leq 0$ og $(\Delta p_0).y(\Delta p_{m-1}).y > 0$. □

Inflexionspunkter

Lad $p, q, r \in \mathbb{R}^2$ være punkter i planen, hvor $p = (p.x, p.y)$, $q = (q.x, q.y)$ og $r = (r.x, r.y)$. Så defineres

$$\det(p,q,r) = \det\left(\begin{bmatrix} p.x & p.y & 1 \\ q.x & q.y & 1 \\ r.x & r.y & 1 \end{bmatrix}\right)$$

og

$$\det(p,q) = \det\left(\begin{bmatrix} p.x & p.y \\ q.x & q.y \end{bmatrix}\right)$$

Sætning 5.4.8. *Lad F være en Bézier kurve givet ved kontrol polygon $P(F) = (p_0, \ldots, p_m)$. Antag, at diameteren af F er mindre end Δ^*. Så har F et inflexionspunkt hvis, og kun hvis,* $\det(p_0,p_1,p_2)\det(p_{m-2},p_{m-1},p_m) < 0$.

Bevis. Først gøres rede for sammenhængen mellem $det(p_0,p_1,p_2)$ og inflexionspunkter. Lad $H(t) = F'_x(t)F''_y(t) - F''_x(t)F'_y(t)$ og bemærk, $F'(0) = p_1 - p_0$ og $F''(0) = p_2 - 2p_1 + p_0$. Så

$$\begin{aligned} H(0) &= \det(p_1 - p_0, p_2 - 2p_1 + p_0) \\ &= \det(p_1 - p_0, p_2 - p_1) \\ &= -\det(p_0 - p_1, p_2 - p_1) \\ &= \det(p_1, p_0, p_2) \\ &= -\det(p_0, p_1, p_2) \end{aligned}$$

og tilsvarende gælder

$$H(1) = -det(p_{m-2}, p_{m-1}, p_m)$$

Antag nu, at $\det(p_0, p_1, p_2)\det(p_{m-2}, p_{m-1}, p_m) < 0$, så findes der et $t_0 \in [0, 1]$ således, at $H(t_0) = 0$ (ifølge skæringssætningen for kontinuerte reelle funktioner). Punktet $F(t_0)$ er så et inflexionspunkt.
Antag, at $F(t_0)$ er et inflexionspunkt, så gælder $H(t_0) = 0$ og $H(t_0^-)H(t_0^+) < 0$. Da diameteren for F er antaget mindre end Δ^*, findes der ikke andre inflexionspunkter, så der gælder $H(0)H(1) < 0 \Leftrightarrow \det(p_0, p_1, p_2)\det(p_{m-2}, p_{m-1}, p_m) < 0$. □

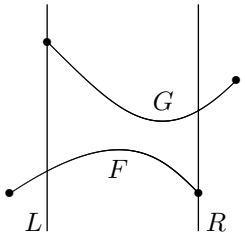
Ved hjælp af de tre ovenstående sætninger er det nu relativt nemt at afgøre, om en Bézier kurve indeholder kritiske punkter. Bemærk, hvis de ovenstående tre kriterier for, om en kurve indeholder kritiske punkter, implementeres ved hjælp af eksakte bigfloat tal, vil der ingen usikkerhed være i de fundne resultater. Hvis kriterierne derimod implementeres med almindelig flydende kommatal aritmetik, vil man ikke kunne garantere et korrekt resultat. Dette er en af nøglerne til, hvorfor Yaps algoritme er komplet, hvilket vil sige, at den finder alle skæringspunkter mellem to givne kurver.

5.4.2 Kobling processen

Lad F og G være elementære kurver, som opfylder Δ-separationsegenskaben og samtidig opfylder, at $diam(F \cup G) < \Delta$, hvilket medfører, at kurverne F og G højst har et skæringspunkt, se definition 3.2.9. Kurverne er givet ved henholdsvis $F[a, b]$ og $G[c, d]$, hvor $a, b, c, d \in \mathbb{B}$ er bigfloat værdier. Kobling processens mål er at afgøre, om kurverne F og G skærer hinanden. I den forbindelse er følgende definition på sin plads.

Definition 5.4.9. *Lad $S : \mathcal{B} \to P_L(\mathbb{R}^2)$ være afbildningen, som tager en elementær kurve $F[a, b]$ og returnerer en lukket delmængde af $\mathbb{R}^2$, begrænset af to lodrette linier, gennem henholdsvis $F(a)$ og $F(b)$. $\mathcal{B}$ er her mængden af elementære (Bézier) kurver, og $P_L(\mathbb{R}^2)$ er mængden af alle lukkede delmængder af $\mathbb{R}^2$.*

Det er oplagt fra definition 5.4.9, at hvis kurverne skærer hinanden, må det foregå indenfor mængden $S(F) \cap S(G)$. Lad i det følgende L og R betegne de lodrette linier, henholdsvis $x - c_L = 0$ og $x - c_R = 0$, hvor $c_L = \max\{F(a).x, G(c).x\}$ og $c_R = \min\{F(b).x, G(d).x\}$. Bemærk, at sådan som L og R er defineret, må der gælde, at L indeholder mindst et at punkterne $F(a)$ eller $G(c)$, og R indeholder mindst et af punkterne $F(b)$ eller $G(d)$. Se figur 5.2.



Figur 5.2: Elementær kurverne F og G, samt linierne L og R

Kobling processen er opdelt i fire skridt, hvoraf de første tre skridt håndterer de nemme tilfælde, henholdsvis ingen skæring, skæring i endepunkt og transversal skæring. Det fjerde og sidste skridt udføres kun, hvis de tidligere skridt ikke har afsluttet processen allerede, og består i at afgøre om kurverne F og G skærer hinanden i et tangentielt, ikke-krydsende, skæringspunkt. Det følgende er en detaljeret beskrivelse af kobling processens forløb for et givet kurvepar.

1. **Ingen skæring**
   Første skridt i kobling processen udgør en undersøgelse af, om $S(F) \cap S(G) = \emptyset$. Dette check kan reduceres til LineIntersection algoritmen i afsnit 5.4.2 på følgende måde. Der kontrolleres for, om de to lodrette linier, gennem endepunkterne til F, skærer kurven G, og om de to lodrette linier, gennem endepunkterne for G, skærer kurven F. Dette udgør fire anvendelser af LineIntersection algoritmen, som kan udføres i vilkårlig rækkefølge. Hvis der findes mindst to skæringer, så er $S(F) \cap S(G) \neq \emptyset$. Ellers gælder $S(F) \cap S(G) = \emptyset$. Hvis $S(F) \cap S(G) = \emptyset$, så skærer kurverne F og G oplagt ikke hinanden, og kobling processen kan afsluttes for dette kurvepar.

2. **Skæring i endepunkt**
   Antag, at $S(F) \cap S(G) \neq \emptyset$. Det næste skridt i kobling processen består i at afgøre, om kurverne skærer hinanden i et af endepunkterne. Bemærk, at der kan kun forekomme et skæringspunkt, når $diam(F \cup G) < \Delta$). Måden, hvorpå det undersøges, om kurverne skærer hinanden i et af endepunkterne, er ved hjælp af BeneathBeyond algoritmen fra afsnit 5.4.4. Algoritmen tager tre argumenter, nemlig en kurve K, og to punkter q og v, og returnerer enten BEYOND, BENEATH, eller ON afhængig af, om punktet q ligger over, under eller på kurven K i retning af punktet v. F.eks vil BeneathBeyond algoritmen, med argumenterne $K = F$, $q = G(c)$ og $v = G(c) + (0, 1)$, returnerer BENEATH for det kurvepar på figur 5.2. Der er behov for to anvendelser af BeneathBeyond algoritmen for at afklare, om F og G skærer hinanden i et af endepunkterne på enten L og R. Hvis et sådan skæringspunkt findes, afsluttes kobling processen, og kurvepar (F, G) udgør et skæringspunkt.

3. **Transversal skæring**
   Antag, at $S(F) \cap S(G) \neq \emptyset$, og at der ikke findes nogle skæringspunkter i endepunkterne for kurverne F og G. Det forrige skridt i kobling processen afgjorde om kurverne F og G lå under, over eller på hinanden langs linierne L og R ved hjælp af BeneathBeyond algoritmen. Hvis kurvene F og G ligger modsat hinanden på henholdsvis L og R, må kurverne nødvendigvis krydse hinanden i et transversalt skæringspunkt. For eksempel, hvis F ligger over G på linien L, og F ligger under G på linien R. Se figur 5.3(b). Hvis kurverne krydser hinanden, kan kobling processen afsluttes, og kurvepar (F, G) udgør et transversalt skæringspunkt.

4. **Tangentiel, ikke-krydsende, skæring**
   Hvis kobling processen når til dette skridt, betyder det, at $S(F) \cap S(G) \neq \emptyset$, og at der ikke findes transversale skæringspunkter eller skæringspunkter i endepunkterne langs de lodrette linier L og R. Så den eneste mulighed for skæring mellem kurverne F og G er et tangentielt, ikke-krydsende, skæringspunkt. En konsekvens af dette er, at F enten ligger over eller under G indenfor $S(F) \cap S(G)$. Hvis dette ikke var tilfældet ville kobling processen have været afsluttet i forrige skridt. I det følgende gøres rede for, hvorledes det undersøges, om der findes et tangentielt, ikke-krydsende, skæringspunkt mellem F og G.

   For at afgøre, om kurverne F og G skærer hinanden tangentielt, for tilfælde (i)-(iii) ville det optimale være, at kunne anvende sætning 4.3.1. Dette er dog ikke umiddelbart muligt, da det ikke kan garanteres, at kurverne udgør et elementært par. Det er dog, som det vil fremgå nedenfor, muligt at opbryde kurverne i såkaldte *halvpar*, og så benytte en tilpasset version af sætning 4.3.1 til at afgøre om kurvene skærer hinanden.

Definition 5.4.10. *Lad $F[a,b]$ og $G[c,d]$ være elementære (Bézier) kurver. Så kaldes $(F[a,b], G[c,d])$ for et halvpar, hvis der gælder en af følgende betingelser.*

(i) $G(c) \in a_F(a)$ *og* $G(d) \notin a_F(b)$

(ii) $G(c) \notin a_F(a)$ *og* $G(d) \in a_F(b)$

Et halvpar kan yderligere betegnes venstre-halvpar (højre-halvpar) afhængig af, om henholdsvis (i) eller (ii) er opfyldt.

Bemærk, at hvis to elementære kurver udgør et halvpar, opfylder de halvdelen af betingelserne for at være et elementært par, jævnfør definition 4.2.1.

Sætning 5.4.11. *Antag, at $(F[a,b], G[c,d])$ er et venstre-halvpar, hvor G ligger over F. Så er $G(c) \in a_F(a)$ og $a_F(b)$ skærer ikke kurven G. Specielt gælder, at $G(d)$ ligger i $U(F)$, altså over F. Lad $b_G(d)$ betegne den nedre halvnormal i punktet $G(d)$, og definer desuden $\theta_G(d)$ til at være vinklen mellem den øvre halvnormal $a_G(d)$, og den positive x-akse. Der findes så tre tilfælde.*

(i) $b_G(d)$ *skærer* $a_F(a)$

(ii) $b_G(d)$ *skærer* $a_F(b)$

(iii) $b_G(d)$ *skærer* F *i* $F(t_0)$. *Lad* $s = sign\{\theta_F(t_0) - \theta_G(d)\} \in \{-1, 0, +1\}$.

Vælg $t_1 \in (a,b)$ således, at $a_F(t_1)$ skærer $G(d)$. Så gælder

$$\alpha_{F,G}(t_1) \begin{cases} > 0 & \text{hvis } (i) \text{ eller } (iii)_s = +1 \\ < 0 & \text{hvis } (ii) \text{ eller } (iii)_s = -1 \\ = 0 & \text{hvis } (iii)_s = 0 \end{cases}$$

En tilsvarende sætning gælder for højre-halvpar.

Bevis. For tilfælde (i) og (ii) gælder, at når t bevæger sig monotont fra $t = a$ til $t = t_1$, vil $a_F(t)$ skærer $b_G(d)$ i et entydigt punkt, som vil nærme sig $G(d)$. Vinklen mellem $a_F(t)$ og $b_G(d)$ er defineret $\alpha_{F,G}(t) = \theta_F(t) - \theta_G(d)$, og $\alpha_{F,G}(t)$ skifter ikke fortegn når $t \in (a, t_1)$. Så der gælder henholdsvis, at $\alpha_{F,G}(a) > 0 \Rightarrow \alpha_{F,G}(t_1) > 0$, og $\alpha_{F,G}(a) < 0 \Rightarrow \alpha_{F,G}(t_1) < 0$ for tilfældene (i) og (ii).
Beviserne for tilfælde $(iii)_{s=+1}$ og $(ii)_{s=-1}$ forløber på samme måde, som beviserne for henholdsvis (i) og (ii). For tilfældet $(iii)_{s=0}$ gælder $s = 0$, og $\alpha_{F,G}(t_0) = \alpha_{F,G}(t_1) = 0$ □

For kurverne $F[a,b]$ og $G[a,b]$, som udgør et venstre-halvpar, kan sætning 5.4.11 nu anvendes på er som følgende. Først afgøres om den nedre halvnormal, $b_G(d)$, skærer F, $a_F(a)$ eller $a_F(b)$. Hvis $b_G(d)$ skærer F, så beregnes fortegnet på udtrykket $\theta_F(t) - \theta_G(d)$, ved hjælp af AlphaSign algoritmen fra afsnit 5.4.4. Nu kan sætning 5.4.11 anvendes til at afgøre fortegnet på $\alpha_{F,G}(t_0)$. Fortegnet af $\alpha_{F,G}(a)$ beregnes også, og herefter er det nu muligt at afgøre, om kurverne skærer hinanden. Hvis $\alpha_{F,G}(a)\alpha_{F,G}(t_0) \leq 0$ skærer F og G hinanden tangentielt uden at krydse hinanden, og kobling processen er færdig.

Der mangler nu kun at blive gjort rede for, hvorledes kurvepar bestående af elementære kurver opbrydes til halvpar, som sætning 5.4.11 kan anvendes på. Antag i det følgende, at $F[a,b]$ ligger under $G[c,d]$ (ombyt eventuelt navnene på F og G, således at F ligger under G), så findes der følgende fire tilfælde.

(i) F er A-elementær og G er A-elementær

(ii) F er A-elementær og G er B-elementær

(iii) F er B-elementær og G er A-elementær

(iv) F er B-elementær og G er B-elementær

For tilfælde (iii) er det nemt at se, at kurverne ikke skærer hinanden. Så hvis F er B-elementær og ligger under G som er A-elementær, så kan kobling processen afsluttes. Se figur 5.3(d). De tre andre tilfælde er mere omfattende og behandles nedenfor. Der gøres dog kun rede for tilfældene (i) og (ii), da tilfælde (iv) behandles på samme måde som (i).

