Fast high-resolution drawing of algebraic curves and surfaces

Nuwan Herath Mudiyanselage

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Fast high-resolution drawing of algebraic curves and surfaces

THÈSE

présentée et soutenue publiquement le 2 juin 2023

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Nuwan Herath Mudiyanselage

Composition du jury

Présidente :	Marie-Odile Berger	Directrice de recherche, Inria Nancy, Équipe Tangram, Loria
Rapporteurs :	Stef Graillat	Professeur, Sorbonne Université, LIP6
	Michael Sagraloff	Professeur, HAW Landshut, Fakultät Informatik
Examinateurs :	Guillaume Moroz	Chargé de recherche, Inria Nancy, Équipe Gamble, Loria
	Marc Pouget	Chargé de recherche, Inria Nancy, Équipe Gamble, Loria
	Nathalie Revol	Chargée de recherche, Inria Lyon, LIP, ENS de Lyon

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Résumé

La visualisation scientifique permet aux utilisateurs de développer une intuition sur leurs données et de les comprendre. Elle a de nombreuses applications : modélisation pour les simulations, conception de mécanismes, imagerie médicale... Nous abordons le problème de la visualisation de courbes et de surfaces algébriques implicites, qui sont solutions d'une équation polynomiale $P(x, y) = 0$ ou $Q(x, y, z) = 0$. Plus précisément, nous traitons le problème du tracé de courbes ou de surfaces de haut degré en haute résolution. Dans ce cas, la plupart des approches de l'état de l'art ne parviennent pas à produire des dessins en un temps raisonnable en raison du coût élevé de l'évaluation polynomiale.

Notre principale contribution consiste à combiner des algorithmes standards de visualisation issus de l'infographie avec des méthodes d'évaluation multipoints issues du calcul formel. Plus précisément, nous utilisons la transformée en cosinus discrète (DCT), qui peut être calculée efficacement avec l'algorithme de transformée de Fourier rapide (FFT). Dans la plupart de nos algorithmes, nous avons combiné cette idée avec un processus de subdivision classique afin de réduire le nombre d'évaluations.

En utilisant un calcul exact de borne d'erreur et l'arithmétique des intervalles, nous proposons de nouveaux algorithmes qui produisent des dessins certifiés. Nous les comparons expérimentalement sur deux classes de polynômes de haut degré. En particulier, certaines de ces approches sont plus rapides que les logiciels de l'état de l'art.

Mots-clés: dessin de courbe, courbe algébrique de haut degré, transformée en cosinus discrète, analyse d'erreur numérique, évaluation multipoint rapide.

Abstract

Scientific visualization allows users to build an intuition and to get an understanding of their data. Its applications are numerous: modeling for simulations, mechanism design, medical imaging... We address the problem of visualizing implicit algebraic plane curves and surfaces, that are solutions of a polynomial equation $P(x, y) = 0$ or $Q(x, y, z) = 0$. More specifically, we handle the problem of drawing high degree curves or surfaces at a high resolution. In this case, most state-of-the-art approaches fail to produce drawings in a reasonable time due to the high evaluation cost of the polynomial.

Our main contribution is to combine standard visualization algorithms from computer graphics with multipoint evaluation methods from computer algebra. More precisely, we use the fast Discrete Cosine Transform (DCT), which can be computed efficiently with the Fast Fourier Transform (FFT) algorithm. In most of our algorithms, we have combined that idea with a classical subdivision process in order to reduce the number of evaluations.

Using exact error bound computation and interval arithmetic, we propose new algorithms which produce certified drawings. We compare them experimentally on two classes of high degree polynomials. Notably, some of those approaches are faster than state-of-the-art drawing software.

Keywords: curve drawing, high degree algebraic curve, discrete cosine transform, numerical error analysis, fast multipoint evaluation.

Remerciements

Je me dois d'abord de remercier celles et ceux qui ont pris de leur temps pour lire et évaluer mon travail, et qui ont accepté de me considérer comme l'un de leurs pairs. Je leur en suis grandement reconnaissant. Je remercie d'abord Stef Graillat et Michael Sagraloff pour le soin approfondi avec lequel ils ont examiné mon travail. Je voudrais aussi remercier Marie-Odile Berger pour avoir bien voulu endosser la responsabilité de présidente de jury et Nathalie Revol pour sa bienveillance et ses précieuses annotations.

Je remercie Guillaume et Marc pour la confiance qu'ils m'ont accordée et qu'ils n'ont pas cessé de renouveler tout au long du chemin parcouru ces dernières années, pour la patience dont ils ont fait preuve durant les moments d'incertitude, pour la qualité de leur accompagnement et l'attention avec laquelle ils ont suivi mon parcours. Le rôle de directeur de thèse se résume-t-il en un paragraphe ? Il va sans dire que ce document n'existerait pas sans eux, sans les heures qu'ils ont passées à m'éclairer, me guider, me conseiller et travailler avec moi. Je ne saurais dire à quel point je leur suis redevable de s'être engagés à me mener jusqu'ici.

Les conférences ont été l'occasion pour moi d'en apprendre plus au sujet de ce qui se faisait dans les domaines de recherche au sein desquels mes travaux se sont intégrés, mais également d'obtenir d'inestimables remarques et des regards extérieurs sur ma production. En particulier, je remercie Marc Mezzarobba, Bernard Mourrain et Joris van der Hoeven à cet égard pour leurs questions et suggestions. Je remercie aussi Tristan Vaccon pour avoir discuté d'opportunités avec moi. Les personnes que j'ai pu rencontrer en conférence m'ont donné un meilleur aperçu du monde académique, à moi qui ai une formation d'ingénieur. Elles sont trop nombreuses pour être toutes citées.

Les lignes qui restent sont destinées à celles et ceux qui m'ont apporté par leur simple présence sur mon chemin, à qui je n'ai jamais directement su dire merci pour cela et qui ont indirectement eu une incidence sur mon travail. Je pense que les rencontres recèlent toujours des richesses : parfois l'apport de l'autre est inéluctable, et d'autres fois il faut savoir être patient et attentif pour apercevoir ce que l'autre peut nous enseigner. J'espère avoir su apprendre d'un peu de ce que chacun a pu avoir à m'apporter.

Avant de remercier le reste des membres permanents de l'équipe Gamble, qui ont contribué et contribue encore à créer un excellent cadre de travail et de formation pour leurs stagiaires, doctorants et post-doctorants, je voudrais à nouveau très brièvement remercier mes directeurs de thèse, Guillaume pour sa curiosité et Marc pour son calme et son recul.

En plus de leurs conseils généraux sur l'enseignement, je remercie Vincent, que j'ai sollicité un nombre considérable de fois, pour son pragmatisme et pour m'avoir rappelé que le bon sens n'est pas toujours naturel et Xavier, qui m'a aidé dans mes réflexions, pour son discernement ainsi que pour l'attention qu'il porte à l'utilité et l'efficacité.

Je remercie Alba pour son insatiable passion, Laurent pour son dévouement sérieux, Monique pour son immense expérience et son inflexible amour des choses bien faites, Olivier pour avoir été un chef d'équipe engagé et inspirant, même si je ne me suis pas converti à Ipe, Sylvain pour avoir signé un grand nombre de mes documents administratifs, pour son regard critique sur le monde et ses mots justes.

Je remercie toutes celles et tous ceux qui ont partagé le bureau que j'occupais durant ces années. Je me souviendrai de Camille pour les secrets que je garderai pour moi, mais aussi son admirable

rigueur, Charles pour son indéniable sérénité et pour avoir répondu à la quantité incalculable de questions que j'ai pu lui poser, George pour sa sympathie et son courage, Hypolite pour son ouverture d'esprit, Justin pour sa capacité de détachement et d'appréciation des choses, Léo pour sa droiture et les innombrables discussions par-dessus nos écrans, Léo pour son existence, ses hauts et ses bas, Loïc pour son allégresse débordante, Matthias pour son attachement aux règles, Pierre pour son envie de découverte, Rémi pour ses propos pertinents et son économie de paroles, Sarah pour sa spontanéité, son indépendance et ses silences pensifs, Tae-Kyeong pour m'avoir offert son point de vue sur la Corée du Sud et Théo pour son immanquable enthousiasme.

Je ne pourrai pas non plus oublier les post-doctorants de l'équipe, qui avaient leur bureau presque en face du nôtre. Ji-Won reste une énigme qui continue à m'intriguer. Benedikt et Florent ont chacun à leur tour passé de trop nombreux après-midis à parler de divers sujets dans notre bureau, je leur ai également rendu visite dans leur bureau un nombre de fois non négligeable, mais j'ai aussi trop longuement discuté avec eux devant ces mêmes bureaux. Pour tous ces moments, je les remercie.

Je remercie celles et ceux qui ne faisaient pas partie de l'équipe, mais que j'ai souvent vu dans notre bureau et qui ont animé une partie de mon temps. Je remercie Athénaïs pour sa bonne humeur, sa persévérance et tout ce qu'elle a initié au sein du laboratoire, Bastien qui a su continuer à organiser la vie au sein du laboratoire pour les non-permanents, Dominique pour sa gaieté, Gabrielle pour sa discrétion et Paul pour son enthousiasme.

Je remercie celles et ceux avec qui j'ai pu échanger au sein du laboratoire, que ce soit à table ou ailleurs. Je remercie Bruno et Jean-Yves d'avoir fait de ce lieu ce qu'il est aujourd'hui. Je remercie Caro, Floriane, Isabelle, Jérémy, Nathalie, Rémi et Tarek pour faire du Loria un laboratoire que les gens quittent à regret.

Je remercie celle et ceux qui m'ont permis d'enseigner à Télécom Nancy et Polytech Nancy et ont su m'accompagner dans cette tâche : Claude, Gérald, Hugues, Pierre, Sébastien et Suzanne.