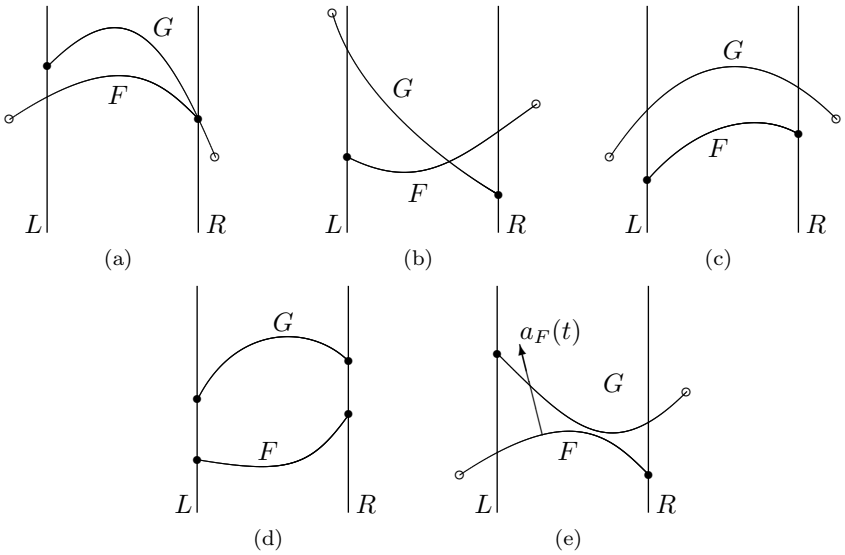
Lad i det følgende $F[a,b]$ og $G[c,d]$ være to elementære Bézier kurver, hvorom der gælder, at F ligger under G. Så findes der et $t \in (a,b)$ således, at den øvre halvnormal, $a_F(t)$, skærer kurven G. Bemærk, at $a_F(t)$ kan i nogle tilfælde skære G i to punkter. Det t, som der søges efter, skal kunne repræsenteres ved et bigfloat tal, hvilket ikke udgør noget problem, da en sådan bigfloat kan findes ved *binær søgning* i parameter intervallet, (a,b) for F.

(i) Da F og G begge er antaget at være A-elementære, gælder det, at den øvre halvnormal, $a_F(t)$, skærer kurven G i et entydigt punkt $G(s)$ for $s \in (c,d)$. Så udgør $(F[a,t], G[c,s])$ og $F([t,b], G[s,d])$ to halvpar, hvor det første er et højre-halvpar og det sidste er en venstre-halvpar. Disse halvpar kan nu testes for skæring ved hjælp af sætning 5.4.11.

$(ii)_a$ F er A-elementær, G er B-elementær og den øvre halvnormal, $a_F(t)$, skærer G i et punkt $G(s)$. Det er nok at betragte det ene af de følgende to halvpar, $(F[a,t], G[c,s])$ eller $(F[t,b], G[s,d])$. Hvilket af de to halvpar, som testes for skæring med sætning 5.4.11, afgøres af vinklen, $\theta_F(t) - \theta_G(s)$, mellem $a_F(t)$ og $a_G(s)$. Hvis $\theta_F(t) - \theta_G(s) \leq 0$ så betragtes $(F[a,t], G[c,s])$, ellers betragtes $(F[t,b], G[s,d])$. Værdien $s \in (c,d)$ ovenfor er ikke nødvendigvis en bigfloat, så algoritmen AlphaSign, afsnit 5.4.4, benyttes til at afgøre fortegnet på vinkel, $\theta_F(t) - \theta_G(s)$.

$(ii)_b$ F er A-elementær, G er B-elementær og den øvre halvnormal, $a_F(t)$, skærer G i to punkter $G(s)$ og $G(s')$ for $c \leq s \leq s' \leq d$.
Hvis $a_F(t)$ peger mod venstre (i forhold til den lodrette retning), så betragtes et af følgende to halvpar, $(F[a,t], G[s,s'])$ eller $(F[t,b], G[s',d])$. Hvilket halvpar som testes med sætning 5.4.11, afhænger af vinklen mellem $\theta_F(t) - \theta_G(s')$, som beregnes med AlphaSign algoritmen. Hvis $\theta_F(t) - \theta_G(s') \leq 0$ betragtes $(F[a,t], G[s,s'])$ ellers betragtes $(F[t,b], G[s',d])$.
Hvis $a_F(t)$ peger mod højre (i forhold til den lodrette retning), så betragtes et af følgende to halvpar, $(F[a,t], G[c,s])$ eller $(F[t,b], G[s,s'])$. Hvilket halvpar, som testes med sætning 5.4.11, afhænger af vinklen mellem $\theta_F(t) - \theta_G(s)$, som beregnes med AlphaSign algoritmen. Hvis $\theta_F(t) - \theta_G(s) \leq 0$ betragtes $(F[a,t], G[c,s])$ ellers betragtes $(F[t,b], G[s,s'])$.

For at opsummerer kobling processen for et kurvepar bestående af elementære kurver $F[a,b]$ og $G[c,d]$. Først kontrolleres om F og G ikke skærer hinanden ved at tjekke, om kurvernes projektion på x-aksen, overlapper. Hvis dette ikke er tilfældet afsluttes kobling processen, og F og G skærer ikke hinanden.
Derefter undersøges kurverne F og G for henholdsvis skæringspunkter i endepunkterne og transversale skæringspunkter. Bemærk, at der kan højst være et skæringspunkt,



Figur 5.3: (a) Skæring i endepunkt, (b) F og G krydser, (c)-(e) F ligger under G.

når $diam(F \cup G) < \Delta$. Hvis der findes et skæringspunkt, afsluttes kobling processen. Til sidst kontrolleres for, om der findes et tangentielt skæringspunkt. Denne del af kobling processen består i at opbryde kurveparret (F, G) i et eller flere halvpar og så bruge sætning 5.4.11 til at afgøre, om et af halvparrene skærer hinanden, hvilket indikerer, at F og G skærer hinanden. Dette konkluderer kobling processen.

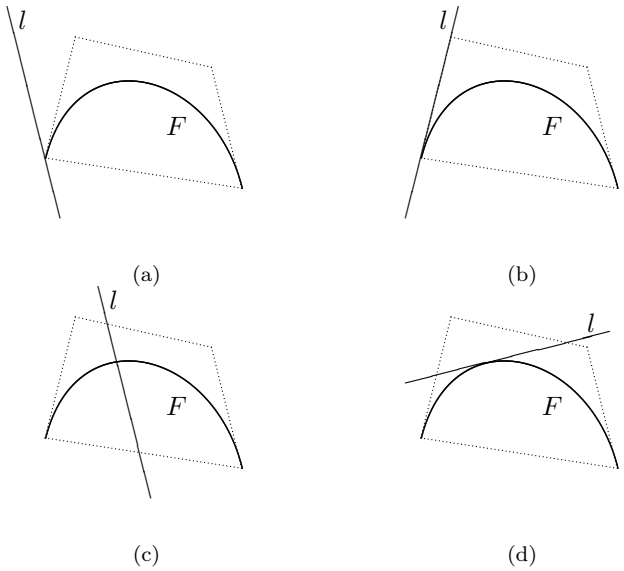
5.4.3 Skæring mellem Bézier kurver og ret linie

Dette afsnit beskriver en komplet skæringsalgoritme mellem rette linier og elementære Bézier kurver. Algoritmen indgår blandt andet i kobling processen, beskrevet i afsnit 5.4.2. Lad $F[a, b]$ være en elementær Bézier kurve givet ved kontrol polygon $P(F) = (p_0, \ldots, p_m)$, $m \geq 2$, og lad l være en ret linie. Kandidatpar (F, l) kaldes for en repræsentant for et skæringspunkt, hvis F og l skærer hinanden. Bemærk, at algoritmen beregner ikke de eksakte skæringspunkter, men derimod returnerer par (F_i, l), hvor F_i er en delkurve af F, som implicit repræsenterer de eksakte skæringspunkter. Der findes tre type af repræsentanter for skæringspunkter.

- (Eksakte) l skærer F i præcist et endepunkt, men skærer ikke det indre af det konvekse hylster til F. Se figur 5.4(a) og 5.4(b).
- (Transversale) l skærer det indre af basesegmentet for F, dvs. l skærer liniestykket mellem p_0 og p_m (men ikke punkterne selv). Se figur 5.4(c).
- (Tangentielle) l skærer ikke det indre af basesegmentet for F, men skærer det indre af det konvekse hylster til F. Se figur 5.4(d).

LineIntersection algoritmen

Før algoritmen beskrives, indføres lidt notation. Hvis $A \in \mathbb{R}^2$ er en lukket delmængde af $\mathbb{R}^2$, så udgør $Int(A)$ det indre af denne mængde. Det vil sige, at $A \setminus Int(A)$ udgør randen af mængden A. Nedenfor benyttes denne notation på følgende måde. Lad $CH(F)$ betegne det konvekse hylster for kurven F, betragtet som en lukket mængde, så udgør $Int(CHF(F))$ de punkter indeholdt i det konvekse polygon, som netop ikke ligger på randen. Tilsvarende, hvis $bs_F(t) = (1-t)p_0 + tp_m$, $t \in [0, 1]$



Figur 5.4: Repræsentanter for skæringspunkter

udgør basesegmentet for F, så er mængden $bs_F([0,1])$ en lukket mængde i $\mathbb{R}^2$, og $Int(bs_F([0,1]))$ bliver så basesegmentet fraregnet endepunkterne p_0 og p_m.

LineIntersection algoritmen kan ses i pseudo-kode, i algoritme 5.4.2, og består stort set af en række test, som enten udelukker eller bekræfter skæring. Rækkefølgen af de individuelle test er vigtig.

1. (Linie 1-3) Først kontrolleres for, om linien l skærer det konvekse hylster til kurven F. Hvis dette ikke er tilfældet, kan det konkluderes, at der ikke findes skæringspunkter, og algoritmen kan afsluttes.

2. (Linie 4-11) Hvis l skærer et, eller begge, af endepunkterne for F, findes der to muligheder. Hvis l er parallel med basesegmentet, $bs_F([0,1])$, findes der to skæringspunkter mellem F og l, som repræsenteres af henholdsvis (F_0, l) og (F_1, l), hvor F_0, F_1 udgør en opdeling af F.

3. (12-14) Hvis l skærer det indre af basesegmentet for F (det vil sige ikke i p_0 og p_m), så vil der være et transversalt skæringspunkt mellem F og l.

4. (15-17) Hvis l skærer det indre af det konvekse hylster til F, og der gælder, at diameteren af dette konvekse hylster er mindre end Δ, så har F op l et tangentielt skæringspunkt. Dette skyldes, at der findes punkter på henholdsvis kurven og på linien, som udgør et (F, l)-antipodalpar, se figur 5.4(d), men da diameteren af det konvekse hylster til F er antaget mindre end Δ vil disse punkter være ens, og dermed udgøre det et tangentielt skæringspunkt.

5. (18) Kurven F opdeles i to delkurver, nemlig F_0 og F_1.

6. (19-24) Hvis l skærer F, i punktet $F(1/2)$, så kan l kun skære det konvekse hylster for enten F_0 eller F_1. Ellers ville l også skære basesegmentet for F, og det er jo udelukket, siden algoritmen er nået til dette punkt. Så antag, at F_i, $i \in \{0,1\}$ udgør en delkurve af F, hvis det konvekse hylster skærer l. Så findes skæringspunkterne mellem l og F_i ved at kalde LineIntersection(F_i, l) rekursivt.

Algoritme 5.4.2 LineIntersection(F, l)

Input: Elementær Bézier kurve F og en ret linie l.
Output: En liste af repræsentanter for skæringspunkter (se ovenfor)

1: **if** $l \cap CH(F) = \emptyset$ **then**
2: **return** $\emptyset$
3: **end if**
4: **if** $p_0 \in l \vee p_m \in l$ **then**
5: **if** $l \parallel bs_F([0,1])$ **then**
6: $(F_0, F_1) \leftarrow$ Subdivide$(F, 1/2)$
7: **return** $[(F_0, l), (F_1, l)]$ {Eksakt skæring i begge endepunkter}
8: **else**
9: **return** $[(F, l)]$ {Eksakt skæring i endepunkt}
10: **end if**
11: **end if**
12: **if** $l \cap Int(bs_F([0,1])) \neq \emptyset$ **then**
13: **return** $[(F, l)]$ {Transversal skæring}
14: **end if**
15: **if** $l \cap Int(CH(F)) \neq \emptyset \wedge diam(F) < \Delta$ **then**
16: **return** $[(F, l)]$ {Tangential skæring}
17: **end if**
18: $(F_0, F_1) \leftarrow$ Subdivide$(F, 1/2)$
19: **if** $F(1/2) \in l$ **then**
20: **if** $l \cap Int(CH(F_0))$ **then**
21: **return** LineIntersect(F_0, l)
22: **else**
23: **return** LineIntersect(F_1, l)
24: **end if**
25: **else**
26: $L_1 \leftarrow$ LineIntersect(F_0, l)
27: $L_2 \leftarrow$ LineIntersect(F_1, l)
28: **return** $L_1 + L_2$
29: **end if**

7. (25-29) Hvis algoritmen er nået hertil, betyder det, at l skæret det konvekse hylster til F, men skærer ikke basesegmentet for F eller punktet $F(1/2)$. Så derfor kaldes LineIntersection på hver af delkurverne F_0 og F_1.

Forfinelse af en repræsentant for et skæringspunkt

Lad i dette afsnit $F[a,b]$ være en elementær Bézier kurve, og lad l være en ret linie, som skærer F i et entydigt punkt, p^*. Det vil sige, at (F,l) udgør en repræsentant for skæringspunktet p^*. Da der er tale om et entydigt skæringspunkt, gælder der, at hvis $F[a,b]$ deles i punktet $F(a+b)/2)$, vil kun den ene af delkurverne F_0 og F_1 skære linien l. Lad nu F_0 være den delkurve, som skærer linien l, så udgør (F_0,l) også en repræsentant for skæringspunktet p^*, og denne repræsentant betragtes som forfinet i forhold til den originale (F,l). De ovenstående overvejelser manifestere sig i pseudo-kode i algoritme 5.4.3.

Algoritme 5.4.3 RefineRep(F,l)

Input: En repræsentant for skæringspunktet $p^* = Point[F,l,a,b]$
Output: En forfinet repræsentant $(F',l)(=p^*)$

```
1: (F_0, F_1) ← Subdivide(F, (a + b)/2)
2: if l ∩ F_0 ≠ ∅ then
3:    return (F_0, l)
4: else
5:    return (F_1, l)
6: end if
```

BeneathBeyond algoritmen

I dette afsnit beskrives algoritmen BeneathBeyond, der kan afgøre, om et givet punkt ligger under, over eller på en elementær Bézier kurve (relativ til en angivet retning). Algoritmen udgør en anvendelse af ovenstående LineIntersection algoritme og anvendes i kobling processen, beskrevet i afsnit 5.4.2.
Lad i det følgende $q, v \in \mathbb{R}^2$ være punkter i planen, så er $R_{(q,v)}(t) = q + t(v-q)$ den stråle (ray på engelsk) som udspringer fra punktet q og peger i retning af punktet v. For elementær kurven F findes nu tre muligheder for q placering i forhold til F (relativ retningen $v-q$), nemlig

- (ON) $q \in F \Leftrightarrow R_{(q,v)}(t)$ skærer F for $t = 0$
- (BEYOND) $R_{(q,v)}(t)$ skærer F for $t > 0$.
- (BENEATH) $R_{(q,v)}(t)$ skærer F for $t < 0$.

Algoritmen virker som følgende. LineIntersection algoritmen bruges til at afgøre, om kurven $F[a,b]$ skærer den rette linie $l(u) = q + (v-q)t$. Hvis dette ikke er tilfældet, så afsluttes algoritmen med resultatet BENEATH. Ellers forsætter algoritmen under antagelse af, at F og l skærer hinanden. Nu kontrolleres, om punktet q ligger i det konvekse hylster for F. Hvis dette ikke er tilfældet, må q position i forhold til basesegmentet for F stemme overens med positionen i forhold til hele kurven. Grunden til dette er, at hvis F og l skæret hinanden, og punktet q ligger eksempelvis under basesegmentet for F, og tilmed ikke ligger i det konvekse hylster for F, så må q være under F. Et tilsvarende argument gælder, hvis q ligger over basesegmentet.
Hvis q derimod ligger indenfor det konvekse hylster for F, så forfines (F,l) med algoritme RefineRep, indtil en af følgende ting er opfyldt. Enten er q ikke længere

Algoritme 5.4.4 BeneathBeyond(F, q, v)

Input: Elementær Bézier kurve $F[a,b]$ og to punkter $q, v \in \mathbb{R}^2$
Output: Enten ON, BENEATH eller BEYOND

```
1: loop
2:   if LineIntersect(F, l(u) = q + u(v − q)) = ∅ then
3:     return BENEATH
4:   end if
5:   if q ∉ CH(F) then
6:     if det(p_0, p_m, q) > 0 then
7:       return BENEATH
8:     end if
9:     if det(p_0, p_m, q) < 0 then
10:      return BEYOND
11:    else
12:      return ON
13:    end if
14:  end if
15:  while q ∈ CH(F) ∧ diam(F) > Δ_3 do
16:    (F, l) ← RefineRep(F, l(u) = q + u(v − q))
17:  end while
18:  if diam(F) < Δ_3 then
19:    return ON
20:  end if
21: end loop
```

i det konvekse hylster, eller også er diameteren for den forfinede kurve F_i blevet mindre end Δ_3, fra sætning 3.2.8. Hvis det sidste er tilfældet, afsluttes algoritmen med resultatet ON, ellers sættes $F = F_i$, og algoritmen gentages med den nye kurve.

5.4.4 Fortegnet på α-vinklen

Sætning 4.3.1 i afsnittet om *Det komplette kriterium for ikke-krydsende, tangentiel skæring* beskrives, hvordan det er nødvendigt at kunne beregne fortegnet til vinklerne $\alpha(0)$ og $\alpha(1)$. I dette afsnit gøres rede for en algoritme, som kan beregne disse fortegn. Algoritmen hedder AlphaSign og stammer fra Yaps artikel [4].

Lad $F[a,b]$ være en elementær Bézier, så gælder $F(t) = (F_x(t), F_y(t))$ for $t \in [a,b]$. For ethvert $t \in [a,b]$ findes en normal til kurven F i punktet $F(t)$, denne normal har en vinkel $\theta_F(t)$ (relativ til den positive x-akse) som er givet ved

$$\cos(\theta_F(t)) = \frac{-F'_y(t)}{\sqrt{(F'_x(t))^2 + (F'_y(t))^2}}, \quad \sin(\theta_F(t)) = \frac{F'_x(t)}{\sqrt{(F'_x(t))^2 + (F'_y(t))^2}} \tag{5.1}$$

hvor $F'(t) = (F'_x(t), F'_y(t))$ er den afledte til F med hensyn til t. Der gælder også, at $\theta_F(t) \in [0, \pi]$, fordi der for en elementær kurve gælder, at $F'_x(t) > 0$ for alle $t \in [a,b]$.

Lad også $l(u) = (cu + d, eu + f)$, $c, d, e, f \in \mathbb{R}$ og $e > 0$ være en ret linie. Så vil vinklen, θ_l til l (relativ til den positive x-akse) være defineret

$$\cos(\theta_l) = \frac{c}{\sqrt{1 + c^2}}, \qquad \sin(\theta_l) = \frac{e}{\sqrt{1 + c^2}} \tag{5.2}$$

og når $e > 0$, så gælder der, at $\theta_l \in (0, \pi)$. Bemærk, at denne antagelse, $e > 0$ ikke ændrer mængden af linier, som kan repræsenteres ved $l(u) = (cu+d, eu+f)$, udover at fjerne muligheden for vandrette linier. For eksempel, hvis l er givet, så $e < 0$, så vil $-l(u) = (-ci - d, -eu - f)$ repræsentere den samme linie, som l, men $-e > 0$.

Lad nu linien l skære kurven F i punktet $F(t^*)$ for $t^* \in [a, b]$. Så vil vinklen $\alpha(t^*)$ være givet ved

$$\alpha(t^*) = \theta_F(t^*) - \theta_l$$

og der gælder, at $\alpha(t) \in (-\pi, \pi)$, da $\theta_F(t) \in [0, \pi]$ for $t \in [a, b]$ og $\theta_l \in (0, \pi)$. Målet er at finde fortegnet til $\alpha(t^*)$, så det bemærkes at $\sin(-x) = -\sin(x)$, så fortegnet for $\alpha(t^*)$ har samme fortegn som $\sin(\alpha(t^*))$, og der gælder så

$$\begin{aligned} \sin(\alpha(t^*)) &= \sin(\theta_F(t^*) - \theta_l) \\ &= \sin(\theta_F(t^*))\cos(\theta_l) - \cos(\theta_F(t^*))\sin(\theta_l) \end{aligned} \tag{5.3}$$

Hvis udtrykkene fra (5.1) og (5.2) indsættes i (5.3), kan fortegnet for $\alpha(t^*)$ reduceres til følgende udtryk

$$\begin{aligned} \sin(\alpha(t^*)) &= \sin(\theta_F(t^*) - \theta_l) \\ &= \sin(\theta_F(t^*))\cos(\theta_l) - \cos(\theta_F(t^*))\sin(\theta_l) \\ &= \frac{F'_x(t)}{\sqrt{(F'_x(t))^2 + (F'_y(t))^2}} \frac{c}{\sqrt{1+c^2}} - \frac{-F'_y(t)}{\sqrt{(F'_x(t))^2 + (F'_y(t))^2}} \frac{e}{\sqrt{1+c^2}} \\ &= \frac{cF'_x(t) - eF'_y(t)}{\sqrt{1+c^2}\sqrt{(F'_x(t))^2 + (F'_y(t))^2}} \end{aligned}$$

hvor det bemærkes, at nævneren altid er positiv, så fortegnet for $\alpha(t^*)$ kan reduceres til fortegnet af udtrykket

$$s_0 = cF'_x(t) + eF'_y(t) \tag{5.4}$$

Hvis $t^* \in [a, b]$ er skæringspunktet mellem F og G, så gælder der, at $F(t^*) = l(u^*)$ hvor

$$u^* = \frac{F_x(t^*) - d}{c} = \frac{F_y(t^*) - f}{e}$$

hvilket giver anledning til ligningen

$$eF_x(t^*) - de = cF_y(t^*) - cf$$

Nu gælder der, at udtrykket s_0 fra (5.4) er et nulpunkt i det polynomielle udtryk givet resultanten

$$res_t(B_1(s,t), B_2(s,t))$$

hvor de to polynomier B_1 og B_2 er givet således

$$B_1(s,t) = s - cF'_x(t) - eF'_y(t), \qquad B_2(s,t) = eF_x(t) - cF_y(t) + (cf - de)$$

Polynomiet $B_2(s,t)$ afhænger faktisk ikke af s, men det betyder ikke noget for resultanten. Graderne af polynomierne B_1 og B_2 er henholdsvis $max\{1, m-1\}$ og m for en elementær (Bézier) kurve af grad m.

Det er nu mulig at give en estimering af størrelsen s_0 ved hjælp af følgende sætning. Denne estimering bruges i AlphaSign algoritmen til bestemmelse af fortegnet for α vinklen.

Sætning 5.4.12. *Lad kontrol polygonen, $P(F) = (p_0, \ldots, p_m)$, være givet således, at koordinaterne til p_i er givet ved L-bit flydende kommatal. Lad ligeledes konstanterne c, d, e, f til linien, $l(u) = (cu + d, eu + f)$ være givet ved L-bit flydende kommatal. Så gælder det, at hvis $s_0 \neq 0$, så er*

$$|S_0| \geq (6m128^L 9^m)^{-m}$$

Bevis. Man kan ikke umiddelbart være sikker på, at B_1 og B_2 har heltalskoefficienter, så i stedet benyttes $2^L B_1$ og $4^L B_2$, som har heltalskoefficienter, hvilket medfører, at normerne vokser med henholdsvis en faktor 2^L og 4^L. Resultanten $res_t(2^L B_1(s,t), 4^L B_2(s,t))$ er så, ifølge [7, Theorem 6.31], begrænset af $(2^L v_1)^m (4^L v_2)^m$, hvor v_1 og v_2 er $(2,1)$-normen for henholdsvis B_1 og B_2. Da s_0 er et nulpunkt i $res_t(2^L B_1(s,t), 4^L B_2(s,t))$, så gælder det, at hvis $s_0 \neq 0$, så er

$$|s_0| \geq \frac{1}{(2^L v_1)^m (4^L v_2)^m} \tag{5.5}$$

Der mangler nu kun at argumentere for størrelserne af v_1 og v_2. Ifølge lemma 3.3.1 kan koefficienterne, a_k til $F_x(t) = \sum_{i=0}^m a_i t^i$ begrænses af $2^{L+k}\binom{m}{k}$. Ved hjælp af lemma 3.3.3 kan nu konkluderes, at $\|F_x(t)\|_1 \leq 2^L 3^m$. Hvis $L \geq 1$, så gælder $\|F_x(t)\|_2 \leq 2^L 3^m - 1$. Det følger så, at $\|F'_x(t)\|_2 \leq m(2^L 3^m - 1) \Rightarrow \|eF'_x(t)\|_2 \leq m2^L(2^L 3^m - 1)$. En tilsvarende grænse gælder for $\|cF'_y(t)\|_2 \leq m2^L(2^L 3^m - 1)$. Grænsen for v_1 er så

$$v_1 \leq 1 + 2m2^L(2^L 3^m - 1) < 2m4^L 3^m$$

og tilsvarende for v_2

$$v_2 \leq 2 \cdot 2^L(2^L 3^m - 1) + 2 \cdot 4^L < 2m4^L 3^m$$

Grænserne for v_1 og v_2 kan nu umiddelbart sættes ind i (5.5). □

AlplhaSign algoritmen

Betragt det følgende problem: Givet en direkte kurve $G[s_0, s_1]$ og en direkte linie, l, som skærer G i et entydigt punkt $G(t^*)$ hvor $t^* \in [s_0, s_1]$. Beregn fortegnet på vinklen mellem l og den normal til G, som går igennem $G(t^*)$. Opgaven består så blot i at beregne fortegnet på funktionen

$$g(t) = cG'_x(t) + eG'_y(t)$$

i punktet $t = t^*$, hvor $G(t) = (G_x(t), G_y(t))$ og $G'(t) = (G'_x(t), G'_y(t))$ er den afledte til G med hensyn til t, og hvor c er hældningskoefficienten til linien l. Det eneste problem er bare, at t^* ikke kan antages at være et direkte tal, så det er ikke umiddelbart muligt at evaluere funktionen $g(t)$ i $t = t^*$. Måden, hvorpå dette problem omgås, er ved at bruge $g(s_0)$ som estimat til $g(t^*)$ og så undersøge, om vi kan sige noget omkring fortegnet. En mere detaljeret forklaring følger.
Funktionen $g : \mathbb{R} \to \mathbb{R}$ givet $g(t) = cG'_x(t) + eG'_y(t)$ udgør et polynomium af grad højst m, da kurven G er givet ved $P(G) = (q_0, \ldots, q_m)$. Så g er m gange differentiabel og Taylor polynomiet $T_m g(t; s_0)$ for g omkring punktet s_0 er defineret

$$T_m g(t; s_0) = \sum_{k=0}^m \frac{g^{(k)}(s_0)}{k!}(t - s_0)^k = g(s_0) + \sum_{k=1}^m \frac{g^{(k)}(s_0)}{k!}$$

hvor $g^{(k)}(t)$ er $g(t)$ differentieret k gange. Det er nu muligt at give en øvre grænse, ϵ, for fejlen ved at bruge $g(s_0)$ som estimat til $g(t^*)$.

$$\begin{aligned}
|g(t^*) - g(s_0)| &= |T_m g(t^*) - g(s_0)| \\
&= \left| \sum_{k=0}^{m} \left(\frac{g^{(k)}(s_0)}{k!} (t^* - s_0)^k \right) - g(s_0) \right| \\
&= \left| g(s_0) + \sum_{k=1}^{m} \left(\frac{g^{(k)}(s_0)}{k!} (t^* - s_0)^k \right) - g(s_0) \right| \\
&= \left| \sum_{k=1}^{m} \frac{g^{(k)}(s_0)}{k!} (t^* - s_0)^k \right| \\
&\leq \sum_{k=1}^{m} \left| \frac{g^{(k)}(s_0)}{k!} (t^* - s_0)^k \right| \\
&\leq \sum_{k=1}^{m} \left| g^{(k)}(s_0) \right| \frac{(s_1 - s_0)^k}{k!} = \epsilon
\end{aligned}$$

Bemærk, at den sidste ulighed, gælder, fordi $s_0 \leq t^* \leq s_1$. Denne grænse, ϵ, kan nu bruges på følgende måde. Hvis der om estimatet $g(s_0)$ til $g(t^*)$ gælder, at $|g(s_0)| \geq \epsilon$, så har $g(s_0)$ og $g(t^*)$ samme fortegn. Dette kan nu anvendes til at udvikle en adaptiv algoritme, som bestemmer fortegnet på α-vinklen.

Algoritme 5.4.5 AlphaSign(l, G, s_0, s_1)

Input: En (direkte) ret linie, l, en (direkte) elementær kurve, $G[s_0, s_1]$, hvorom der gælder, at l og G skærer hinanden i punktet $G(t^*)$ for $t^* \in [s_0, s_1]$.
Output: Fortegnet på vinklen $\alpha(t^*) = \theta_G(t^*) - \theta_l$.

```
1: loop
2:   for i = 0 to m do
3:     g^(i)(s_0) ← CalculateDerivative(i)
4:   end for
5:   ε ← Calculateε(s_0, s_1)
6:   if |g(s_0)| ≥ ε then
7:     return sign(g(s_0))
8:   end if
9:   if ε ≤ (6m128^L 9^m)^(-m) then
10:    return 0
11:  end if
12:  if t* ∈ [s_0, (s_0 + s_1)/2] then
13:    s_1 = (s_0 + s_1)/2
14:  else
15:    s_0 = (s_0 + s_1)/2
16:  end if
17: end loop
```

En kort forklaring af algoritme 5.4.5 følger. Det centrale i denne algoritme er, at den udelukkende arbejder på direkte objekter. På trods af det kan den stadig beregne et eksakt fortegn til den indirekte værdi $\alpha(t)$ for $t \in [s_0, s_1]$. Se eventuelt afsnit 2.6 for mere information om direkte objekter.
Algoritmen begynder med at beregne alle de m første afledte til funktionen g, hvorefter den udregner den øvre grænse, ϵ, til fejlen, som hører til estimatet $g(s_0)$ for udtrykket $g(t^*)$ (se ovenfor for flere detaljer). Bemærk, at CalculateDerivative og Calculateϵ ikke er tiltænkt at være selvstændige delalgoritmer, men dækker

over delprocesser i algoritmen. For eksempel gøres, under beregningen af de afledte til $g(t)$ i delprocessen CALCULATEϵ, brug af de værdier for $g^{(i)}(s_0)$, som beregnes under delprocessen CALCULATEDERIVATIVE. Næste skridt i algoritmen, linie 6-8, udgør en kontrol af, om det estimat, $g(s_0)$, til $g(t^*)$ er præcist nok til at afgøre om fortegnet er positivt eller negativt. I givet fald afsluttes algoritmen. Hvis estimatet ikke var præcist nok til at lave denne afgørelse, kontrolleres så, om præcisionen er tilstrækkelig til at afgøre, om fortegnet er neutralt, e.i. 0. Dette gøres i linie 9-11. Hvis algoritmen når til linie 12, betyder det, at den ikke var i stand til at afgøre fortegnet på $\alpha(t^*)$ med det nuværende estimat af skæringspunktet $G(t^*)$, hvor $t \in [s_0, s_1]$. Bemærk, at hvis (G, l, s_0, s_1) udgør et estimat af det indirekte skæringspunkt $G(t^*)$ mellem G og l for $t^* \in [s_0, s_1]$, så vil (G, l, s_0', s_1') være et bedre estimat, hvis der gælder $s_0 \leq s_0' \leq t^* \leq s_1' \leq s_1$. Linie 12-16 i algoritmen foretager følgende forbedring af estimatet af (G, l, s_0, s_1). Hvis $t^* \in [s_0, (s_0 + s_1)/2]$, så sættes $s_0 = (s_0 + s_1)/2$, ellers sættes $s_1 = (s_0 + s_1)/2$, hvorefter hele algoritmen udføres med det forberede etstimat.