Je souhaite tout particulièrement remercier celle et ceux dont j'ai été loin ces dernières années, celle et ceux que j'ai laissés pour aller à Nantes, Osaka, puis Nancy et que je laisserai peut-être encore pour découvrir d'autres villes, mais qui ont pourtant paradoxalement toujours été là. Je souhaite remercier celle et ceux qui, à défaut de me comprendre, me connaissent, m'ont accepté, m'acceptent et m'apprécient tel que je suis. Franck, Guillaume, Julian, Julien, Julien, Léo, Louis, Tanguy, Thomas et Zineb, merci.

Enfin, je remercie ma famille pour son infaillible soutien. Elle sera toujours plus fière que moi de ce que j'ai accompli.

À celles et ceux que j'aurais pu oublier, je voudrais dire que je suis désolé. J'ai écrit ses lignes après avoir longuement procrastiné avec le secret espoir de n' avoir oublié personne. Pour qui se reconnaîtrait, je n'attends pas de pardon.

Contents

List of Figures

List of Tables

List of Algorithms

List of Subroutines

Introduction

1 Context

Graphics are visual representations which can serve different purposes. They can be artistic, functional or even both. Our focus is on data visualization in computer graphics. So, we are interested in the functional aspect. More specifically, we focus on scientific visualization, that is graphical illustration of data in order to show data in a condensed way or even get insight about it. These techniques are useful from an academic point of view, but are also practical tools in engineering. The applications of the methods provided by scientific visualization are numerous. For instance, computer-aided design is as much a tool for artists as it is a tool for engineers: it is used in animation and video games, but also in modeling for simulations, mechanism design and control theory.

In today's digital world, scientific visualization is about data processed by computers and then interpreted by humans. The data itself may be multi-dimensional. The representations, which are produced, are made of one-, two- or three-dimensional geometrical objects, because that is what we, humans, are able to grasp. Nevertheless, data visualization provides tools to convey more dimensions, through shapes, size and colors for example. It remains that the representations themselves are at most three-dimensional. In this context, we propose methods for the visualization of plane curves and surfaces.

In cartography, topographic maps are examples of two-dimensional representations in which color is used to convey a third dimension. For instance, in Figure 1 a scale relates different colors to different elevation level. The boundary between two successive colors is a set of lines were the elevation is identical at each point. Those lines are called *contour lines* or *isolines*. They are defined as the solution set of an equation of the form

$$f(x, y) = h$$

where x and y are the coordinates on the plane, the function f returns the elevation at a given point, that is the height above or below a fixed reference point, and h is a fixed height. The generalization of contour lines to any number of variables is called *level sets*. The previous equation can be rewritten as

$$F(x, y) = f(x, y) - h = 0$$

where F is a function of two variables. Curves which are defined by an equation of the form $F(x, y) = 0$ are called *implicit curves*. The adjective "implicit" means that it is neither an equation in terms of x, nor in terms of y. When F is a polynomial function, the curve is called an *algebraic curve*. More generally, solutions of systems of polynomial equations are called

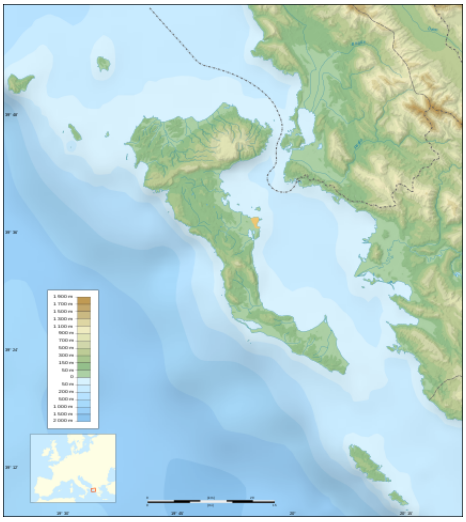


Figure 1: Topographic map of the Corfu island area in Greece by Eric Gaba (CC BY-SA 3.0).[1]

.

algebraic varieties and are the object of study of algebraic geometry. It has applications in many fields such as control theory, robotics or cryptography, among others.

Computer-aided design software manipulate objects represented by parametric curves and surfaces, and also implicit curves and surfaces. On the one hand, parametric representations are suited for generating points of a plot. On the other hand, implicit representations require solving an equation to locate the points of a curve or surface. Nonetheless, implicit curves and surfaces are convenient for the visualization of scalar fields like elevation, temperature, density or pressure. That is the approach taken for surface reconstruction in medical imaging. Magnetic resonance imaging (MRI) scan is used to get images of soft tissue and computed tomography (CT) scan is used for bones. Both methods produce two-dimensional images, corresponding to slices of the body. In 1987, Lorensen and Cline introduced the marching cubes algorithm to construct an approximated isosurface from an actually three-dimensional scalar field [LC87]. It paved the way to other algorithms which construct polygonal meshes which approximate the surface (see Section 2).

Another way to visualize an implicit surface is to use ray tracing. The function which defines the implicit surface is a signed distance function. Ray tracing compute intersections of the surface with straight lines, which are defined by one-dimensional equations. Thanks to recent advances in hardware, it is a very efficient way to visualize data and interact with it in real time. Ray tracing constructs a view of the objects, not the three-dimensional objects present in the scene. The construction can be done in real time. So, the view point can be moved around, allowing to move around the objects as in the other methods. Additionally, ray tracing models light transport and can therefore intrinsically simulate effects like reflection, refraction or shadows.

2 State of the art

We do not intend to be exhaustive in the description of the approaches. The algorithms we present here are meshing algorithms. In the two-dimensional setting, a mesh of a curve is a

[1]Source: `https://commons.wikimedia.org/wiki/File:Corfu_topographic_map-blank.svg`.

set of polygonal chains which approximates it. In the three-dimensional setting, it is a set of polygonal surfaces. The main ideas originated either from the marching cubes algorithm or from curve continuation methods [GVJ+09, Part III]. Only the first family is presented in the following sections. Briefly, the latter locate a point on the curve and try to find the rest of the curve by continuity. Different challenges can arise, like the fact that one has to find a point on each component of the curve or the risk of cycling in the same portion of the curve.

We first present the *marching cubes* algorithm in Section 2.1. Then, we presented methods which preserve the topology of the curves in Sections 2.2 and 2.3. Roughly, that means that the latter methods preserve the loops and the self-intersections of the curves as opposed to marching cubes.

2.1 Marching cubes

The original application of the *marching cubes* algorithm is the construction of a visualization of data from computed tomography (CT) and magnetic resonance (MR). In the medical context, it is reasonable to work with continuous functions. Marching cubes aims at constructing mesh approximating isosurfaces of a scalar field [LC87]. Let us first look at the simpler two-dimensional version of the algorithm: *marching squares*.

Marching squares

We would like to visualize the curve $\mathcal{C}$ defined as the set of points which verify $f(x, y) = 0$. Instead we sample f on the domain of interest and try to visualize $\mathcal{C}'$ defined as the set of points which verify $\phi(x, y)$ where ϕ is an interpolant of f. We actually construct a piecewise linear output, which is said to be topologically consistent. That is to say, the curve returned is crack-free. From a set of sample points of a function f, marching squares tries to approximate an interpolating function for the samples.

Let us describe the gist of the algorithm. It is given a function which defines the implicit curve we want to represent and the window in which it should be visualized (see Figure 2a). The window is given with a grid. The function is evaluated at each vertex and the sign of the evaluation is stored (see Figure 2b). The function is assumed to be continuous. Therefore, if two adjacent vertices have different signs, it means that the function vanishes along the edge which connects them. In other words, the target curve crosses that edge. From this remark, for each square of the grid it is possible to identify edges crossed by the target curve (see Figure 2c). Each edge of a square is either crossed or not, this results in $2^4 = 16$ different combinations. They can even be reduced to four cases, up to rotation and sign change. The algorithm provides a lookup table, which gives a way to draw a linear approximation of the curve between the edges it crosses inside the square for each case. It is composed of zero, one or two lines. By construction, the algorithm returns a piecewise linear representation of the curve (see Figure 2d).

The evaluation of the function at the vertices and the construction of the approximation is local inside each square. Yet, it yields a global approximation. That means that the procedure can be parallelized. The lookup table is easy to produce. Remark however that there is an ambiguity for saddle points (see Figure 3). One has to make a choice. The disambiguation either produces one component or two components of the curve (see Figure 3). In the illustrations the dotted lines are connected to the center of the edges, but the approximation could be better.