I Yaps artikel [4] gør han opmærksom på, at en implementering med fordel kan genbruge nogle af de beregnede værdier for $g^{(i)}(s_0)$. For eksempel er det ikke nødvendigt at beregne nye værdier for $g^{(i)}(s_0)$, hvis den foregående iteration har ændret intervallet $[s_0, s_1]$ til $[s_0, (s_0 + s_1)/2]$ i linie 12-16.

Yap giver også et hint om, at de afledte $g^{(i)}(s_0)$ kan beregnes ved hjælp af teknikker indenfor *automatic differentiation*, som gør det mulig at beregne alle de afledte til $g(s_0)$ således, at det ikke koster meget mere end at beregne $g(s_0)$. Denne påstand er dog ikke bekræftet!

Til sidst gives en kort beskrivelse af kompleksiteten af ALPHASIGN algoritmen. Kompleksiteten af ALPHASIGN er adaptiv i den forstand, at *worst-case* kompleksiteten kun opnås i tilfælde af, at algoritmen afsluttes i linie 10. Antallet af iterationer er afgjort af, hvor præcis en tilnærmelse (G, l, s_0, s_1) er til det eksakte skæringspunkt $G(t^*)$.

Kapitel 6

Implementering og evaluering

Evalueringen af Yaps algoritme i forhold til effektivitet i praksis vil foregå på to område. For det første gøres rede for, at Yaps algoritme reelt udgør en komplet løsning af skæringsproblemet, hvor de fundne løsninger, i.e. skæringspunkter, er eksakte, da en undersøgelse af effektiviteten i praksis ikke giver mening for en algoritme, der ikke opnår sine forventede mål. Den anden del af evalueringen består i at se, hvor meget det koster tidsmæssigt at rette op på de robusthedsproblemer, i.e. at opnå komplethed og eksakte løsninger, som kendte opdelingsbaserede algoritmer lider af. Denne tidsmæssige sammenligning gøres ved at sammenholde udførselstiderne for en implementering af henholdsvis den generelle skæringsalgoritme baseret på opdeling og så Yaps skæringsalgoritme. En sådan sammenligning er relativ nem at udføre, da den generelle skæringsalgoritme jo udgør makrofasen i Yaps algoritme.

6.1 Komplethed og eksakte løsninger

Konceptuelt virker Yaps algoritme på følgende måde. Makrofasen deler input kurverne F og G i et endeligt antal kandidatpar (F_i, G_j). Opdelingen foregår ved at dele den kurve, hvis konvekse hylster har størst diameter, og stopper først, når hvert kurvepar opfylder betingelsen om, at diameteren af foreningsmængden af deres respektive konvekse hylstre er mindre end Δ^*. På denne måde fremkommer et endeligt, men muligvis stort, antal kurvepar som udgør kandidater til ovennævnte repræsentanter for skæringspunkter mellem kurverne F og G. På disse kurvepar bruges nu reject-kriteriet, baseret på det faktum, at en Bézier kurve er helt indeholdt i sit konvekse hylster, til at fjerne de kurvepar, hvor kurvernes konvekse hylstre ikke skæret hinanden. Tilbage er så netop de kurvepar, der udgør kandidatpar og altså opfylder følgende tre ting.

- $F_i \subseteq F$ og $G_j \subseteq G$.
- $diam(CH(F_i) \cup CH(G_j)) < \Delta^*$.
- $CH(F_i) \cap CH(G_j) \neq \emptyset$.

Hvis der for de originale input kurver gælder, at de kun indeholder et endeligt antal (F, G)-antipodalpar, så vil alle skæringspunkter mellem F og G være isolerede punkter. Parres denne antagelse med de tre ovenstående, egenskaber betyder det, at hvert kandidatpar (F_i, G_j) højst kan have et entydigt skæringspunkt. Grunden

til dette er, at sætning 3.2.6, der afhænger af antagelsen om antipodalpar, giver den en mindste grænse for, hvor tæt to skæringspunkter kan være på hinanden, og denne afstand er større end Δ^*.
Mikrofasen kan nu for hvert kandidatpar (F_i, G_j), som opfylder de tre ovenstående egenskaber, afgøre, om der findes et entydigt skæringspunkt mellem F_i og G_j. Hvis der gør, så udgør (F_i, G_j) en repræsentant for et skæringspunkt i løsningsmængden.

Med denne konceptuelle gennemgang af Yaps algoritme er det nu muligt at gøre rede for, hvorledes den opnår komplethed og eksakte løsninger. Først mindes om, hvad begreberne eksakthed og komplethed står for.

- **Eksakthed** En skæringsalgoritme kaldes eksakt, hvis de løsninger, den finder, udgør eksakte
- **Komplethed** En skæringsalgoritme kaldes komplet, hvis den finder samtlige skæringspunkter. entydige skæringspunkter.

Bemærk, disse to krav siger ikke noget om, hvorledes løsningsmængden, eller de enkelte løsninger repræsenteres. I Yaps skæringsalgoritme for Bézier kurver repræsenteres løsningsmængden, for et par af kurver, F og G, som en kø af skæringspunkter. De enkelte skæringspunkter er givet ved kurvepar (F', G'), hvorom der gælder, at $F' \subseteq F$, $G' \subseteq G$ og $F' \cap G' = p$, hvor $p \in \mathbb{E}^2$. Det vil sige, kurverne F' og G' er delkurver af de oprindelig kurver, og de indeholder præcis et entydigt skæringspunkt. Grunden til, at skæringspunkterne ikke er eksplicit givet, er, at koordinaterne kan være enten irrationale eller rationale, men kan ikke repræsenteres som bigfloat tal.

Eksakthed

Yaps algoritme opnår eksakthed ved at benytte bigfloat tal og så reducere alle sine beregninger til eksakt aritmetiske operationer $(+, -, \times)$ på disse bigfloat tal. Der findes dog omstændigheder, hvor reduktionen til simple regneoperationer kan være lidt problematisk, nemlig når der indgår størrelser, som ikke kan repræsenteres som bigfloat tal. I disse situationer regnes kun indirekte på de ikke-eksakte størrelser. Afsnit 5.4.4 er et eksempel på en sådan situation. Her ønskes fortegnet af udtrykket $g(t^*)$ beregnet for en funktion $g : \mathbb{R} \to \mathbb{R}$. Problemet er bare, at t^* ikke nødvendigvis kan repræsenteres direkte, altså ved bigfloat tal. Så dermed kan funktionsværdien heller ikke repræsenteres direkte. Den måde, som dette problem løses på, er ved at bruge følgende faktum. Hvis E er et eksakt tal og E' en tilnærmelse til E. Så gælder der, at fortegnet på E' er det samme som fortegnet på E, hvis der gælder $|E - E'| < |E'|$. Fortegnet på $g(t*)$ kan så evalueres ved at se på, om fejlen ved at bruge $g(s)$ istedet, hvor s kan repræsenteres som en bigfloat, som estimat til størrelsen $g(t^*)$. Hvis fejlen $|g(t^*) - g(s)|$ er mindre end $|g(s)|$, så har $g(t^*)$ samme fortegn som $g(s)$.
Ved denne slags teknikker opnår Yaps algoritme eksakte løsninger selv i tilfælde, hvor ikke-eksakte størrelser umiddelbart indgår i beregninger.

Komplethed

Yaps skæringsalgoritme opnår komplethed. Denne påstand er baseret på to fakta. For det første hvis det antages, at to Bézier kurver F og G skærer hinanden i et entydigt punkt $p \in F \cap G$. Så må der gælde, at hvis F deles i F_0 og F_1 i et punkt forskelligt fra p, så må p ligge på enten F_0 eller F_1.
For det andet findes, for Bézier kurverne F og G, en mindste afstand mellem to forskellige skæringspunkter, hvilket betyder, at to kurver kan deles op således,

at hver kurvepar højst indeholder et skæringspunkt. Bemærk, at dette faktum afhænger af antagelsen om at kurverne F og G kun har endeligt mange (F, G)-antipodalpar.
Med disse to fakta er det nu ikke svært at se, hvorfor Yaps algoritme opnår komplethed. Når makrofasen er afsluttet, har den fundet et endeligt antal kandidatpar, som højst kan indeholde et entydigt skæringspunkt. Alle de andre par, som blev smidt væk under makrofasen, kan ikke indeholde skæringspunkter, fordi deres konvekse hylstre ikke skærer hinanden, men de er trods alt blevet undersøgt. Mikrofasen løber nu alle kandidatpar igennem og for hvert kandidatpar afgøres, om kurver indeholder et entydigt skæringspunkt. Hvordan denne afgørelse foretages, beskrives i 5.4.1 og 5.4.2, men der gælder specielt, at der er tale om et komplet kriterium. Det vil sige, når Yaps skæringsalgoritme er færdig, så er alle skæringspunkter blevet fundet, og algoritmen er dermed komplet.

6.2 Effektivitet i praksis

Om en algoritme er effektiv i praksis er generelt svært at afgøre, da en sådan afgørelse nødvendigvis må have med domænet, hvor algoritmen finder sin anvendelse, at gøre. Yaps algoritme er ikke tiltænkt anvendt i et specifikt domæne, så der er ikke umiddelbart nogle oplagte mål for effektiviteten i praksis for en given implementering. Men hvis Yaps algoritme betragtes som et forsøg på at udbedre de robusthedsproblemer, som en tilsvarende, ikke komplet opdelingsbaseret skæringsalgoritme lider af, kan effektiviteten i praksis jo afgøres ved at se på den tidsmæssige pris, som betales for at få løst robusthedsproblemerne i aktuelle implementeringer.

Den generelle skæringsalgoritme, baseret på opdeling (subdivision), er givet i algoritme 6.2.1. Den er givet her med et minimalt antal partielle kriterier, hvilket vil sige, at den kun indeholder reject-kriteriet baseret på konveks hylster egenskaben for Bézier kurver, som generelt bruges til at fjerne kurvepar, som ikke kan skære hinanden.
GenericIntersection algoritmen fungerer på følgende måde. Input kurverne deles op i kurvepar ved gentagene gange at dele den kurve, som har størst diameter. Bemærk, at diameteren for en Bézier kurve er givet ved diameteren af det konvekse hylster af kontrol polygonen. For hvert sådan fremkomne kurvepar kontrolleres, om de udgør et kandidatpar, i.e. der kontrolleres for, om deres respektive konvekse hylstre skærer hinanden. Hvis kurveparret ikke udgør et kandidatpar, smides dette par bort. Den gentagne opdeling af kurveparrene forsætter, indtil der enten ikke er flere kurvepar at dele, eller indtil diameteren af foreningsmængden af de respektive konvekse hylstre er mindre end ϵ. Hvis et kandidatpar har en sådan diameter, betragtes kandidatparret, som en repræsentant for et skæringspunkt, og rapporteres. Det er nu oplagt, hvorfor GenericIntersection ikke er en komplet skæringsalgoritme. Problemet består umiddelbart i, at der ikke er nogen måde at vide på, om et givet kandidatpar har nul, et eller flere skæringspunkter, når parret rapporteres som skæringspunkt. Så de kandidatpar, der rapporteres fra GenericIntersection algoritmen, kan altså indeholde et vilkårligt antal skæringspunkter, så længe diameteren af kurvernes foreningsmængde er mindre end ϵ. Om problemerne med denne algoritme kan løses, og GenericIntersection gøres komplet, er ikke oplagt. Men hvis det skal lykkedes skal to ting være opfyldt.

(i) Efter opdelingen af input kurverne er afsluttet, skal der gælde, at ethvert kandidatpar kan højst have et skæringspunkt.

(ii) Der findes en metode til at afgøre, om et givet kandidatpar har et skæringspunkt.

Algoritme 6.2.1 GenericIntersection(F, G, ϵ)

Input: $P(F) = (p_0, p_1, \ldots, p_m)$, $P(G) = (q_0, q_1, \ldots, q_n)$ og $\epsilon > 0$.
Output: En kø S, som indeholder kandidatpar (F_i, G_j), hvorom der gælder, at $F_i \subseteq F$, $G_j \subseteq G$ og $F_i \cap G_j$ er muligvis ikke tom.

```
1: S ← ∅    Q ← ∅
2: Q ← (F, G) {(F, G) sættes ind i Q}
3: while Q ≠ ∅ do
4:    (F', G') ← Q {(F', G') tages ud af Q}
5:    if CH(F') ∩ CH(G') ≠ ∅ then
6:       if Diam(CH(F') ∪ CH(G')) < ε then
7:          S ← (F', G')
8:       else if Diam(CH(F')) > Diam(CH(G')) then
9:          (F_0, F_1) ← Subdivide(F')
10:         Q ← (F_0, G')
11:         Q ← (F_1, G')
12:      else
13:         (G_0, G_1) ← Subdivide(G')
14:         Q ← (F', G_0)
15:         Q ← (F', G_1)
16:      end if
17:   end if
18: end while
19: return S
```

Det er netop disse to ting, som Yap adresserer i sin artikel [4]. Han beviser blandt andet, udfra input kurvenes algebraiske egenskaber, at der findes grænser for, hvor tæt to forskellige skæringspunkter kan være på hinanden. Dette muliggør bestemmelsen af en værdi for ϵ i GenericIntersection, som kan garantere, at de opdelte kandidatpar højst har et skæringspunkt, hvis deres foreningsmængde har en diameter, der er mindre end ϵ. Denne værdi kaldes Δ^*, og hvis ϵ sættes lig Δ^* i GenericIntersection, er (i) løst. Grunden til, at dette er rigtigt, er givet i detaljer i kapitel 3, men er kort sagt, at Δ^* udgør en diameter, som er mindre end den mindste afstand mellem to forskellige skæringspunkter på input kurverne. Så hvis et kurvepar har diameter mindre end Δ^*, så kan det højst indeholde et skæringspunkt. Yap viser også, hvordan man afgør, om et givet kandidatpar indeholder et skæringspunkt, hvis kandidatparrets diameter er mindre end Δ^* værdien, og altså højst kan have et skæringspunkt. Detaljerne, hvor dette gøres i praksis, kan findes i kapitel 5. Yaps skæringsalgoritme kan derfor betragtes som en udvidelse af GenericIntersection på følgende måde. Initialiseringsfasen i GenericIntersection udvides, så den beregner den korrekte ϵ værdi, så (i) ovenfor opfyldes. Den iterative del af GenericIntersection algoritmen, kaldes nu makrofasen, fordi den behandler makropar, i.e. den laver dem om til mikropar. Efter makrofasen tilføjes en iterativ proces, som afgører, for hvert mikropar, der slipper igennem makrofasen, om parret indeholder et entydigt skæringspunkt. Denne nye iterative proces kaldes mikrofasen, fordi den behandler mikropar.

Med en sådan opbygning af Yaps skæringsalgoritme er det naturligt at undersøge to ting med henblik på at evaluere effektiviteten af en given implementering i praksis. For det første kan ϵ værdien vælges frit, i en implementering af GenericIntersection algoritmen. Så på den måde giver ϵ en mulighed for at styre effektiviteten. Hvis man ønsker en hurtig løsning af skæringsproblemet, men som knap er så præcis eller korrekt, kan man sætte ϵ lavt. Denne frihedsgrad findes ikke i Yaps algoritme.

Her afhænger ϵ værdien netop af egenskaberne for input kurverne. Så en interessant ting at undersøge for en implementering af Yaps algoritme ville være, om algoritmen kan opnå effektivitet i praksis, når ϵ værdien ikke kan vælges frit.
For det andet, under antagelse af at makrofasen stadig kan opnå effektivitet i praksis, når ϵ ikke kan vælges frit, hvor meget koster det så at udføre mikrofasen i praksis.
Det store spørgsmål er altså, om fastsættelsen af ϵ værdien i Yaps algoritme ødelægger dennes mulighed for at opnå effektivitet i praksis. Som det vil vise sig, har udvidelsen til Yaps algoritme store negative konsekvenser for effektiviteten af netop denne grund. Det følgende er derfor en redegørelse for denne sammenhæng.

Yaps algoritme - en delvis implementering

Grundlaget for min implementering af Yaps algoritme har været en evaluering af dennes effektivitet i praksis. Som nævnt ovenfor forventes effektiviteten i praksis at være påvirket af, at ϵ ikke kan vælges frit i makrofasen. Dette har ført til en delvis implementering af Yaps skæringsalgoritme, hvor jeg har fokuseret på makrofasen for at få afklaret, om fastsættelsen af ϵ vil ødelægge muligheden for effektivitet i praksis. Det skulle nemlig vise sig at være tilfældet.
I det følgende skitseres min implementering af initialiseringsfasen og makrofasen af Yaps skæringsalgoritme. Formålet med denne skitse er at blive i stand til at give en kompleksitetsanalyse af relevante dele af af Yaps algoritme. Source-koden til makrofasen er givet i appendiks B.

Initialiseringsfasen

Initialiseringsfasen består primært af en beregning af en geometrisk separationsgrænse Δ^*. Se afsnit 5.4.1 for en definition. Beregningen af Δ^* er baseret på eksakte bigfloat tal og kan reduceres til eksakte regneoperationer $(+, -, \times)$. Som nævnt i afsnit 5.2 så er Δ^* et minimum af fire andre separationsgrænser. Disse fire separationsgrænser beregnes alle ved hjælp af de matematiske begreber; fakultet, potenser og binomial koefficienter. Fakultet, potenser og binomial koefficienter kan reduceres til eksakte regneoperationer på bigfloat tal på følgende måde

- **Fakultet** $n! = \prod_{k=1}^{n} = k$
- **Potens** $x^n = x^n = x \times x \times \cdots \times x$
- **Binomial koefficient** $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

hvilket udnyttes i analysen af worst-case kompleksiteten af initialiseringsfasen af Yaps algoritme. Hvis m og n er graden af input kurverne, og koordinaterne til kontrol punkterne er givet ved (L, l)-bit flydende kommatal, så vil worst-case kompleksiteten af initialiseringsfasen været givet ved O(m + n + L). Dette skyldes, at

$$\Delta^* = \min\{\Delta_1(m, n, a, b), \Delta_2(m, n, a, b), \Delta_4(m, n, a, b, L), \Delta_6(m, n, L)\}$$

hvor a og b er givet i sætning 3.3.2, og de andre fire separationsgrænser er givet i kapitel 3 og 5. De enkelte geometriske separationsgrænser kan reduceres til et lineært antal simple regneoperationer i variablerne m, n og L. Bemærk, variablerne a og b er ikke nævnt. Det er fordi de selv er opbygget af et lineært antal simple regneoperationer i de andre variable. Så worst-case kompleksiteten af initialiseringsfasen bliver $O(m + n + L)$.

Makrofasen

Makrofasen er som nævnt ovenfor, identisk med GenericIntersection givet i algoritme 6.2.1. Som det kan ses i pseudo-koden, kan denne fase opdeles i et antal mindre algoritmer, nemlig ConvexHull, ConPolyInt, Diam og Subdivide. De enkelte delalgoritmer beskrives nedenfor.

ConvexHull
Algoritmen beregner det konvekse hylster, defineret i sætning 2.6.2, for en endelig punktmængde. Bogen [6, afsnit 1.1] beskriver en algoritme til beregning af det konvekse hylster med worst-case kompleksitet $O(n \log n)$, hvor n er antallet af punkter, som det konvekse hylster skal beregnes for. Algoritmen bruges i to situationer. For det første bruges den til at beregne det konvekse polygon, som indeholder en givet Bézier kurve. For det andet bruges den i MergeConvexHulls algoritmen til at bestemme det mindste polygon, som indeholder kontrol punkterne for to givne Bézier kurver.

ConPolyInt
Hvis to konvekse polygoner skærer hinanden, kan denne algoritme beregne dette polygon. Den bruges i forbindelse med konveks hylster kriteriet, som bruges til at afgøre, om to kurver udgør et kandidatpar, det vil sige, at de opfylder $CH(F') \cap CH(G') \neq \emptyset$. Bogen [9, afsnit 7.6] beskriver en implementering af denne algoritme. Worst-case kompleksiteten er $O(n+m)$, hvor n og m er antallet af hjørnepunkter i de respektive polygoner.

Diam
Algoritmen beregner diameteren for et konvekst polygon, hvor der med diameteren menes den maksimale afstand mellem to punkter indeholdt i det konvekse polygon. Et generelt problem med beregningen af diameter, i forhold til makrofasen, er, at den er afhængig af beregning af den euklidiske afstandsfunktion $|p-q| = \sqrt{(p.x-q.x)^2+(p.y-q.y)^2}$. Problemet består i, at denne afstandsfunktion ikke kan reduceres til et antal simple og eksakte regneoperationer på bigfloat tal, fordi $\sqrt{\cdot}$ ikke er en eksakt operation på bigfloat tal. Så en anden metode til beregningen af diameteren er nødvendig. Til dette formål observeres, at diameteren af en konveks polygon altid er mindre eller lig diagonalen i det rektangel, som indeholder polygonen, og har sider parallelle med de to akser. For et polygon givet ved $P(F) = (p_0, p_1, \ldots, p_m)$, så er ovennævnte diagonal, afstanden mellem punkterne (x_{min}, y_{min}) og (x_{max}, y_{max}). Hvis samtidig bruges, at

$$\begin{aligned} |p-q| &= \sqrt{(p.x-q.x)^2+(p.y-q.y)^2} \\ &\leq \sqrt{(p.x-q.x)^2}+\sqrt{(p.y-q.y)^2} \\ &= (p.x-q.x)+(p.y-q.y) \end{aligned} \tag{6.1}$$

så kan diagonalen i det omsluttende rektangel estimeres ved hjælp af (6.1). Spørgsmålet er, om et sådant estimat vil påvirke makrofasen. Svaret er nej. Grunden er, som det kan ses i algoritme 6.2.1, at diameteren kun bruges to steder. Det første sted sammenlignes diameteren for $CH(F') \cup CH(G')$ med grænsen Δ^*. Om denne sammenligning gælder, at hvis den estimerede diameter fra (6.1) er mindre end Δ^*, så er den eksakte diameter også mindre. Den anden anvendelse af diameteren i algoritme 6.2.1, består kun i at afgøre, hvilken kurve, der skal opdeles. Denne beslutning påvirker ikke udfaldet af makrofasen, så den estimerede diameter kan godt bruges i stedet for den eksakte. Worst-case kompleksiteten af Diam algoritmen bliver O(n), hvor n er antal hjørnepunkter i polygonen, da det er nødvendigt at gennemløbe alle disse hjørnepunkter for at finde minimum og maksimum af koordinaterne.

SUBDIVIDE
Algoritmen beregner kontrol punkterne til de to Bézier kurver, som dukker op, når en given Bézier kurve opdeles ved hjælp af deCasteljaus algoritme. Denne process er beskrevet i afsnit 2.5, og er baseret på deCasteljaus algoritme. Worst-case kompleksiteten er den samme som for deCasteljaus algoritme, nemlig lineær i antallet af kontrol punkter i den opdelte kurves kontrol polygon. Det vil sige, for en Bézier kurve F, givet ved $P(F) = (p_0, p_1, \ldots, p_n)$, vil SUBDIVIDE algoritmen have $O(n)$ kompleksitet.

Fra det ovenstående kan kompleksiteten af hele makrofasen gøres op som følgende. Lad i det følgende F og G være de kurver, som makrofasen forsøger at finde skæringspunkter for. De to kontrol polygoner er givet ved henholdsvis $P(F) = (p_0, p_1, \ldots, p_m)$ og $P(G) = (p_0, p_1, \ldots, p_n)$. Bemærk, at når en Bézier kurve opdeles med SUBDIVIDE algoritmen, vil de to delkurver have det samme antal kontrol punkter, som den kurve der blev delt.
Under afvikling af makrofasen vil løkken bliver gennemløbet et endeligt antal gange. Hvert gennemløb består af et antal anvendelser af de ovennævnte fire delalgoritmer. Ved at tælle, hvor mange gange de enkelte delalgoritmer anvendes, fremkommer følgende skema.

CONVEXHULL	4	$O((m+n)\log(m+n))$
CONPOLYINT	1	$O(m+n)$
DIAM	3	$O(m+n)$
SUBDIVIDE	1	$O(m+n)$

Udfra skemat ovenfor er det nu muligt at give en worst-case kompleksitet for hele makrofasen i Yaps algoritme. Hvis k er antal gennemløb af løkken og hvis m og n er antal kontrol punkter i de to input kurver, så er den samlede kompleksitet givet ved

$$O(k(m+n)\log(m+n)).$$

En umiddelbar konsekvens af dette er, at siden de forventede værdier for m og n er relative små, i.e. under ti i praksis, så vil antal gennemløb af løkken, k, blive den dominante faktor, og som illustreres nedenfor, har denne størrelse en direkte korrelation med størrelsen af ϵ.

En empirisk undersøgelse

En empirisk undersøgelse af effektiviteten af Yaps skæringsalgoritme i praksis må begynde med at afprøve en given implementering af makrofasen på et helt almindeligt kurvepar. Hvis Yaps skæringsalgoritme skal være effektiv i praksis, må denne fase nødvendigvis også være det. Det følgende viser dette ikke tilfældet.

Analyse af ϵ værdiens størrelse

Analyse af størrelsen på den fastsatte ϵ værdi i Yaps algoritme giver et billede af, hvor lille en værdi, der er tale om, når specifikke type af kurver udgør input kurverne. Som det skal vise sig, afhænger denne ϵ værdi kun af nogle specifikke egenskaber for input kurverne.
Den fastsatte ϵ værdi i Yaps algoritme er givet som et minimum af fire specifikke separationsgrænser. Disse fire separationsgrænser er alle beregnet på baggrund af enkelte egenskaber for input kurverne. Værdien Δ_1 er en af disse separationsgrænser, og den værdi giver allerede et billede af størrelsesordnen af den nødvendige ϵ, så de

andre grænser analyseres ikke.

Lad være givet to Bézier kurver, hvis kontrol polygoner indeholder henholdsvis $m+1$ og $n+1$ kontrol punkter. Så er Δ_1 givet som følgende.

$$\Delta_1(m,n,a,b) = (2^{\frac{3}{2}}NK)^{-D}2^{-12m^2n^2}) \tag{6.2}$$

hvor

$$K = \max\{\sqrt{13}, 4ma, 4nb\}, \qquad N = \binom{3+2m+2n}{5}, \qquad D = m^2n^2(3+\frac{4}{m}+\frac{4}{n})$$

Som det kan ses, afhænger Δ_1 kun af a og b udover m og n. a og b er $\|\cdot\|_2$-normen til de implicitte algebraiske kurver. Som beskrevet tidligere er disse $\|\cdot\|_2$-normer ikke umiddelbart tilgængelige, så den øvre grænse for værdierne bruges istedet. Disse værdier er henholdsvis

$$a = (16^L 9^m)^m \qquad b = (16^L 9^n)^n$$

jævnfør sætning 3.3.2, når koordinaterne til kontrol polygonerne for F og G er givet ved (L,l)-bit flydende kommatal. Figur 6.1 og 6.2 nedenfor viser nogle konkrete værdier for Δ_1 beregnet ved hjælp af (6.2). Bemærk, at værdierne for Δ_1 ikke er eksakte, men blot overslag, som viser tendensen i udviklingen for varierende L og m. De enkelt overslag udgør den nærmeste potens af to, som er mindre end det eksakte udtryk.
Figur 6.1 viser, hvorledes Δ_1 udvikler sig for fastholdte m og n og varierende L. Værdierne for m og n er henholdsvis fire og tre. Bemærk, at værdierne for m og n er holdt små for bedre at kunne vise udviklingen af Δ_1 når L vokser.

L	1	2	4	8
$\Delta_1(m,n,a,b)$	2^{-67008}	2^{-78528}	$2^{-103872}$	$2^{-149952}$

Figur 6.1: $\Delta_1(m,n,a,b)$ for varierende L

Som det kan ses, aftager Δ_1 meget voldsomt selv med små tilvækster til L, og som værdien for $L=8$ viser, er der ingen grund til at forsætte beregningerne for større L. Observer, at L afspejler (bit-)kvaliteten af koordinaterne til kontrol punkterne for de involverede kurver. Det vil sige, at selv for kurver givet med koordinater af meget lav præcision, vil Δ_1 værdien være utrolig lille.