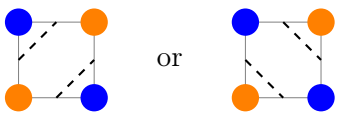
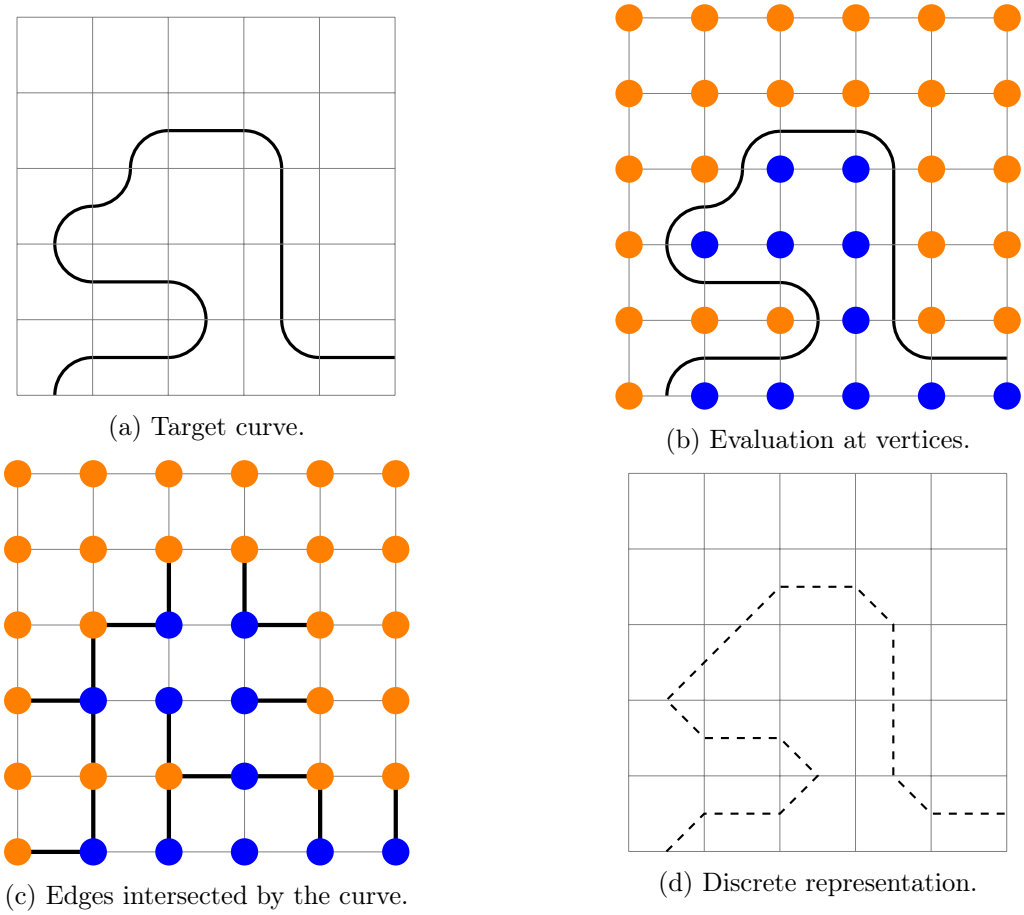


Figure 3: Ambiguity of saddle points.



(a) Target curve.

(b) Evaluation at vertices.

(c) Edges intersected by the curve.

(d) Discrete representation.

Figure 2: Illustration of the steps of the marching squares algorithm; orange dots (●) represent positive evaluations, blue dots (●) represent negative evaluations.

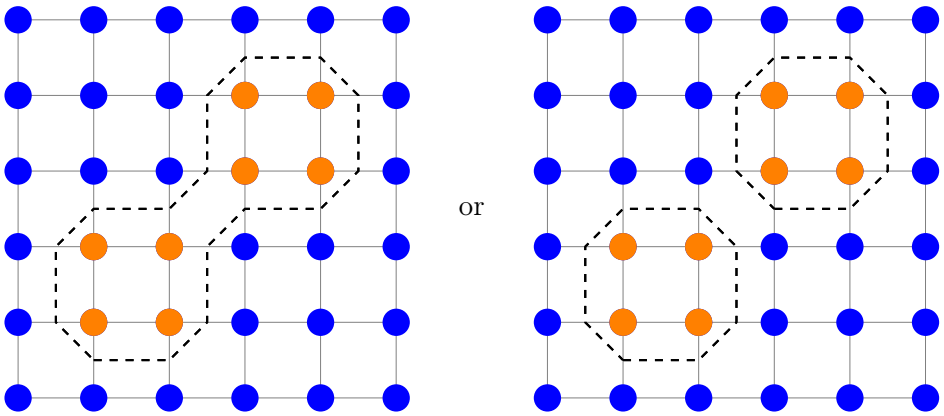


Figure 4: Example of the consequence of the ambiguity of saddle points.

The point on the edge could be the linear interpolation of the vertices weighted by the evaluation at those points.

Marching cubes

Naturally, the ideas presented about marching squares extend to the three-dimensional case. The marching cubes algorithm receives a function which defines a target implicit surface. The window and the grid are three-dimensional. The function is evaluation at each vertex. For each cube, there are $2^8 = 256$ different combinations. They can be reduced to 15 cases, up to rotation, reflection and sign change. Once again, there is a lookup table. For each case, it gives one or several polygons. By construction, the polygons are connected and form the desired surface.

Since this first paper, several others proposed to add cases to the table to remove ambiguities. Some focus on topological consistency, others on topological correctness. One of the latest versions has been implemented by Lewiner et al., but it still has some issues [LLVT03]. It is an improved version of Chernyaev's Marching Cubes 33 algorithm.

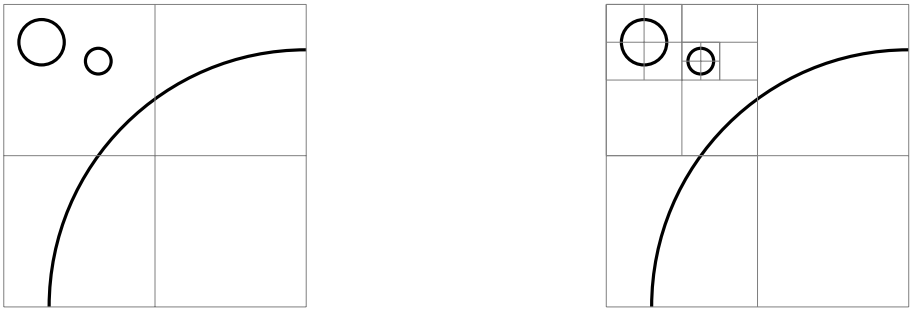
2.2 Methods with adaptive grids

The fixed grid of marching squares, respectively marching cubes, is not sufficient to capture all the details of the curve, respectively the surface. There may be features smaller than the resolution. So some methods use an adaptive mesh in order to get them. In this section, we present two-dimensional versions of the algorithms, for the three-dimensional ones derive from them directly.

These methods use adaptive grids, which are locally refined in order to capture the details of the curve, as illustrated by Figure 4.

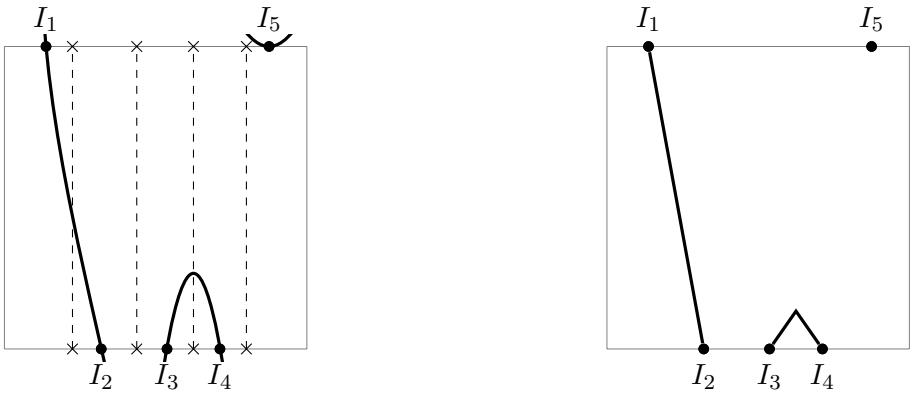
Snyder's algorithm

Snyder introduces the use of interval arithmetic in computer graphics. He proposes an algorithm to approximate implicit curves. His algorithm uses a global parameterization criterion that subdivides cells until no detail of the curve can be lost, in order to achieve topological correctness. The algorithm relies on this global parameterizability criterion, it handles multivariate implicit curves.



(a) A 2×2 grid. (b) The same grid with local refinements.

Figure 5: Grid before and after refinement.



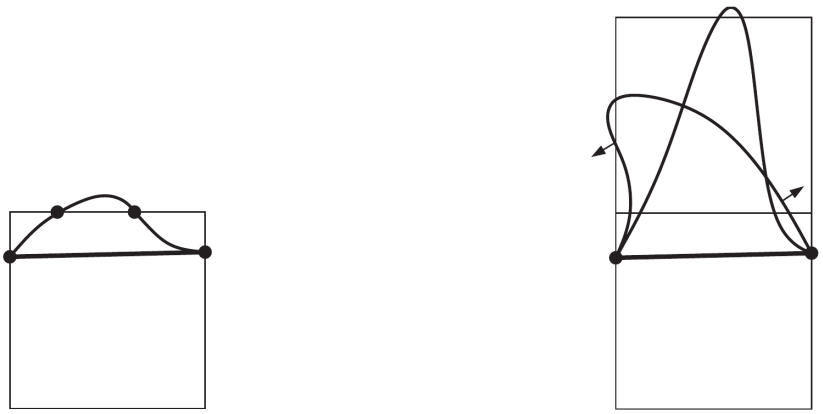
(a) Identification of the connections. (b) Piecewise linear approximation of the curve.

Figure 6: Construction of the approximation for a globally parameterizable curve in a given cell.

A curve $\mathcal{C} = \{(x, y) \in [a, b] \times [c, d] \,|\, f(x, y) = 0\}$ is *globally parameterizable* in x, if for each value x the equation $f(x, y) = 0$ has at most one solution y in the cell $[a, b] \times [c, d]$.

The grid is refined using interval arithmetic until the curve is globally parameterizable inside each cell of the grid — it can be written as the solution of an equation of the form $y = f(x)$ or $x = g(y)$. Without loss of generality, let us look at a curve globally parameterizable in x in a cell (see Figure 5). First, one locates the intersections of the curve with the boundaries of the cell. There is at most one intersection with the left boundary and the right boundary, and a finite number of intersections with the top and bottom boundaries. Those intersections (I_i) are ordered from left to right according to their x-coordinate (x_i): $x_1 < x_2 < \ldots$.

Since the curve is globally parameterizable, two consecutive intersections may be linked or not, no other connection between intersections is possible. For each pair of consecutive intersections I_i and I_{i+1}, one simply checks if the vertical line of equation $x = c$ with $c \in]x_i, x_{i+1}[$ has different signs when it intersects the top and the bottom boundary. They are represented with dashed lines in Figure 5a. If there is a change of sign, the intersections are linked; otherwise, no action is required. Figure 5b shows different cases: I_1 and I_2 are connected with a line segment, I_3 and I_4 are connected with two line segments and I_5 is linked to no line segment of this cell. This step is also done with interval arithmetic. So, if it is not possible to decide whether there is a change of sign or not, the cell is further subdivided.



(a) Isotopic approximation.

(b) Violation of the small normal variation.

Figure 7: Illustration of the small normal variation criterion, from [BT06, Section 5.2.4].

Plantinga and Vegter's algorithm

Plantinga and Vegter use a criterion on the gradient to determine whether a box is refined enough to produce a correct mesh with the signs of the evaluation at the vertices. It uses the small normal variation condition:

$$\langle \nabla f(p_1), \nabla f(p_2) \rangle \geq 0, \forall p_1, p_2 \in [a, b] \times [c, d].$$

This condition constraints the angle between any pair of vectors obtained by evaluating the gradient of f in the square $[a, b] \times [c, d]$. That translates into a condition on the normal vectors of the curve. As illustrated in Figure 6, it guarantees that the curve does not go far outside of the cell. By simply evaluating the vertices, the two intersections with the upper edge are missed in Figure 6a. Nonetheless, the segment between the endpoints from the left to the right edge is isotopic to the actual curve. The algorithm starts with a given box and recursively subdivides it until, in every box $[a, b] \times [c, d]$, either $f(x) \neq 0$ for all $x \in [a, b] \times [c, d]$, or the small normal variation condition holds. Once the grid is refined, the curve cannot leave the adjacent square, as in Figure 6b, without violating the small normal variation condition in the adjacent square. This allows the approximating segment to be isotopic and also geometrically close. All the computations rely on interval arithmetic to provide the correctness.

Thanks to the small normal variation condition Plantinga and Vegter's algorithm does not need topological correctness in each cell to provide global topological correctness, as opposed to Snyder's algorithm.

2.3 Algebraic methods

Some algorithms produce a topologically correct approximation of curves using algebraic methods. They only require the curve to be algebraic, no smoothness is assumed. So, they can handle singular curves. The polynomial is supposed to be square-free and to have no vertical asymptote. The algorithms decompose the plane into vertical stripes which contain disjoint portions of the curve and vertical lines which contain a finite number of points of the curve. That uses the cylindrical algebraic decomposition (CAD), which was originally introduced by Collins [Col75]. It later became a tool to analyze the topology of curves [Roy90]. The vertical stripes locate

(a) c is a critical point and s is an singular point. (b) Topologically equivalent polygonal chain.

Figure 8: Decomposition of the curve defined by $y^2 - (x^3 + x^2) = 0$.

regular portions of the curve and the vertical lines locate critical points (see Definition 0.2.1). Those points include also the singular points (see Definition 0.2.2).

Definition 0.2.1. A *critical point* (x, y) verifies

$$f(x, y) = 0 \text{ and } \frac{\partial f}{\partial y}(x, y) = 0.$$

Definition 0.2.2. A *singular point* (x, y) verifies

$$f(x, y) = 0, \frac{\partial f}{\partial x}(x, y) = 0 \text{ and } \frac{\partial f}{\partial y}(x, y) = 0.$$

Example 0.2.1. For the curve defined by $y^2 - (x^3 + x^2) = 0$ in Figure 7a, the vertical lines are defined by $x = -1$ and $x = 0$ and the stripes are the open domains $]-\infty, -1[\times \mathbb{R}$, $]-1, 0[\times \mathbb{R}$ and $]0, +\infty[\times \mathbb{R}$.

From this decomposition, one can construct a piecewise linear topologically equivalent representation. In each stripe, one can find the number of branches by choosing a value for x to get a univariate polynomial in y and then by isolating the zeros in order to count them. One has to determine the number of branches which emanate on the right and on the left for each critical point. This requirement is not visible in Figure 7, because we chose a simple example.

Example 0.2.1 (continued)**.** We evaluate our polynomial $P(x, y) = y^2 - (x^3 + x^2)$ for $x = -2$, $x = -\frac{1}{2}$ and $x = 1$. That gives us $P_1(y) = y^2 + 4$, $P_2(y) = y^2 - \frac{1}{8}$ and $P_3(y) = y^2 - 12$. P_1 has no real root. P_2 and P_3 have both two distinct real roots. Therefore, the curve has no branch in $]-\infty, -1[\times \mathbb{R}$, two branches in $]-1, 0[\times \mathbb{R}$ and in $]0, +\infty[\times \mathbb{R}$. In this example the way the branches connect to the critical points is trivial and results in Figure 7b.

With the CAD, one exhibits the piecewise smooth nature of any algebraic curve. So, any relevant method can be used to get better geometric approximations of those portions. The challenge is in linking the smooth domains at the singularities.

3 Motivations

3.1 From root finding to surface drawing

Representing algebraic curves and surfaces implies being able to find the solutions of the polynomial equations which define them. Therefore, let us briefly run through this connection between algebra and geometry, and some useful tools.

In the mathematical world, *finding a root* initially meant finding the square root of a real number and then the n^{th} root of a real number.[2] That is for a real number a finding x such that $x^2 = a$ or more generally such that $x^n = a$ with $n \in \mathbb{N}$. This denomination then extended to polynomial equations of the form $a_0 + a_1 x + \cdots + a_n x^n = 0$. For these objects, finding the roots means finding all the solutions. Eventually, now the term "root" applies to any equation. That being said, one usually refers to zeros of some function and to roots of a polynomial function in order to describe the zero set of the function, which is the set comprised of the values at which the function vanishes — for a real-valued function $f\colon X \to \mathbb{R}$, it is $f^{-1}(0) = \{x \in X \mid f(x) = 0\}$.

The study of univariate polynomial functions led to the development of methods to find their roots. The idea of "root" can be extended to zeros of multivariate functions. Systems of multivariate polynomial equations happen to be at the center of algebraic geometry. For example, the zeros of a bivariate polynomial function P lie on a plane curve $\mathcal{C} = \{(x, y) \in \mathbb{R}^2 \mid P(x, y) = 0\}$ and the zeros of a trivariate polynomial function Q lie on a surface $\mathcal{S} = \{(x, y, z) \in \mathbb{R}^3 \mid Q(x, y, z) = 0\}$.

We are interested in the real roots of the polynomials. Due to algebraic number representation problems, most root finding algorithms face numerical instability issues [GVJ+09, Part II]. The standard way to find the roots is to isolate them — compute disjoint intervals which contain exactly one root each, but whose union contain all of them — and then improve the precision of the results. Some tools like Sturm's theorem, Descartes' rule of signs, Budan's theorem and Vincent's theorem use the sign variation of a given sequence in order to isolate the roots. Interval arithmetic, which was originally developed as a way to provide guarantees on the results of computations, can also be used in reliable root isolation and approximation techniques.

3.2 High degree curves and surfaces

Scientific visualization allows its users to build intuition and get an understanding of their data and of what it models. In control theory, computer-aided design is a tool to optimize and tune parameters of a system. Let us look at an example in robotics, in order to see that one may have to deal with algebraic curves of high degrees [CLO07, Chapter 6].

Robots are characterized by the "positions" they can reach. Robotics has to answer two main questions: "Which positions can a robot reach?" and "How can a robot reach a given position?" [Lat12]. Those "positions" depend on the structure of the robot itself. Let us consider a simple example like the robotic arms, which can be found on assembly lines of car manufacturing plants (see Figure 9). They mimic a more complex version of a human arm, made of rigid bodies connected by joints and with a tool at the end. Such robots are able to rotate their components around the joints. These rotations are parameters which define the "position" of the tool. The *degrees of freedom* of a mechanical system is the number of independent parameters which define the "position" of the tool. It corresponds to the dimension of the space in which all the positions can be represented. Similarly, mechanical engineers are interested in the number of independent

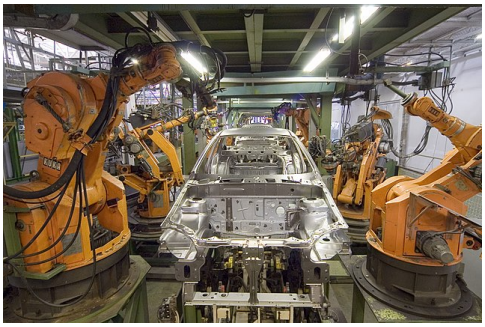


Figure 9: Industrial robots from KUKA by Mixabest (CC BY-SA 3.0).[3]

[2] See Gandz on the origin of the word "root" and for further information [Gan26; Gan28].
[3] Source: `https://commons.wikimedia.org/wiki/File:KUKA_Industrial_Robots_IR.jpg`.

degrees of freedom that one has in the configuration (position and orientation) of the tool. That corresponds to the dimension of the geometrical object which represent the possible configurations. Therefore, one has to deal with high dimensional objects in high dimensional spaces. Planning the motion of a robot requires to manipulate those objects and understand their shapes. Since the parameterization of the problem may introduce multivariate polynomial functions, they may be algebraic varieties. One could need to compute intersections of hypersurfaces or projections of curves [CLO07; PM02, Chapter 5]. These kind of operations can produce high degree algebraic varieties. It could grow beyond a degree of 10 or 20.

Obviously, these concerns are not restricted to robotics. Other areas are mechanical engineering in a broader sense, civil engineering, aerospace engineering, biomechanics and even molecular modeling. Nowadays, we are relying on computers to perform complicated operations and solve complex problems. Therefore, we draw on computer algebra techniques to answer those questions efficiently and precisely.