Figur 6.2 viser påvirkningen af et varierende m på Δ_1 værdien for fastholdt L og n. L er fastholdt på værdien fire. Idet et varierende m allerede viser en voldsomt aftagende tendens for Δ_1, fandtes det unødvendigt at variere n samtidig, så denne er fastholdt på værdien tre.

m	4	5	6	7
$\Delta_1(m,n,a,b)$	$2^{-103872}$	$2^{-209445}$	$2^{-382968}$	$2^{-647703}$

Figur 6.2: $\Delta_1(m,n,a,b)$ for varierende m

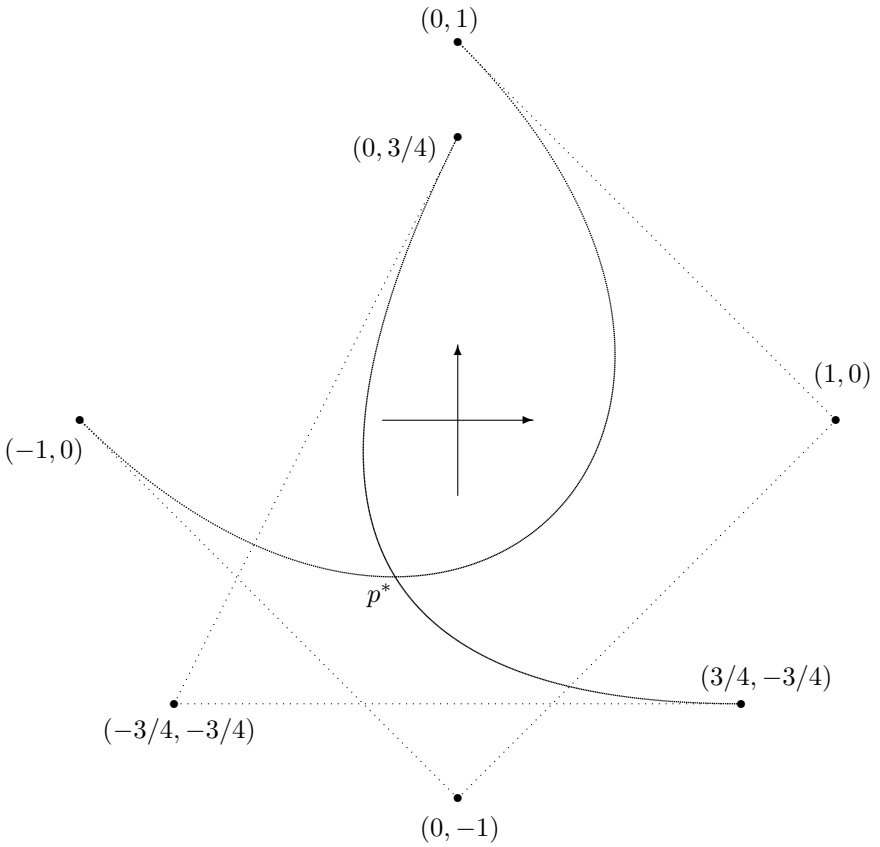
Første undersøgelse

Den første undersøgelse af min implementering af makrofasen i Yaps algoritme søger en sammenhæng mellem ϵ værdien og antal iterationer samt udførselstiden af makrofasen. Til dette formål introduceres et konkret kurvepar. Lad Bézier kurverne F og G være givet ved følgende to kontrol polygoner.

$$P(F) = [(-1,0),(0,-1),(1,0),(0,1)]$$

$$P(G) = [(0,3/4),(-3/4,-3/4),(3/4,-3/4)]$$

Kurverne er vist på figur 6.3, og som det fremgår på figuren, skærer kurverne hinanden i et entydigt skæringspunkt. Så umiddelbart skulle dette kurvepar udgøre et simpelt kurvepar i den forstand, at egenskaberne for kurverne, som bestemmer værdien af ϵ, er minimale. Graderne for kurverne er henholdsvis $m = 4$ og $n = 3$. Samtidig er koordinaterne givet ved få bits, så ϵ værdien vil for dette kurvepar antage en af de mindre værdier beregnet ovenfor.
Undersøgelsen består så i at afvikle makrofasen på dette kurvepar for varierende ϵ værdier og måle udførselstiden for de enkelte afviklinger. Figur 6.4 viser resultatet af denne undersøgelse. I figuren vises en tabel, der indeholder udførselstiderne for min implementering af makrofasen for forskellige værdier af ϵ. Første kolonne i tabellen viser de varierede ϵ værdier. Kolonne to viser antallet af iterationer i makrofasen, mens den sidste kolonne viser udførselstiden. I tabellen kan ses at udførselstiden bliver meget ringe for selv et simpelt kurvepar, når ϵ værdien bliver lille.



Figur 6.3: Et kandidatpar (F, G) som indeholder et entydigt skæringspunkt

Den ovenstående analyse af ϵ viser tydeligt, at værdierne bliver meget små. Så selv for lave værdier af L, m og n bliver ϵ en meget lille værdi. Hvis for eksempel ϵ

ϵ	Iterationer	Tid
2^{-100}	435	0.453
2^{-200}	835	1.078
2^{-300}	1231	2.078
2^{-400}	1629	3.531
2^{-500}	2033	5.578
2^{-1000}	4033	26.69
2^{-2000}	8021	146.3
2^{-4000}	16035	844.3

Figur 6.4: Udførselstider (sekunder) for varierende ϵ værdier

skal være mindre end $\Delta_1(4,3,8) = 2^{-149952}$, og hvis figur 6.4 viser tendensen for udførselstiden, kan man forvente at vente længe på et svar fra Yaps algoritme!
Analysen af størrelsesordenen af ϵ viser også, hvorfor det ikke vil give et bedre resultat at kigge på andre kurver. Udtrykket for Δ_1 givet i (6.2) afhænger nemlig kun af værdierne m, n og L. Bemærk, at ingen af værdierne siger noget om kurvernes egentlige forløb. De to første siger blot, at de respektive kontrol polygoner har henholdsvis $m+1$ og $n+1$ kontrol punkter, og den sidste værdi, L, giver kun et udtryk for, hvor meget præcision, koordinaterne til kontrol punkterne, er givet med. Det vil sige, der er ingen af disse størrelser, der påvirker kurvens billede. Der gælder selvfølgelig, at antallet af kontrol punkter, og den præcision som koordinaterne er givet med, påvirker kurvernes egenskaber, men det påvirker ikke dens forløb, da dette er givet ved de enkelte punkters position i forhold til hinanden.
Konsekvenserne af, at ϵ skal være mindre end Δ_1, og at denne størrelse kun afhænger af m,n og L, betyder, at to kurvepar med samme værdier af m,n og L vil have samme påvirkning af effektiviteten af Yaps algoritme, uafhængig af deres individuelle forløb og under antagelse af, at kurvene skærer hinanden. Så en empirisk undersøgelse af Yaps algoritmes effektivitet i praksis kan nøjes med at betragte en enkelt repræsentant for de kurvepar, som har ens m, n og L værdier. Men når værdierne i figurerne 6.1 og 6.2 viser en så kraftig aftagende tendens af Δ_1 værdierne, så er det næppe nødvendigt afprøve kurver med egenskaber givet ved store værdier for m, n og L, da værdien af Δ_1 allerede er ekstrem lille, selv for små værdier.

På figur 6.4 kan også læses, at størrelsen af ϵ er omvendt proportional med antal iterationer. Det vil sige, at en halvering af ϵ medfører en fordobling af antal iterationer. Det interessante er dog udførselstiden. Den er også omvendt proportional med størrelsen af ϵ. Det er dog kun for ϵ værdier større end 2^{-1000}, at der er tale om en faktor to. Når ϵ værdierne bliver mindre end 2^{-1000}, så stiger faktoren til fem. På grund af den allerede meget høje udførselstid for makrofasen for ϵ mindre end 2^{-4000} er det ikke undersøgt nærmere, om faktoren forbliver fem eller vokser yderligere, når ϵ nærmer sig værdier nær Δ_1. Denne stykvise omvendte proportionalitet skyldes formentlig, at beregningerne i Yaps algoritme er baseret på eksakt bigfloat aritmetik. De næste to undersøgelser af udførselstiden af makrofasen forsøger at give et empirisk bevis for denne påstand.

Anden undersøgelse

Den anden undersøgelse af min implementering af makrofasen går ud på at afgøre, hvorledes udførselstiden er fordelt under en afvikling af makrofasen. En sådan undersøgelse vil kunne vise, om en given delalgoritme, eller beregning, tager længere tid end de andre. Undersøgelsen er lavet på samme måde som den forrige, men nu måles udførselstiderne for de enkelte delalgoritmer. Figur 6.5 resultatet af denne undersøgelse. Der er tale om algoritmerne DIAM, CONPOLYINT, SUBDIVIDE og CONVEXHULL. Tiden i alt og antal iterationer er også med i oversigten for at give sammenhængen med figur 6.4.

ϵ	Iterationer	Tid i alt	DIAM	CONPOLYINT	SUBDIVIDE	CONVEXHULL
2^{-100}	435	0.453	0.000	0.377	0.015	0.061
2^{-200}	835	1.078	0.000	0.828	0.079	0.171
2^{-300}	1231	2.078	0.031	1.735	0.047	0.265
2^{-400}	1629	3.531	0.000	2.782	0.078	0.655
2^{-500}	2033	5.594	0.047	4.464	0.064	1.019
2^{-1000}	4033	26.66	0.079	21.23	0.186	5.158
2^{-2000}	8021	146.1	0.171	117.6	0.454	27.89
2^{-4000}	16035	844.1	0.420	677.8	1.496	164.3

Figur 6.5: Udførselstiden (sekunder) delt ud på de forskellige delalgoritmer i makrofasen

Resultaterne af denne undersøgelser viser, at det meste af beregningerne i makrofasen foregår primært et sted. Dette sted er CONPOLYINT algoritmen. Samtidig kan observeres, at den samlede udførselstid for CONPOLYINT udviser samme stykvise omvendte proportionale adfærd, som hele makrofasen i den forrige undersøgelse. Så i næste undersøgelse fokuseres på CONPOLYINT algoritmen, for at vise sammenhængen mellem den eksakte bigfloat aritmetik og den stykvise omvendte proportionale adfærd.

Tredje eksperiment

Den tredje og sidste undersøgelse forsøger at vise, hvorfor udførselstiden i de første to undersøgelser udviser en stykvis omvendt proportional adfærd. Undersøgelsen går ud på at måle den minimale, maksimale og gennemsnitlige udførselstid af CONPOLYINT algoritmen for varierende ϵ. Hvis disse målinger udviser den samme adfærd, som i de tidligere undersøgelser, kan det konkluderes, at det må være den eksakte bigfloat aritmetik, der forårsager den stykvise omvendte proportionale adfærd. Resultatet af undersøgelsen kan ses i figur 6.6. De følgende observationer kan gøres omkring talmaterialet.

1. Når ϵ halveres fordobles antallet af iterationer i makrofasen, hvilket også betyder en fordobling af kald til CONPOLYINT algoritmen.

2. Når ϵ halveres og er mindre end 2^{-1000}, så fordobles udførselstiden brugt i CONPOLYINT algoritmen. Når ϵ værdier mindre end eller lig 2^{-1000} halveres, så femdobles udførselstiden.

ϵ	Iterationer	Tid_{ialt}	Tid_{gns}	Tid_{min}	Tid_{max}
2^{-100}	435	0.400	$7.816 \cdot 10^{-4}$	0.000	0.016
2^{-200}	835	0.875	$1.049 \cdot 10^{-3}$	0.000	0.016
2^{-300}	1231	1.774	$1.441 \cdot 10^{-3}$	0.000	0.016
2^{-400}	1629	2.788	$1.711 \cdot 10^{-3}$	0.000	0.016
2^{-500}	2033	4.247	$2.089 \cdot 10^{-3}$	0.000	0.016
2^{-1000}	4033	19.48	$4.831 \cdot 10^{-3}$	0.000	0.062
2^{-2000}	8021	106.2	$1.324 \cdot 10^{-2}$	0.000	0.063
2^{-4000}	16035	644.4	$3.813 \cdot 10^{-2}$	0.000	0.141

Figur 6.6: Minimum, maksimum og gennemsnitlig udførselstid (sekunder) per afvikling af ConPolyInt for varierende ϵ

3. Den gennemsnitlige udførselstid for et kald til ConPolyInt algoritmen vokser monotont når ϵ aftager.

4. Den maksimale udførselstid for et kald til ConPolyInt algoritmen er konstant for ϵ større end 2^{-1000}. Ved ϵ lig 2^{-1000} fordobles den ovennævnte maksimale udførselstid, og det samme sker ved ϵ lig 2^{-4000}.

Udfra disse fire observationer konkluderes, at stigningen i den samlede udførselstid for makrofasen skyldes, at repræsentationen af bigfloat tallene, i koordinaterne til kontrol punkterne for kurverne, vokser i takt med, at ϵ værdien falder. Argumentet for denne konklusion følger. Fra observationerne 1,2 og 3 kan konkluderes, at en af to ting kan forårsage en ændring i adfærden af udførselstiden for makrofasen. Den ene kan være, at algoritme logikken pludselig degenererer for små ϵ, hvilket kunne indikeres af nogle høje udførselstider for enkelt udførsler af ConPolyInt algoritmen. Dette kan dog ses fra observation 3 ikke at være tilfældet, idet den gennemsnitlige udførselstid vokser monotont. Den anden mulighed er at de enkelte bigfloat beregninger begynder at tage længere tid i takt med at ϵ falder. En god indikator af, at dette er tilfældet, kan observeres i observation 4, der viser, at stigningen sker i diskrete spring. Dette indikerer, at repræsentationen af de involverede bigfloat tal udvides, og dermed gør beregningerne med disse langsommere.

Resultater

Som udgangspunkt undersøgtes, hvorvidt Yaps skæringsalgoritme kunne opnå sine mål omkring komplethed og eksakthed af sine løsninger. I afsnit 6.1 betragtedes Yaps skæringsalgoritme fra et teoretisk synspunkt. Det kunne konkluderes, at Yaps skæringsalgoritme var både komplet og eksakt. Kompletheden afhang af makrofasens evne til at opbryde kurverne i små nok kurvepar, som højst indeholder et skæringspunkt. Det gjorde, at der skulle beregnes et specifikt ϵ, som afhang af egenskaberne for input kurverne. Med hensyn til eksaktheden af skæringspunkterne, fundet af Yaps skæringsalgoritme, viste sig at afhænge af en opbrydning af alle beregninger i simple regneoperationer, som ved hjælp af bigfloat tal, kunne udføres eksakt. Opbrydningen viste sig at være mulig, og Yaps skæringsalgoritme er dermed også eksakt.

Effektiviteten i praksis er baseret på effektiviteten af henholdsvis makrofasen og mikrofasen. Den umiddelbare bekymring med hensyn til effektiviteten af makrofasen har været, at ϵ værdien ikke længere kunne sættes frit, men derimod afhang af egenskaberne for input kurverne. Disse egenskaber viste sig at føre til nogle meget små værdier af det fastsatte ϵ, se eventuelt figur 6.1 og 6.2. Dette medførte, at udførselstiden for selv simple kurvepar var enorm i praksis. Grunden til dette viste sig at være en marginal ændring i udførselstiden for de enkelte beregninger med bigfloat tal, som udføres mange gange på grund af et højt antal iterationer i makrofasen.

Kapitel 7

Konklusion

Formålet med dette speciale har været at evaluere en konkret implementering af Yaps skæringsalgoritme. Min oprindelige plan var at implementere den fulde skæringsalgoritme og undersøge, om den udgør en komplet løsning af skæringsproblemet, og samtidig er effektiv i praksis. Det vil sige, at jeg ønskede, at undersøge om den i praksis fandt alle skæringspunkter, og om disse var eksakte. En gennemgang af Yaps skæringsalgoritme viser, at den kan opdeles i to faser, henholdsvis makrofasen og mikrofasen, som udføres sekventielt. Makrofasens primære opgave består i at opdele kurverne i kandidatpar, der højst indeholder et entydigt skæringspunkt. Mikrofasen kan så for disse kandidatpar af- eller bekræfte, om der findes et skæringspunkt. Om en implementering er effektiv i praksis afhænger derfor af, om de individuelle faser er effektive. Men specielt gælder, at hvis ikke makrofasen er effektiv, vil Yaps skæringsalgoritme ikke kunne opnå at blive effektiv. Dette skyldes, at mikrofasen afhænger af, at makrofasen kan opdele input kurverne i kandidatpar, som højst indeholder et skæringspunkt. Mikrofasen kan derfor ikke blive færdig, før makrofasen er færdig.

Efter implementering af makrofasen i Yaps skæringsalgoritme viser de første empiriske målinger, at udførselstiden er meget afhængig af værdien af stop-kriteriet, ϵ, i makrofasen. For eksempel er udførselstiden for et simpelt kurvepar med et skæringspunkt 0.453 sekunder for $\epsilon = 2^{-100}$, men 844.3 sekunder for $\epsilon = 2^{-4000}$. Generelt viser undersøgelsen, at udførselstiden er stykvis omvendt proportional med ϵ værdiens størrelse. Det vil sige, når ϵ halveres i intervallet mellem 2^{-100} og 2^{-1000}, fordobles udførselstiden, mens hvis ϵ halveres i intervallet mellem 2^{-1000} og 2^{-4000}, femdobles udførselstiden. På grund af den meget høje udførselstid for ϵ værdier under 2^{-4000}, er udførselstiden ikke undersøgt for mindre ϵ værdier.
En nærmere undersøgelse af den stykvise omvendt proportionale tendens i udførselstiden viser sig at skyldes de eksakte bigfloat tal, som er nødvendige for at sikre en eksakt skæringsalgoritme. Det viser sig ved profilering af makrofasen, at proportionalitetsfaktoren mellem ϵ og udførselstiden vokser i diskrete spring i takt med, at ϵ aftager. Disse diskrete spring kan forklares med, at udførselstiden for de enkelte eksakte regneoperationer stiger i takt med det aftagende ϵ.

Umiddelbart vækker den stykvise omvendt proportionale tendens i udførselstiden ikke bekymring i forhold til effektiviteten af Yaps skæringsalgoritme, da jeg på dette tidspunkt ikke har undersøgt, hvor stor ϵ værdien reelt skal være for at sikre, at et kandidatpar højst kan indeholde et entydigt skæringspunkt. Men viser sig, at der er grund til bekymring. Værdierne for stop-kriteriet i makrofasen viser sig nemlig, for selv de mest simple kurver, at være mindre end 2^{-64000}. Dette vil, med den ovennævnte stykvise omvendt proportionale tendens, betyde, at udførselstiden

for makrofasen kan forventes at være større end 527687.5 sekunder, hvilket svarer til cirka 146 timer. Med udførselstider af denne størrelsesorden vil makrofasen, og dermed den fulde skæringsalgoritme, aldrig opnå effektivitet i praksis.

En undersøgelse af, hvordan stop-kriteriet i makrofasen skal beregnes, viser, at ϵ værdiens størrelse kun afhænger af nogle få algebraiske egenskaber for input kurverne. Disse egenskaber udgør antal kontrol punkter i de respektive kontrol polygoner, $\|\cdot\|_2$-normen af de implicitte algebraiske kurver samt bitkvaliteten af koordinaterne til kontrol punkterne. Ved nærmere eftersyn af formlerne for de enkelt geometriske separationsgrænser, som ligger til grund for ϵ værdien, bemærkes, at selv for minimale input værdier i disse formler bliver ϵ mindre end 2^{-10000}. En anden konsekvens af måden, hvorpå ϵ beregnes, er, at hvis to kandidatpar har samme algebraiske egenskaber, i.e. antal kontrol punkter, $\|\cdot\|_2$ og bitkvalitet, så vil ϵ have samme størrelse.

Alt i alt kan konkluderes, at Yaps algoritme i teorien udgør en komplet og eksakt skæringsalgoritme. Men effektiviteten i praksis kan ikke opnås. Årsagen er, at mikrofasen forventer, at de kandidatpar, som den modtager fra makrofasen højst indeholder et entydigt skæringspunkt. Dette er nødvendigt for at kunne udføre kobling processen. Spørgsmålet er, om noget kan gøres for at øge effektiviteten i Yaps skæringsalgoritme i praksis. Umiddelbart ser jeg ikke nogen anden mulighed end at forbedre de geometriske separationsgrænser, som ligger til grund for størrelsen af stop-kriteriet i makrofasen. Dette udgør et rent matematisk problem, som jeg vil overlade til fremtidens matematikere.

Bilag A

Kode - Geometriske primitiver

Det følgende er source-koden til de geometriske primitiver som anvendes i Yaps algoritme. Der er tale om tre primitiver, henholdsvis Bézier kurver, polygoner og punkter.

```
namespace Geometry
{
  struct Point
  {
    CORE::BigFloat x, y;
    Point();
    Point( const CORE::BigFloat& _x, const CORE::BigFloat& _y );
    Point( int _x, int _y );
    Point( const Point& p );
    Point& operator=( const Point& p )
    {
      if ( this != &p )
      {
        x = p.x;
        y = p.y;
      }
      return *this;
    }
  };

  class Polygon
  {
  public:
    explicit Polygon( unsigned int size=0, Point* pts=0 );
    Polygon( const Polygon& p );
    ~Polygon();
    Polygon* Polygon::clone() const;
    Point& operator[]( unsigned int i ) { return m_pts[i]; }
    const Point& operator[]( unsigned int i ) const { return m_pts[i]; }
    unsigned int getSize() const { return m_size; }
    const Point* getData() const { return m_pts; };
  private:
    unsigned int m_size;
    Point* m_pts;
    // Reference Counting:
    //  * DO NOT CALL THESE METHODS OR CHANGE THE DATA-MEMBER
    //  * The methods and data-member are used by the smartpointer
    //    class.
    friend class SmartPointer<Polygon>;
    void addRef() const { ++m_refs; }
    void releaseRef() const { if ( --m_refs == 0 ) delete this; }
   mutable unsigned int m_refs;
  };

  class BezierCurve
  {
  public:
    BezierCurve( unsigned int size, Point* pts,
      const RasterColor& color = RasterColor( 0.0, 0.0, 1.0 ) );
```

```
    ~BezierCurve();
    BezierCurve* clone() const;
    unsigned int getDegree() const;
    const Polygon& getConvexHull() const;
    const Polygon& getControlPolygon() const;
    const RasterColor& getColor() const { return m_color; }
  private:
    Polygon* m_cp;
    mutable Polygon* m_ch;
    RasterColor m_color;

    // Reference Counting:
    //  * DO NOT CALL THESE METHODS OR CHANGE THE DATA-MEMBER
    //  * The methods and data-member are used by the smartpointer
    //    class.
    friend class SmartPointer<BezierCurve>;
    void addRef() const { ++m_refs; }
    void releaseRef() const { if ( --m_refs == 0 ) delete this; }
    mutable unsigned int m_refs;
  };
}
```

Bilag B

Kode - Makrofasen

Det følgende er min implementering af makrofasen i Yaps skæringsalgoritme. Bézier kurver, polygoner og punkter er givet appendiks A. Makrofasen kan opbrydes i et antal mindre delalgoritmer, heriblandt en konveks hylster algoritme. Source-koden til disse delalgoritmer kan findes i appendiks C.

```
bool Algorithm::intersectBezier( const BezierCurve& b0, const BezierCurve& b1, std::ostream& file  )
{
  SmartPointer<BezierCurve> F = b0.clone();
  SmartPointer<BezierCurve> G = b1.clone();
  m_macroQ.clear();
  m_microQ.clear();
  m_macroQ.push( CurvePair( F, G ) );
  unsigned int k=0;
  while ( !m_macroQ.empty() )
  {
    ++k;
    CurvePair S = m_macroQ.pop();
    SmartPointer<Polygon> I = intersection( S.first->getConvexHull(), S.second->getConvexHull() );
    if ( I->getSize() <= 0 || diameter( *I ) == BigFloat::getZero() )
      continue;
    SmartPointer<Polygon> II = merge( S.first->getControlPolygon(), S.second->getControlPolygon() );
    const BigFloat d = diameter( *II );
    if ( d < m_deltaStar )
    {
      m_microQ.push( S );
    }
    else
    {
      const BigFloat d1 = diameter( S.first->getControlPolygon() );
      const BigFloat d2 = diameter( S.second->getControlPolygon() );
      if ( d1 > d2 )
      {
        CurvePair N = subdivide( *S.first, 0.5 );
        m_macroQ.push( CurvePair( N.second, S.second ) );
        m_macroQ.push( CurvePair( N.first, S.second ) );
      }
      else
      {
        CurvePair N = subdivide( *S.second, 0.5 );
        m_macroQ.push( CurvePair( N.second, S.first ) );
        m_macroQ.push( CurvePair( N.first, S.first ) );
      }
    }
  }
  return !m_microQ.empty();
}
```