4 Contributions

4.1 Problem statement

We address the problem of visualizing implicit algebraic plane curves and surfaces, that is curves defined by a bivariate polynomial equation

$$P(x, y) = 0$$

and surfaces defined by a trivariate polynomial equation

$$Q(x, y, z) = 0.$$

We aim at efficiently computing a discrete representation on a grid of fixed resolution and to handle specifically the case of high degree curves or surfaces with a high resolution. In most state-of-the-art approaches, the authors analyze and optimize their algorithms while assuming that the cost of evaluating the function P or Q is negligible. Such an assumption becomes invalid when the degree is high. For example, a bivariate polynomial of total degree 20 has 231 monomials. In particular, in our case, we consider that degrees are at least 20. For the sake of simplicity, we handle square $N \times N$ grids and we consider the resolution N to be high when it is greater than 1,000.

4.2 Overview

Fast evaluation at nodes with Chebyshev polynomials

The main idea to compute implicit algebraic curves and surfaces more efficiently is to use fast multipoint evaluation methods from computer algebra in order to evaluate the polynomials which define these curves and surfaces. Indeed, a polynomial of degree d can be evaluated at one value in linear arithmetic complexity using Horner's method. Thus, the naive evaluation at d values is quadratic in d. It can actually be cheaper by using a divide-and-conquer approach. The fast multipoint evaluation algorithm improves the complexity of these d evaluations to soft-linear in d (that is $O(d \operatorname{polylog}(d))$) [vzGG13, Chapter 10]. Remark however that fast multipoint evaluation algorithms have a bit complexity quadratic in d, if we work with a fixed constant precision lower than d [vdH08; KS16].

If we restrict our multipoint evaluation on a set of Chebyshev nodes (see Section 2.2 of Prerequisites), we can then use the Discrete Cosine Transform (DCT), in the same way as the Discrete Fourier Transform (DFT) can be used for multipoint evaluation on the roots of unity in the complex field [vzGG13, Section 8.2]. In this case, the bit complexity is soft-linear in d, when the precision is constant [Sch82, Section 3]. This special multipoint evaluation inherits the numerical stability of the DCT and we extend the error analysis already known for the DFT [BJM+20]. Thus, we convert the polynomial from the monomial basis to the Chebyshev basis and apply the DCT on the coefficients of the resulting polynomial. By doing so, we get a fast evaluation of the original polynomial at the Chebyshev nodes.

Fast evaluation on edges with interval arithmetic or Taylor approximation

In order to know whether the plane curve or the surface crosses an edge of a pixel, respectively an edge of a voxel, we adapt the fast evaluation method to be able to evaluate a univariate polynomial P on edges. We guarantee that we do not miss any intersection point of the curve with the underlying grid and that we exclude pixels or voxels when the evaluation of P on their edges is far from zero. Such guarantees are obtained using a combination of interval arithmetic, a careful analysis of the numerical error of the DCT algorithm and Taylor approximation.

Even though this analysis of the DCT allows us to bound the values of the polynomial P at the Chebyshev nodes (c_n), it cannot bound its values on small intervals covering the edges around the Chebyshev nodes. We solve this problem by using Taylor approximation at c_n:

$$P(x) = \underbrace{P(c_n) + \cdots + \frac{P^{(k)}(c_n)}{k!}(x - c_n)^k}_{P_{c_n,k}(x)} + R_{c_n,k}(x).$$

The Taylor approximations of order k are low degree approximating polynomials $P_{c_n,k}$ which can be evaluated around each node c_n with interval arithmetic. The coefficients of the Taylor approximations at all the Chebyshev nodes can be efficiently computed using the DCT k times, once for each successive derivative of P. Moreover, we control the error by bounding the remainder $R_{c_n,k}$. By combining the evaluation of low degree Taylor approximations to the fast evaluation of their coefficients using the DCT, we get fast evaluations on the edges.

We arbitrarily choose $k = 3$ for our experiments, because a slight change of this value has a negligible effect on the output, but a larger change drastically affects the computation time.

Implicit curve drawing and guarantees

In Part I, we present three algorithms which use the fast multipoint evaluation based on the DCT to draw implicit curves defined by bivariate polynomials $P(X, Y)$. Given a grid, they detect either whether the curve intersects the lines of the grid or whether the curve intersects a pixel of the grid. The first two algorithms detect intersections with edges and the third one detects intersections with pixels.

First, we design an algorithm which evaluates the input polynomial P using the DCT both in x and in y (see Idea 3). Subsequently, the grid has to be aligned on Chebyshev nodes (c_n) in both direction. The algorithm evaluates the first variable by using the DCT, then uses Taylor approximation with the DCT to do the evaluations along the vertical fibers. That is illustrated by the top half of Figure 8. Thus, writing $P(X, Y) = \sum_{s=0}^{d} p_s(X)Y^s$, we can use the DCT on

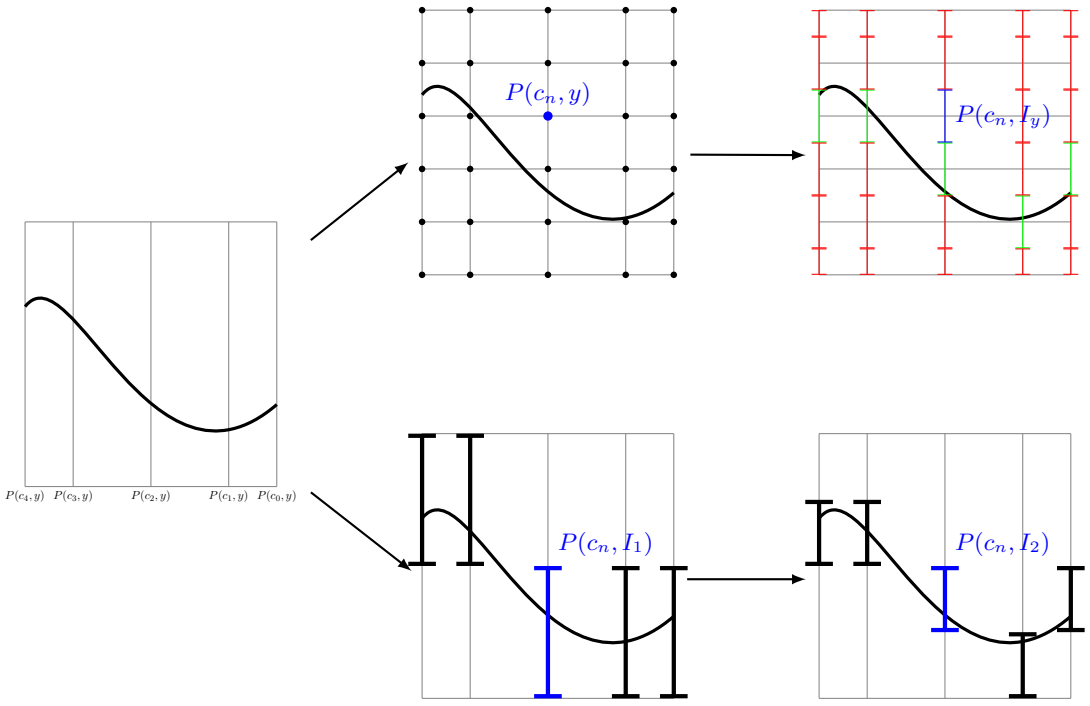


Figure 10: Diagram of the main steps of edge enclosing algorithms. First, the polynomial is partially evaluated in the first variable at Chebyshev nodes. Then, either the partial polynomials are evaluated around Chebyshev nodes along the vertical axis (top), or the partial polynomials are used to isolate the curve by subdivision (bottom).

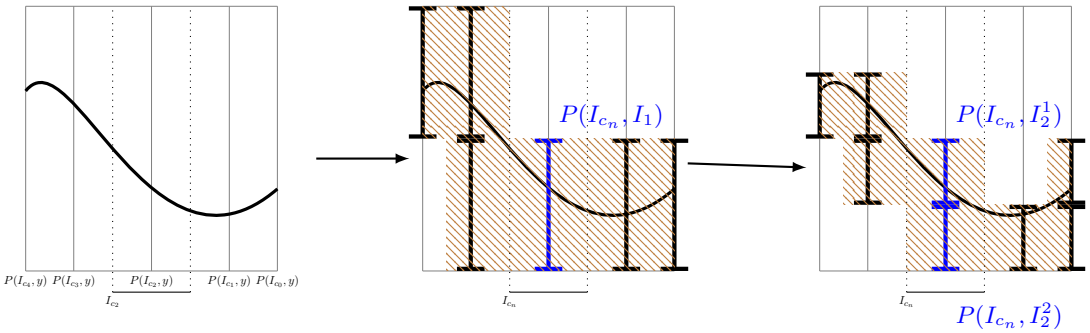


Figure 11: Diagram of the main steps of the pixel enclosing algorithm. First, the polynomial is partially evaluated in the first variable around the Chebyshev nodes. Then, the partial polynomials are used to isolate the curve by subdivision.

each p_i, which leads to N univariate polynomials $q_n(Y) = P(c_n, Y)$ of degree d, that can also be evaluated using the DCT once again. This can be done in a quasi-quadratic number of arithmetic operations in N if $N \gg d$.

Second, we design an algorithm which incorporates the fast evaluation with the DCT to a subdivision method from interval arithmetic (see Ideas 3 and 2). For the computation of implicit curves, interval arithmetic combined with quad-tree subdivision approaches can discard large parts of the plane with few evaluations and reduce the number of evaluations required to plot the considered curve [Sny92; PV04]. The domains are evaluated with interval arithmetic; if they contain a zero, then they are subdivided and evaluated recursively up to a given precision in order to locate the zeros. We take advantage of the DCT to rapidly evaluate the first variable, then we use subdivision to locate intersections of the curve with vertical fibers of the grid. That is illustrated by the bottom half of Figure 8. This constrains us to use a non-uniform grid aligned on Chebyshev nodes (c_n) along the x-axis. Although this grid is non-uniform, the distance between two consecutive nodes of such a grid of size $N \times N$ is always less than π/N, while the distance would be $1/(N-1)$ for a uniform grid. This makes the grid sufficiently dense for plotting. Writing $P(X, Y) = \sum_{s=0}^{d} p_s(X)Y^j$, we can first use the DCT on each p_s, which leads to N univariate polynomials $q_n(Y) = P(c_n, Y)$ of degree d, and then we solve each polynomial $q_n(Y)$ with a subdivision approach. If $N \gg d$, this leads to a complexity in $\widetilde{O}(dNT)$, where $T \leq N$ is the maximum number of nodes in the considered subdivision trees. In Chapter 3 of Part I, we show that this approach performs well in practice.

For these two algorithms, the output is a set of vertical edges. Inverting the role of X and Y and calling the algorithms a second time produces the set of horizontal edges. Getting a drawing amounts to lighting a pixel if one of its edges is in the output.

Third, we integrate the use of the Taylor approximation to the second algorithm (see Idea 4). That is illustrated by Figure 9. This allows us to detect the presence of the curve inside pixels and requires only one call to the algorithm. Once more, we write $P(X, Y) = \sum_{s=0}^{d} p_s(X)Y^s$. For the partial evaluation, we can use Taylor approximations at the Chebyshev nodes with the DCT on each p_s, which leads to N univariate polynomials $q_n(Y) = P(I_{c_n}, Y)$ of degree d, where I_{c_n} is an interval around c_j. Then, we solve each polynomial $q_n(Y)$ with the subdivision approach based on interval arithmetic along vertical stripes.

We implemented our approaches in Python. In Chapter 3 of Part I, we remark that the timings

match the complexity analysis. We also compare our implementations with state-of-the-art software, and show that for high resolutions, the approach based on a mixed strategy using fast multipoint evaluations and subdivision proves to be the fastest. We are also able at least to provide guarantees for the output curve and at best to certify it. Indeed, some algorithms presented here trade certification for speed, but can still provide partial guarantees on the output.

Implicit surface drawing and guarantees

In Part II, we show how ideas used for plane curves can be extended to the three-dimensional case. We present two algorithms for the drawing of implicit surfaces defined by a trivariate polynomial $P(X, Y, Z)$. We use fast evaluation techniques for the two first directions and reserve subdivision for the last one.

The first algorithm simply applies the DCT successively in the first two variables to get a fast evaluation of the partial polynomials $q_{n,m}(Z) = P(c_n, c_m, Z)$ and then performs a subdivision along the z-axis of a three-dimensional Chebyshev grid. The algorithm has to be called three times to get the edges in each direction, by inverting the roles of X, Y and Z. Similarly to the two-dimensional algorithm, one gets a drawing by lighting a voxel if one of its edges is in the output. Those voxels are also the only ones the marching cubes algorithms needs to examine. By doing so, we improve the costliest step of that algorithm. Not only do we evaluate the polynomial fast with the DCT, but we also reduce the number of voxels that need to be visited to construct the approximating mesh. The marching cubes algorithm is often the first step of the visualization, even for methods with refined grids. Therefore, these contributions are also relevant for those methods.

The second algorithm uses Taylor approximation with the DCT for the evaluation of the first two variables, getting the partial polynomials $q_{n,m}(Z) = P(I_{c_n}, I_{c_m}, Z)$. Then it solves each polynomial $q_{n,m}(Y)$ with the subdivision approach based on interval arithmetic. That allows us to detect the presence of the surface inside voxels with one call to the algorithm.

Conventions

Notations

Symbol(s)	Meaning
$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{D}$	set of dyadic numbers
$\mathbb{F}$	set of floating-point numbers
d	partial degree of a polynomial
N	grid resolution
T	size of a tree
$[\![a, b]\!]$	integer interval
I, J	intervals
x_i, y_j, z_k	coordinates in the three dimensions
$a_{r,s,t}$	coefficient of a trivariate polynomial

Algorithm naming and highlighting colors

For the main algorithms, which return edges, pixels or voxels, we use the following naming convention: <DOMAIN>_ENCLOSING<ELEMENT>_<METHOD>[-<METHOD>[-<METHOD>]].

- <DOMAIN>: REAL or FLOAT
- <ELEMENT>: EDGES, PIXELS or VOXELS
- <METHOD>: method used in processing one dimension

For visual clarity, the different steps of the main algorithms are highlighted in different colors (see Algorithm 9 for instance).

- evaluation
- subdivision
- decision

If two distinct evaluation steps are explicitly written, two colors are used.

- first evaluation
- second evaluation

Prerequisites

1 Models of arithmetic

In Part I, Chapter 1 is concerned by the design of a fast algorithm to visualize curves which works with exact real arithmetic. It is the basis for Chapter 2, which is concerned by the design of practical algorithms in order to implement them. Indeed, physical computing devices have limitations. For instance, they cannot handle numbers as a continuum. They rather deal with a discrete representation — floating-point numbers, which are a subset of real numbers [Hig02, Chapter 2].

1.1 Floating-point numbers

We denote by $\mathbb{F}$ the set of all floating-point numbers and by $\mathbb{R}$ the set of all real numbers. $\mathbb{F}$ is defined by four integer parameters b, p, e_{min} and e_{max}, and an element x of $\mathbb{F}$ is of the form

$$x = \pm\, m\, b^e$$

where

- $m = (d_0.d_1 \dots d_{p-1})_b$, with $d_i \in [\![0, b-1]\!]$, is the significant,
- b is the radix, that is the number of unique digits used to represent numbers,
- e is the exponent,
- e_{max} is the maximum value of the exponent e,
- e_{min} is the minimum value of the exponent e and
- p is the precision, that is the number of digits in the significant.

The IEEE standard 754 for floating-point numbers impose additional conditions in its different formats such as $e_{min} = 1 - e_{max}$ [IEE19]. The representations also encapsulate the notion of signed zero, signed infinity and not-a-number (NaN). When the result of an arithmetic operation is in $\mathbb{R} \setminus \mathbb{F}$, the standard provide rules to round it to a value in $\mathbb{F}$. The difference between the result obtained with exact arithmetic and rounded arithmetic is called the *roundoff error*.

1.2 The real random-access machine model

In order to analyze the computational complexity of our algorithms in Chapter 1 of Part I, we introduce a model of computation which is a simplified mathematical model of a standard computer. A model of computation specifies the primitives operations that may be executed and

their respective costs. To each primitive operation, we assign a fixed cost, that is charged when the operation is executed. We present here a model of computation for real numbers as described by Preparata and Shamos [PS85, Section 1.4]. In the *real random-access machine* (real RAM) model, the following operations are primitive and are available at unit cost (unit time):

1. the arithmetic operations with real numbers ($+$, $-$, $\times$, $/$);
2. comparison between two real numbers ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$);
3. indirect addressing of memory (integer addresses only).

For applications requiring them:

4. k-th root, trigonometric functions, exponential function, and logarithm function (in general, analytic functions).

The real RAM model is moreover capable of holding a single real number in each storage location, getting rid of the roundoff problem of finite-precision floating-point arithmetic.

In the rest of the document, we present algorithms which are either prefixed by REAL_ or by FLOAT_ depending on the model of arithmetic they are intended to work for.

2 Chebyshev polynomials and Chebyshev nodes

2.1 Chebyshev polynomials

Chebyshev polynomials can be defined as the unique sequence of polynomials $(T_k)_{k\in\mathbb{N}}$ satisfying

$$T_k(\cos\theta) = \cos(k\theta). \tag{1}$$

They can also be computed using the fundamental recurrence relation and its initial conditions [MH03, Equation (1.3)]:

$$\begin{aligned} &T_0(X) = 1, \\ &T_1(X) = X, \\ &T_k(X) = 2XT_{k-1}(X) - T_{k-2}(X), \text{ for } k \geq 2. \end{aligned} \tag{2}$$

2.2 Chebyshev nodes

For $N \in \mathbb{N}^*(N > 0)$, the Chebyshev nodes $(c_k)_{k\in[\![0,N-1]\!]}$ are the roots of T_N:

$$c_k = \cos\left(\frac{2k+1}{2N}\pi\right) \text{ for } k \in [\![0, N-1]\!]. \tag{3}$$

Lemma 0.2.1. *For $N \in \mathbb{N}^*$, the distance between two consecutive Chebyshev nodes is smaller than $\frac{\pi}{N}$.*

Proof. Let $N \in \mathbb{N}^*$. For all $k \in [\![0, N-1]\!]$, $c_k \in]-1, 1[$.

The cosine function is strictly decreasing on $]0, \pi[$ and its derivative has its maximum at $\frac{\pi}{2}$. So, for a regular sample of values of cosine the greatest distance between two consecutive points is around $\frac{\pi}{2}$.

(a) $N = 6$. (b) $N = 7$.

Figure 1: Illustration of the Chebyshev nodes for even and odd values of N.