Bilag C

Kode - Geometriske funktioner

Det følgende er source-koden til de geometriske operationer som bruges i Yaps skæringsalgoritme. Disse operationer inkluderer afstandsfunktioner, determinanter, konveks hylster og skæring mellem konvekse polygoner.

```
#include "Misc.h"
#include "Geometry.h"
#include "Defs.h"

using namespace Geometry;

const BigFloat Geometry::distance( const Point& p, const Point& q )
{
  BigFloat value = sqrt( (q.x - p.x)*(q.x - p.x) + (q.y - p.y)*(q.y - p.y) );
  value.makeExact();
  return value;
}

const BigFloat Geometry::area( const Point& p, const Point& q, const Point& r )
{
  return -q.x * p.y + r.x * p.y + p.x * q.y - r.x * q.y - p.x * r.y + q.x * r.y;
}

bool Geometry::left( const Point& p, const Point& q, const Point& r )
{
  return ( area( p, q, r ) > 0 );
}

bool Geometry::lefton( const Point& p, const Point& q, const Point& r )
{
  return !( area( p, q, r ) < 0 );
}

bool Geometry::colinear( const Point& p, const Point& q, const Point& r )
{
  return ( area( p, q, r ) == 0 );
}

bool Geometry::pointInsidePolygon( const Polygon& Q, const Point& p )
{
  const unsigned int sizeQ = Q.getSize();
  const Point* data = Q.getData();
  switch ( sizeQ )
  {
  case 0:
    return false;
  case 1:
    return ( p.x == data[0].x ) && ( p.y == data[0].y );
  default:
    for ( unsigned int i=0; i<sizeQ; ++i )
    {
      if ( !lefton( data[i], data[(i+1)%sizeQ], p ) )
        return false;
```

```
    }
    return true;
  }
}

const Geometry::LexResult Geometry::lexCompare( const Point& p, const Point& q )
{
  if ( p.x < q.x ) return lrLESS;
  if ( p.x > q.x ) return lrGREATER;
  if ( p.y < q.y ) return lrLESS;
  if ( p.y > q.y ) return lrGREATER;
  return lrEQUAL;
}

bool Geometry::isConvex( const Polygon& p )
{
  unsigned int size=p.getSize();
  const Point* data=p.getData();

  for ( unsigned int i=0; i<size; ++i )
  {
    unsigned int j = (i+1) % size;
    unsigned int k = (i+2) % size;
    if ( !lefton( data[i], data[j], data[k] ) )
      return false;
   }
   return true;
}

namespace
{
  bool operator<=( const Geometry::Point& p, const Geometry::Point& q )
  {
    return ( lexCompare( p, q ) <= 0 );
  }
}

Geometry::Polygon* Geometry::convexHull( unsigned int size, const Point* pts )
{
  switch ( size )
  {
    case 2:
      if ( lexCompare( pts[0], pts[1] ) != 0 )
      {
        Point* newpts = new Point[2];
        newpts[0] = pts[0];
        newpts[1] = pts[1];
        return new Polygon( 2, newpts );
      }
      // else case 1!!!
    case 1:
      {
        Point* newpts = new Point[1];
        newpts[0] = pts[0];
        return new Polygon( 1, newpts );
      }
case 0:
      return new Polygon();
  }

  const Point** sortedpts = new const Point*[size];
  for ( unsigned int i=0; i<size; ++i )
    sortedpts[i] = &pts[i];
  quicksort<const Point>( sortedpts, 0, size-1 );

  DoublyLinkedList<const Point*> Lupper;

  Lupper.pushBack( sortedpts[0] );
  Lupper.pushBack( sortedpts[1] );

  for ( unsigned int i=2; i<size; ++i )
  {
    Lupper.pushBack( sortedpts[i] );
    while ( ( Lupper.getSize() > 2 ) && !left( *Lupper.getBack()->prev->prev->data, *Lupper.getBack()->prev->data, *Lupper.getBa
    {
      Link<const Point*>* link = Lupper.getBack()->prev;
      Lupper.removeLink( link );
    }
  }
```

```
  DoublyLinkedList<const Point*> Llower;

  Llower.pushBack( sortedpts[size-1] );
  Llower.pushBack( sortedpts[size-2] );

  for ( unsigned int i=3; i<=size; ++i )
  {
    Llower.pushBack( sortedpts[size-i] );
    while ( ( Llower.getSize() > 2 ) && !left( *Llower.getBack()->prev->prev->data, *Llower.getBack()->prev->data, *Llower.getBa
    {
      Link<const Point*>* link = Llower.getBack()->prev;
      Llower.removeLink( link );
    }
  }

  Llower.popFront();
  Llower.popBack();

  if ( sortedpts != 0 )
    delete[] sortedpts;

  const unsigned newsize = Lupper.getSize() + Llower.getSize();

  Point* newpts = new Point[newsize];
  unsigned int i=0;
  const Point* p = Lupper.popFront();
  while ( p != 0 )
  {
    newpts[i++] = *p;
    p = Lupper.popFront();
  }

  p = Llower.popFront();
  while ( p != 0 )
  {
    newpts[i++] = *p;
    p = Llower.popFront();
  }
  return new Polygon( newsize, newpts );
}

Geometry::Polygon* Geometry::convexHull( const Polygon& poly )
{
  return convexHull( poly.getSize(), poly.getData() );
}

const BigFloat Geometry::diameter( const Polygon& poly )
{
  const unsigned int size=poly.getSize();
  const Point* data = poly.getData();

  BigFloat xmin = data[0].x;
  BigFloat xmax = data[0].y;
  BigFloat ymin = data[0].x;
  BigFloat ymax = data[0].y;

  for ( unsigned int i=1; i<size; ++i )
  {
    if ( data[i].x < xmin )
      xmin = data[i].x;
    else if ( data[i].x > xmax )
      xmax = data[i].x;

  if ( data[i].y < ymin )
    ymin = data[i].y;
  else if ( data[i].y > ymax )
    ymax = data[i].y;
  }

  return CORE::core_max( xmax-xmin, ymax-ymin );
}

namespace
{
  bool Between( const Geometry::Point& a, const Geometry::Point& b, const Geometry::Point& c )
  {
    if ( a.x != b.x )
      return ( ( a.x <= c.x ) && ( c.x <= b.x ) ) || ( ( a.x >= c.x ) && ( c.x >= b.x ) );
    else if ( a.y != b.y )
      return ( ( a.y <= c.y ) && ( c.y <= b.y ) ) || ( ( a.y >= c.y ) && ( c.y >= b.y ) );
```

```
  else
    return ( c.x == ( ( a.x + b.x ) / 2 ) ) &&
           ( c.y == ( ( a.y + b.y ) / 2 ) );
}

char ParallelInt( const Geometry::Point& a, const Geometry::Point& b, const Geometry::Point& c, const Geometry::Point& d, Geom
{
  if ( !colinear( a, b, c ) )
    return '0';

  if ( Between( a, b, c ) && Between( a, b, d ) )
  {
    p = c;
    q = d;
    return 'e';
  }
  if ( Between( c, d, a ) && Between( c, d, b ) )
  {
    p = a;
    q = b;
    return 'e';
  }
  if ( Between( a, b, c ) && Between( c, d, b ) )
  {
    p = c;
    q = b;
    return 'e';
  }
  if ( Between( a, b, c ) && Between( c, d, a ) )
  {
    p = c;
    q = a;
    return 'e';
  }
  if ( Between( a, b, d ) && Between( c, d, b ) )
  {
    p = d;
    q = b;
    return 'e';
  }
  if ( Between( a, b, d ) && Between( c, d, a ) )
  {
    p = d;
    q = q;
    return 'e';
  }
  return '0';
}

char SegSegInt( const Geometry::Point& a, const Geometry::Point& b, const Geometry::Point& c, const Geometry::Point& d, Geomet
{
  const BigFloat denom =
  a.x * ( d.y - c.y ) +
  b.x * ( c.y - d.y ) +
  d.x * ( b.y - a.y ) +
  c.x * ( a.y - b.y );

  if ( denom == 0.0 )
    return ParallelInt( a, b, c, d, p, q );

  // b = d
  if ( b.x == d.x && b.y  == d.y )
    return '0';

  // b = c
  if ( b.x == c.x && b.y == c.y )
    return '0';

  // d = a
  if ( d.x == a.x && d.y == a.y )
    return '0';

  char code = '?';

  BigFloat num =
  a.x * ( d.y - c.y ) +
  c.x * ( a.y - d.y ) +
  d.x * ( c.y - a.y );

  BigFloat s;
```

```
  if ( num == BigFloat::getZero() )
    s = BigFloat::getZero();
  else if ( num == denom )
    s = BigFloat::getOne();
  else
  {
    s = num / denom;
    s.makeExact();
  }

  num = -(
    a.x * ( c.y - b.y ) +
    b.x * ( a.y - c.y ) +
    c.x * ( b.y - a.y ) );

  BigFloat t;
  if ( num == BigFloat::getZero() )
    t = BigFloat::getZero();
  else if ( num == denom )
    t = BigFloat::getOne();
  else
  {
    t = num / denom;
    t.makeExact();
  }

  const BigFloat& zero = BigFloat::getZero();
  const BigFloat& one = BigFloat::getOne();

  if ( ( s > 0 ) && ( s < one ) && ( t > zero ) && ( t < one ) )
    code = '1';
  else if ( ( s < zero ) || ( s > one ) || ( t < zero ) || ( t > one ) )
    code = '0';
  else
    code = 'v';

  p.x = a.x + s * ( b.x - a.x );
  p.y = a.y + s * ( b.y - a.y );

  return code;
}


Geometry::Polygon* intersect_EE( const Geometry::Polygon& P, const Geometry::Polygon& Q )
{
  return new Geometry::Polygon();
}

Geometry::Polygon* intersect_PP( const Geometry::Polygon& P, const Geometry::Polygon& Q )
{
  if ( ( P[0].x == Q[0].x ) && ( P[0].y == Q[0].y ) )
  {
    Geometry::Point* newpts = new Geometry::Point[1];
    newpts[0] = P[0];
    return new Geometry::Polygon( 1, newpts );
  }
  else return new Geometry::Polygon();
}

Geometry::Polygon* intersect_PA( const Geometry::Polygon& P, const Geometry::Polygon& Q )
{
  if ( Geometry::pointInsidePolygon( P, Q[0] ) )
  {
    Geometry::Point* newpts = new Geometry::Point[1];
    newpts[0] = Q[0];
    return new Geometry::Polygon( 1, newpts );
  }
  else return new Geometry::Polygon();
}

Geometry::Polygon* intersect_LL( const Geometry::Polygon& P, const Geometry::Polygon& Q )
{
  static Geometry::Point p;
  static Geometry::Point q;

  char code = SegSegInt( P[0], P[1], Q[0], Q[1], p, q );

  if ( code == '1' || code == 'v' )
  {
    Geometry::Point* newpts = new Geometry::Point[1];
```

```
      newpts[0] = p;
      return new Geometry::Polygon( 1, newpts );
    }
    else if ( code == 'e' )
    {
      Geometry::Point* newpts = new Geometry::Point[1];
      newpts[0] = p;
      newpts[1] = q;
      return new Geometry::Polygon( 2, newpts );
    }
    else return new Geometry::Polygon();
  }

  Geometry::Polygon* intersect_LA( const Geometry::Polygon& P, const Geometry::Polygon& Q )
  {
    if ( pointInsidePolygon( Q, P[0] ) )
    {
      // P[0] inside Q
      if ( pointInsidePolygon( Q, P[1] ) )
      {
        // P[0] and P[1] inside Q
        Geometry::Point* newpts = new Geometry::Point[2];
        newpts[0] = P[0];
        newpts[1] = P[1];
        return new Geometry::Polygon( 2, newpts );
      }
      else
      {
        // P[0] inside, P[1] outside Q
        Geometry::Point* newpts = new Geometry::Point[2];

        const unsigned int sizeQ = Q.getSize();
        for ( unsigned int i=0; i<sizeQ; ++i )
        {
          Geometry::Point p, q;
          char code = SegSegInt( P[0], P[1], Q[i], Q[(i+1)%sizeQ], p, q );
          if ( code == '1' || code == 'v' )
          {
            newpts[1] = P[0];
            newpts[0] = p;
            break;
          }
          else if ( code = 'e' )
          {
            newpts[1] = p;
            newpts[0] = q;
            break;
          }
        }

return new Geometry::Polygon( 2, newpts );
      }
    }
    else
    {
      // P[0] outside Q
      if ( pointInsidePolygon( Q, P[1] ) )
      {
        // P[0] outside, P[1] inside Q
        Geometry::Point* newpts = new Geometry::Point[2];
        const unsigned int sizeQ = Q.getSize();
        for ( unsigned int i=0; i<sizeQ; ++i )
        {
          Geometry::Point p, q;
          char code = SegSegInt( P[0], P[1], Q[i], Q[(i+1)%sizeQ], p, q );
          if ( code == '1' || code == 'v' )
          {
            newpts[1] = P[1];
            newpts[0] = p;
            break;
          }
          else if ( code = 'e' )
          {
            newpts[1] = p;
            newpts[0] = q;
            break;
          }
        }
        return new Geometry::Polygon( 2, newpts );
      }
```

```
    else
    {
      // P[0], P[1] outside Q
      Geometry::Point pts[2];
      Geometry::Point p, q;
      unsigned int j=0;
      const unsigned int sizeQ = Q.getSize();
      for ( unsigned int i=0; i<sizeQ; ++i )
      {
        char code = SegSegInt( P[0], P[1], Q[i], Q[(i+1)%sizeQ], p, q );
        if ( code == '1' || code == 'v' )
        {
          pts[j] = p;
          if ( j > 1 )
            break;
          ++j;
        }
        else if ( code = 'e' )
       {
          Geometry::Point* newpts = new Geometry::Point[2];
          newpts[0] = p;
          newpts[1] = q;
          return new Geometry::Polygon( 2, newpts );
        }
      }
      if ( j > 0 )
      {
        Geometry::Point* newpts = new Geometry::Point[j];
        if ( j == 1 )
        {
          newpts[0] = pts[0];
        }
        else if ( j == 2 )
        {
          newpts[0] = pts[0];
          newpts[1] = pts[1];
        }
        return new Geometry::Polygon( j, newpts );
        }
        return new Geometry::Polygon();
    }
  }
}

Geometry::Polygon* intersect_AA( const Geometry::Polygon& P, const Geometry::Polygon& Q )
{
  static const Geometry::Point Origin( BigFloat::getZero(), BigFloat::getZero() );

  const unsigned int n = P.getSize();
  const unsigned int m = Q.getSize();

  unsigned int a = 0;
  unsigned int b = 0;
  unsigned int aa = 0;
  unsigned int ba = 0;

  enum { Pin, Qin, Unknown } inflag = Unknown;
  Geometry::Point p, q;
  DoublyLinkedList<Geometry::Point> list;
  unsigned int iterations = 0;
  do
  {
    unsigned int a1 = ( a + n - 1 ) % n;
    unsigned int b1 = ( b + m - 1 ) % m;

    Geometry::Point A( P[a].x - P[a1].x, P[a].y - P[a1].y );
    Geometry::Point B( Q[b].x - Q[b1].x, Q[b].y - Q[b1].y );

    const BigFloat cross = Geometry::area( Origin, A, B );
    const BigFloat aHB = Geometry::area( Q[b1], Q[b], P[a] );
    const BigFloat bHA = Geometry::area( P[a1], P[a], Q[b] );

    char code = SegSegInt( P[a1], P[a], Q[b1], Q[b], p, q );

    if ( code == '1' || code == 'v' )
    {
      if ( inflag == Unknown )
        aa = ba = 0;

      if ( code != 'v' )
```

```
      list.pushBack( p );

    if ( bHA != BigFloat::getZero() && aHB > BigFloat::getZero() )
      inflag = Pin;
    else if ( aHB != BigFloat::getZero() && bHA > BigFloat::getZero() )
      inflag = Qin;
  }

  if ( ( code == 'e' ) && ( ( A.x * B.x + A.y * B.y ) < BigFloat::getZero() ) )
  {
    // De to polygoner deler kun en kant( eller de deraf )
    // Output polygon som består af den fundne edge
    Geometry::Point* newpts = new Geometry::Point[2];
    newpts[0] = p;
    newpts[1] = q;
    return new Geometry::Polygon( 2, newpts );
  }

  if ( cross == BigFloat::getZero() && aHB < BigFloat::getZero() && bHA < BigFloat::getZero() )
  {
    // Polygonerne skære ikke hinanden
    return new Geometry::Polygon();
  }
  else if ( cross == BigFloat::getZero() && aHB == BigFloat::getZero() && bHA == BigFloat::getZero() )
  {
    if ( inflag == Pin )
    {
      ++ba;
      b = (b+1) % m;
    }
    else
    {
      ++aa;
      a = (a+1) % n;
    }
  }
  else if ( cross > BigFloat::getZero() )
  {
    if ( bHA > BigFloat::getZero() )
    {
      if ( inflag == Pin )
        list.pushBack( P[a] );
      ++aa;
      a = (a+1) % n;
    }
    else
    {
      if ( inflag == Qin )
        list.pushBack( Q[b] );
      ++ba;
      b = (b+1) % m;
    }
  }
  else // if ( cross < BigFloat::getZero() )
  {
    if ( aHB > BigFloat::getZero() )
    {
      if ( inflag == Qin )
        list.pushBack( Q[b] );
      ++ba;
      b = (b+1) % m;
    }
    else
    {
      if ( inflag == Pin )
        list.pushBack( P[a] );
      ++aa;
      a = (a+1) % n;
    }
  }

} while ( ( ( aa < n ) || ( ba < m ) ) && ( aa < 2*n ) && ( ba < 2*m ) );

if ( inflag == Unknown || list.getSize() <= 0 )
{
  if ( P.getSize() > 0 && Geometry::pointInsidePolygon( Q, P[0] ) )
    return P.clone();
  else if ( Q.getSize() > 0 && Geometry::pointInsidePolygon( P, Q[0] ) )
    return Q.clone();
  else
```

```
        return new Geometry::Polygon();
      }
      else
      {
        const unsigned int size = list.getSize();
        Geometry::Point* newpts = new Geometry::Point[size];

        unsigned int i=0;
        while ( list.getSize() > 0 )
        newpts[i++] = list.popFront();

        return new Geometry::Polygon( size, newpts );
      }
    }
  }


Geometry::Polygon* Geometry::intersection( const Polygon& P, const Polygon& Q )
{
  typedef Polygon* (*IntersectFunc)( const Polygon&, const Polygon& );
  static const IntersectFunc func[16] =
  {
    &intersect_EE,    &intersect_EE,    &intersect_EE,    &intersect_EE,
    &intersect_EE,    &intersect_PP,    &intersect_PA,    &intersect_PA,
    &intersect_EE,    &intersect_PA,    &intersect_LL,    &intersect_LA,
    &intersect_EE,    &intersect_PA,    &intersect_LA,    &intersect_AA
  };

  const unsigned int sizeP = std::min<unsigned int>( P.getSize(), 3 );
  const unsigned int sizeQ = std::min<unsigned int>( Q.getSize(), 3 );

  if ( sizeP == 0 || sizeQ == 0 )
    return new Polygon();
  Geometry::Polygon* ppp = 0;
  if ( sizeP <= sizeQ )
    ppp = func[sizeP*4 + sizeQ]( P, Q );
  else
    ppp = func[sizeP*4 + sizeQ]( Q, P );
  return ppp;
}

const Point Geometry::evaluateHORNER( const Polygon& P, const BigFloat& t )
{
  const unsigned int d = P.getSize() - 1;
  const Point* c = P.getData();

  BigFloat t1 = BigFloat::getOne() - t;
  int n_choose_i = 1;

  Point aux( c[0].x * t1, c[0].y * t1 );

  BigFloat fact = BigFloat::getOne();
  for ( unsigned int i=1; i<d; ++i )
  {
    fact *= t;
    n_choose_i = n_choose_i * (d-i+1) / i;
    aux.x = ( aux.x + fact * n_choose_i * c[i].x ) * t1;
    aux.y = ( aux.y + fact * n_choose_i * c[i].y ) * t1;
  }

  aux.x = aux.x + ( fact * t ) * c[d].x;
  aux.y = aux.y + ( fact * t ) * c[d].y;

  return aux;
}


CurvePair Geometry::subdivide( const BezierCurve& B, const BigFloat& t )
{
  const unsigned int degree = B.getDegree();
  BigFloat t1 = BigFloat::getOne() - t;

  // right subcurve
  const Polygon& BB = B.getControlPolygon();

  Point* right = new Point[degree+1];
  for ( unsigned int i=0; i<=degree; ++i )
    right[i] = BB[i];

  for ( unsigned int r=1; r<=degree; ++r )
```

```
    for ( unsigned int i=0; i<=degree-r; ++i )
    {
      right[i].x = t1 * right[i].x + t * right[i+1].x;
      right[i].y = t1 * right[i].y + t * right[i+1].y;
    }

  // left subcurve
  Point* left = new Point[degree+1];

  for ( unsigned int i=0; i<=degree; ++i )
    left[degree-i] = BB[i];

  for ( unsigned int r=1; r<=degree; ++r )
    for ( unsigned int i=0; i<=degree-r; ++i )
    {
      left[i].x = t * left[i].x + t1 * left[i+1].x;
      left[i].y = t * left[i].y + t1 * left[i+1].y;
    }

  return CurvePair( new BezierCurve( degree+1, left, B.getColor() ), new BezierCurve( degree+1, right, B.getColor() ) );
}

Polygon* Geometry::merge( const Polygon& P, const Polygon& Q )
{
  const unsigned int sizeP = P.getSize();
  const unsigned int sizeQ = Q.getSize();

  if ( sizeP + sizeQ > 0 )
  {
    Point* newpts  = new Point[sizeP+sizeQ];

    unsigned int j=0;
    for ( unsigned int i=0; i<sizeP; ++i )
      newpts[j++] = P[i];
    for ( unsigned int i=0; i<sizeQ; ++i )
      newpts[j++] = Q[i];
    SmartPointer<Polygon> M = new Polygon( sizeP + sizeQ, newpts );
    return convexHull( M->getSize(), M->getData() );
  }
  else return new Polygon();
}
```

Litteratur

[1] N.M. Patrikalakis and Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer Verlag, Heidelberg, Germany, 2002.

[2] Younis Hijazi. Arrangements of planar curves. In *H. Hagen, A. Kerren, P. Dannenmann (Eds.), Visualization of Large and Unstructured Data Sets Proceedings of the first workshop of DFG*, 2006.

[3] G. Farin. *Curves and surfaces for CAGD : a practical guide*. Morgan Kaufmann, 2002.

[4] Chee K. Yap. Complete subdivision algorithms, i: intersection of bezier curves. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 217–226, New York, NY, USA, 2006. ACM Press.

[5] Chee K. Yap. Complete subdivision algorithms, i: intersection of bezier curve. *ftp://cs.nyu.edu/pub/local/yap/exact/bezier1.ps.gz*, 2006.

[6] M. Overmars M. de Berg, M. van Kreveld and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer, second edition, 2000.

[7] C.K. Yap. *Fundamental Problems of Algoritmic Algebra*. Oxford University Press, Inc., 2000.

[8] M.-S. Kim and I.-K. Lee. Gaussian approximation of objects bounded by algebraic curves. In *Proc. 1990 IEEE Int'l, Conf. on Robotics and Automation*, pages 322–326, 1990.

[9] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, second edition, 1994.