The Chebyshev nodes are a regular sampling of the cosine function. Thus,

$$\max |c_i - c_{i+1}| = \begin{cases} c_{\frac{N}{2}-1} - c_{\frac{N}{2}} \text{ if } N \text{ is even} \\ c_{\frac{N-1}{2}} - c_{\frac{N+1}{2}} \text{ if } N \text{ is odd.} \end{cases}$$

If N is even,

$$\begin{aligned} c_{\frac{N}{2}-1} - c_{\frac{N}{2}} &= \cos\left(\frac{2(\frac{N}{2}-1)+1}{2N}\pi\right) - \cos\left(\frac{2\frac{N}{2}+1}{2N}\pi\right) \\ &= \cos\left(\frac{N-1}{2N}\pi\right) - \cos\left(\frac{N+1}{2N}\pi\right) \\ &= \cos\left(\frac{\pi}{2} - \frac{\pi}{2N}\right) - \cos\left(\frac{\pi}{2} + \frac{\pi}{2N}\right) \\ &= -\sin\left(-\frac{\pi}{2N}\right) + \sin\left(\frac{\pi}{2N}\right) \\ &= 2\sin\left(\frac{\pi}{2N}\right). \end{aligned}$$

If N is odd,

$$\begin{aligned} c_{\frac{N-1}{2}} - c_{\frac{N+1}{2}} &= \cos\left(\frac{2(\frac{N-1}{2})+1}{2N}\pi\right) - \cos\left(\frac{2\frac{N+1}{2}+1}{2N}\pi\right) \\ &= \cos\left(\frac{\pi}{2}\right) - \cos\left(\frac{\pi}{2} + \frac{\pi}{2}\right) \\ &= \sin\left(\frac{\pi}{N}\right) \end{aligned}$$

Since $\forall x > 0, \sin(x) < x$, we have $\max |c_i - c_{i+1}| < \frac{\pi}{N}$, for N even and odd. □

2.3 Converting a polynomial from the monomial basis to the Chebyshev basis

For $d \in \mathbb{N}$, the Chebyshev polynomials $(T_k(X))_{k \in [\![0,d]\!]}$ form a basis for the polynomials of degree d. Algorithm 1 allows us to convert a polynomial from the monomial to the Chebyshev basis. It does the change of basis progressively as follows [KLVS98, Section 11.3]. Let the coefficients in the monomial basis be the $(a_i)_{i \in [\![0,d]\!]}$ and the coefficients in the Chebyshev basis be the $(b_{i,0})_{i \in [\![0,d]\!]}$. In order to get them, we compute the triangular matrix $(b_{i,j})_{i \in [\![0,d-j]\!], j \in [\![0,d]\!]}$: we start by $b_{0,d}$, follow with each consecutive row and end with the $(b_{i,0})_{i \in [\![0,d]\!]}$.

$$\begin{array}{rcccccccc}
P(X) & = & a_0 & + & a_1X & + \ldots + & a_{d-1}X^{d-1} & + & a_dX^d \\
 & = & a_0 & + & a_1X & + \ldots + & a_{d-1}X^{d-1} & + & X^d b_{0,d}T_0(X) \\
 & = & a_0 & + & a_1X & + \ldots + & X^{d-1}[b_{0,d-1}T_0(X) & + & b_{1,d-1}T_1(X)] \\
 & & & & & \vdots & & & \\
 & = & a_0 & + & X[b_{0,1}T_0(X) & + \ldots + & b_{d-2,1}T_{d-2}(X) & + & b_{d-1,1}T_{d-1}(X)] \\
 & = & b_{0,0}T_0(X) & + & b_{1,0}T_1(X) & + \ldots + & b_{d-1,0}T_{d-1}(X) & + & b_{d,0}T_d(X)
\end{array}$$

In order to get the relation between each row of $(b_{i,j})_{i\in[\![0,d-j]\!],j\in[\![0,d]\!]}$, we write the polynomial at the j^{th} and the $(j+1)^{\text{th}}$ step.

$$P(X) = \sum_{k=0}^{j-1} a_k X^k + X^j \sum_{k=0}^{d-j} b_{k,j} T_k(X) \tag{4}$$

$$P(X) = \sum_{k=0}^{j} a_k X^k + X^{j+1} \sum_{k=0}^{d-j-1} b_{k,j+1} T_k(X) \tag{5}$$

From Equation (2), we get the identities

$$\begin{aligned}
XT_k(X) &= \frac{1}{2}[T_{k+1}(X) + T_{k-1}(X)], \text{ for } k \geq 1, \\
XT_0(X) &= T_1(X) = X.
\end{aligned} \tag{6}$$

Then, Equations (5) and (6) give

$$\begin{aligned}
P(X) =& \sum_{k=0}^{j} a_k X^k + X^j b_{0,j+1} T_1(X) + \frac{X^j}{2} \sum_{k=1}^{d-j-1} b_{k,j+1}[T_{k+1}(X) + T_{k-1}(X)], \\
P(X) =& \sum_{k=0}^{j} a_k X^k + X^j b_{0,j+1} T_1(X) + \frac{X^j}{2} \sum_{k\in\{0,1\}} b_{k+1,j+1} T_k(X) \\
&+ \frac{X^j}{2} \sum_{k=2}^{d-j-2} (b_{k+1,j+1} + b_{k-1,j+1}) T_k(X) + \frac{X^j}{2} \sum_{k\in\{d-j-1,d-j\}} b_{k-1,j+1} T_k(X).
\end{aligned} \tag{7}$$

By identification of Equations (4) and (7), we get the recurrence relations

$$\begin{aligned}
a_j + \frac{1}{2} b_{1,j+1} &= b_{0,j} \\
b_{0,j+1} + \frac{1}{2} b_{2,j+1} &= b_{1,j} \\
\frac{1}{2}[b_{k-1,j+1} + b_{k+1,j+1}] &= b_{k,j}, \text{ for } k \in [\![2, d-j-1]\!] \\
\frac{1}{2} b_{k-1,j+1} &= b_{k,j}, \text{ for } k \in \{d-j-1, d-j\}
\end{aligned} \tag{8}$$

For these relations to be defined for $j \in [\![0, d]\!]$, we impose $d \geq 1$ and define $b_{1,d+1} = 0$ and $b_{2,d} = 0$. The computation of the triangular matrix $(b_{i,j})_{i \in [\![0,d-j]\!], j \in [\![0,d]\!]}$ can be translated into Subroutine 1. It has two versions, one for real numbers and one for floating-point numbers, but they do exactly the same operations on their respective domains. For the floating-point numbers, $\mathbb{IF}$ denotes the set of floating-point intervals as explained in Section 8.6.

Subroutine 1 Conversion from the monomial to the Chebyshev basis.

Input: A polynomial $\sum_{i=0}^{d} a_i X^i$ in the monomial basis
Output: The polynomial in the Chebyshev basis, $\sum_{i=0}^{d} b_i T_i(X)$
Require: $d \geq 1$

```
1: function Real_MonomialToChebyshev(a)
2:     b_prev ∈ R^{d+1}, b_cur ∈ R^d
3:     b_prev ← (0, ..., 0)
4:     for j ← d to 0 do
5:         b_cur[0] ← a[j] + b_prev[1]/2
6:         b_cur[1] ← b_prev[0] + b_prev[2]/2
7:         for k ← 2 to d − j do
8:             b_cur[k] ← (b_prev[k − 1] + b_prev[k + 1])/2
9:         end for
10:        b_prev[0 : d] ← b_cur
11:    end for
12:    return b_cur
13: end function
```

```
1: function Float_MonomialToChebyshev(a)
2:     b_prev ∈ IF^{d+1}, b_cur ∈ IF^d
3:     b_prev ← (0, ..., 0)
4:     for j ← d to 0 do
5:         b_cur[0] ← a[j] + b_prev[1]/2
6:         b_cur[1] ← b_prev[0] + b_prev[2]/2
7:         for k ← 2 to d − j do
8:             b_cur[k] ← (b_prev[k − 1] + b_prev[k + 1])/2
9:         end for
10:        b_prev[0 : d] ← b_cur
11:    end for
12:    return b_cur
13: end function
```

Lemma 0.2.2. *The number of operations of Subroutine 1 (*Real_MonomialToChebyshev *or* Float_MonomialToChebyshev*) is* $O(d^2)$.

Proof. The algorithms is composed of an outer loop from Lines 4 to 11 and an inner loop from Lines 7 to 9. In the inner loop, an addition and a division are performed $d - j - 1$ times. In the outer loop, an addition and a division are performed twice, then the inner loop is called, finally an affectation is made. So, in that outer loop $O(d - j)$ arithmetic operations are performed for $j \in [\![0, d]\!]$. Hence, the complexity of the algorithm is $O(d^2)$. □

It is possible to improve the space complexity of this algorithm by a clever ordering of the operations which gets rid of $\mathbf{b_{prev}}$ [PTVF07, Section 5.11]. Let us discuss the bit complexity of the change of basis. Given a polynomial of degree d in the monomial basis, computing its coefficients in the Chebyshev basis can be done by using Subroutine 1 (Float_MonomialToChebyshev). The latest bound on the multiplication of two p-bit integers is $\mathsf{M}(p) = O(p \log(p))$ [HvdH21], it is conjectured to be the best bound. In radix-2 precision-p floating-point arithmetic, the complexity of the multiplication is $O(\mathsf{M}(p))$. Thus, the change of basis can be done in $O(d^2 p \log(p))$ bit operations in precision-p arithmetic. Remark that it is also possible to do it in $O(d \log(d))$ arithmetic operations [BSS08; Pan98], however those methods are based on Taylor shift operations that require a number of bit operations quadratic in d [vzGG97]. More precisely, we could get the change of basis in $O(\mathsf{M}(d^2 \log(d) + dp))$ for polynomials with integer coefficients of bit-length at most p. The problem of the change of basis has been studied for many families of polynomials, not only between those bases and the monomial basis, but also between the different families and it generalizes to the study of power series manipulation[BK78; vdH02; Pan98; Ger00; BES05; PST98; DJR97; YX98; BP04; AR91; Kei09].

3 The Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is widely used in signal processing as a compression tool. It has four variants. When the DCT is mentioned without any precision, it refers to the DCT-II. The DCT can be used to perform a change of basis toward the Chebyshev basis. For a given $n \in \mathbb{N}^*$, the DCT computes the set $(X_i)_{i\in[\![0,n-1]\!]}$ from the set $(x_i)_{i\in[\![0,n-1]\!]}$ using the following formula.

$$\forall k \in [\![0, n-1]\!], X_k = \sum_{i=0}^{n-1} x_i \ \cos\left[\frac{\pi(2i+1)k}{2n}\right] \tag{9}$$

4 The Inverse Discrete Cosine Transform

The inverse transformation to the DCT-II is the DCT-III. It is often called the IDCT.

Definition 0.4.1. Given $d, N \in \mathbb{N}^*$ with $d+1 \leq N$ and $(X_i)_{i\in[\![0,d]\!]}$ a sequence of real numbers, the Inverse Discrete Cosine Transform $\mathrm{IDCT}((X_i)_{i\in[\![0,d]\!]}, N)$ is the sequence $(x_k)_{k\in[\![0,N-1]\!]}$ defined by

$$\forall k \in [\![0, N-1]\!],\ x_k = \frac{1}{N}\left(\frac{1}{2}X_0 + \sum_{i=1}^{d} X_i \cos\left(\frac{i(2k+1)}{2N}\pi\right)\right). \tag{10}$$

When input and output sizes match, it is omitted: $\mathrm{IDCT}((X_i)_{i\in[\![0,d]\!]})$ denotes $\mathrm{IDCT}((X_i)_{i\in[\![0,d]\!]}, d+1)$.

5 Evaluating a polynomial

Horner's method or *Horner's scheme* is an efficient algorithm for polynomial evaluation. It is optimal for the evaluation of an arbitrary polynomial of degree d (evaluation in monomial form and no preconditioning). It relies on the Horner's rule:

$$\begin{aligned}\sum_{r=0}^{d} a_r X^r &= a_0 + a_1X + a_2X^2 + a_3X^3 + \cdots + a_{d-1}X^{d-1} + a_dX^d \\ &= a_0 + X\left(a_1 + X\left(a_2 + X\left(a_3 + \cdots + X\left(a_{d-1} + X\,a_d\right)\cdots\right)\right)\right).\end{aligned}$$

In order to evaluate $P(X) = \sum_{r=0}^{d} a_r X^r$ at x, one has to compute the recursive sequence defined by

$$\begin{aligned} b_d &= a_d \\ b_{r-1} &= a_{r-1} + b_r x \end{aligned}$$

The evaluation $P(x)$ is equal to b_0. Computing it requires only d additions and d multiplications. This is the method used for the evaluation of polynomials.

Lemma 0.5.1. *The evaluation of a polynomial of degree d has an arithmetic complexity of $O(d)$.*

Remark. Horner's method is a special case of the *Clenshaw algorithm* or *Clenshaw summation*, which computes the weighted sum of a finite series of functions $\phi_k(x)$, $S(x) = \sum_{k=0}^{n} a_k\phi_k(x)$, when they verify a three-term linear recurrence. Horner's method is a method to evaluate polynomials in the monomial form, other methods exist for other forms.

6 Random polynomial families

Random univariate polynomials are of the form

$$P(x) = \sum_{r=0}^{d} a_r \xi_r x^r$$

where $(a_r)_{r\in[\![0,d]\!]} \in \mathbb{C}^{d+1}$ and ξ_0, ξ_1, ... are independent identically distributed (i. i. d.) random variables. For some choices of the coefficients a_i, polynomials are well-studied (see [KZ14], [NV22] and references therein). Among those families, one finds

- Kac polynomials with $a_r = 1$,
- elliptic polynomials with $a_r = \sqrt{\binom{d}{r}}$,
- flat polynomials with $a_r = \frac{1}{\sqrt{r!}}$.

When d increases, representing flat polynomials with fixed precision becomes difficult. Therefore, we do not consider this family. The other two families extend to higher dimensions as follows.

$$P(x,y) = \sum_{r=0}^{d}\sum_{s=0}^{d} a_{r,s} \xi_{r,s} x^r y^s$$

where $(a_{i,j})_{i,j\in[\![0,d]\!]^2} \in \mathbb{C}^{(d+1)^2}$ and ξ_0, ξ_1, ... are i. i. d. random variables. The families are

- Kac polynomials with $a_{i,j} = 1$,
- Kostlan-Shub-Smale (KSS) polynomials with $a_{r,s} = \sqrt{\frac{d!}{r!s!(d-r-s)!}}$ [SS93; EK95; Kos02].

Implicit curves defined by these bivariate random polynomials are used for the experiments in Chapter 3 of Part I.

7 Binary trees

There are some variations in the terminology of trees in computer science. Here, we clarify our choices for binary trees and illustrate some notions with Figure 2, as this data structure appears naturally in subdivision processes (see Section 1.2.3 of Part I).

A binary tree has a *root* node and each node has at most two *children* node, the left child and the right child. A node which has a child is called the child's *parent* node. An *internal node* is a node that has at least a child. A *leaf* node is a node that does not have any children.

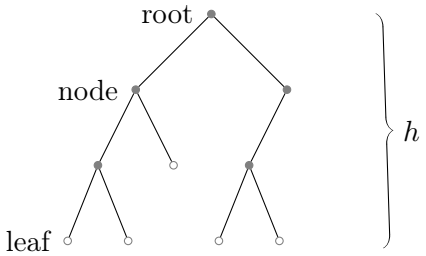


Figure 2: A binary tree.

A node is connected to its child or its parent by an *edge*. The *distance* is the number of edges along the shortest path between two nodes. The *depth* of a node is the distance between the node and the root node. The *height* h of a tree is the length of the longest path from the root to a leaf. Consequently, a tree with one node has a height of 0.

We denote by T the *size* of a tree, that is the numbers of nodes (internal nodes and leaves). A perfect binary tree is a tree in which all internal nodes have exactly two children and all leaves have the same depth. A perfect binary tree of height h has $T = 2^{h+1} - 1$ nodes.

8 Interval arithmetic

Interval arithmetic is used to compute the enclosure of sets of solutions to computational problems. Algorithms can rely on interval arithmetic to handle accumulated rounding errors, approximation errors or propagated uncertainties for example. Since Moore's book on the subject, notations and definitions have evolved [MKC09; BKY09]. The following sections present the definitions that we have selected, in particular in Section 8.6.

8.1 Definition of an interval

We denote by $\mathbb{R}$ the set of real numbers. For $a, b \in \mathbb{R}$ with $a \leq b$, we denote a *closed interval* by $[a, b]$. It is the set of real numbers x which verify $a \leq x \leq b$. In the rest of the document, we refer to *closed intervals* simply as *intervals*. The set of intervals over $\mathbb{R}$ is denoted by $\mathbb{IR}$.

When dealing with a value v not known or not represented exactly, its range is bounded above and below:

$$v_l \leq v \leq v_u.$$

In interval arithmetic, we represent the inequalities with an interval:

$$v \in [v_l, v_u].$$

We do not say that v is bounded by v_l and v_l, but that v lies in the interval $[v_l, v_u]$. This interval is called an *enclosure* of v. Even though the notations are equivalent, we prefer the second one. This allows us to ignore the inequalities and work directly with the bounds.

The lower and upper bounds of an interval I are denoted $\underline{I}$ and $\overline{I}$. So,

$$I = [\underline{I}, \overline{I}].$$

An interval I is *degenerate* if $\underline{I} = \overline{I}$. Then, it contains exactly one real value v. We identify the real number v with the degenerate interval $[v, v]$. In particular, $0 \cdot I = [0, 0] \cdot I$ and $1 \cdot I = [1, 1] \cdot I$.

8.2 Basic interval arithmetic operations

We now define the basic arithmetic operations in $\mathbb{IR}$: the sum, the difference, the product and the quotient. Given that intervals are sets, we can simply define these four arithmetic operations.

Table 1: Interval multiplication $I \cdot J$.

	$J \geq 0$	$0 \in J$	$J \leq 0$
$I \geq 0$	$[\underline{I}\,\underline{J}, \overline{I}\,\overline{J}]$	$[\overline{I}\,\underline{J}, \overline{I}\,\overline{J}]$	$[\overline{I}\,\underline{J}, \underline{I}\,\overline{J}]$
$0 \in I$	$[\underline{I}\,\overline{J}, \overline{I}\,\overline{J}]$	$[\min(\underline{I}\,\overline{J}, \overline{I}\,\underline{J}), \max(\underline{I}\,\underline{J}, \overline{I}\,\overline{J})]$	$[\overline{I}\,\underline{J}, \underline{I}\,\underline{J}]$
$I \leq 0$	$[\underline{I}\,\overline{J}, \overline{I}\,\underline{J}]$	$[\underline{I}\,\overline{J}, \underline{I}\,\underline{J}]$	$[\overline{I}\,\overline{J}, \underline{I}\,\underline{J}]$

Definition 0.8.1. For $I, J \in \mathbb{IR}$, we respectively define the sum, the difference, the product and the quotient of two intervals,

$$\begin{aligned} I + J &= \{x + y \,|\, x \in I, y \in J\}, \\ I - J &= \{x - y \,|\, x \in I, y \in J\}, \\ I \cdot J &= \{xy \,|\, x \in I, y \in J\}, \\ I/J &= \{x/y \,|\, x \in I, y \in J\} \text{ when } 0 \notin J. \end{aligned}$$

Actually, we want to have a way to do these operations without computing the sets. We want formulas on intervals to compute the arithmetic operations. The addition and the subtraction are straightforward:

$$\begin{aligned} I + J &= [\underline{I} + \underline{J}, \overline{I} + \overline{J}], \\ I - J &= [\underline{I} - \overline{J}, \overline{I} - \underline{J}]. \end{aligned}$$

The multiplication is given by the minimum and the maximum of four products:

$$I \cdot J = [\min S, \max S] \text{ where } S = \{\underline{I}\,\underline{J}, \underline{I}\,\overline{J}, \overline{I}\,\underline{J}, \overline{I}\,\overline{J}\}.$$

By testing the sign of $\underline{I}$, $\overline{I}$, $\underline{J}$ and $\overline{J}$, we can reduce the number of multiplications. Refer to Table 1 to see the nine cases. Then the only case which requires four products is the one where I and J both contain 0.

For the division I/J, which is defined when $0 \notin J$, we can use the multiplication, since $x/y = x \cdot (1/y)$. So,

$$I/J = I \cdot (1/J)$$

where

$$1/J = \{y' | 1/y' \in J\} = [1/\overline{J}, 1/\underline{J}]].$$

8.3 Algebraic properties

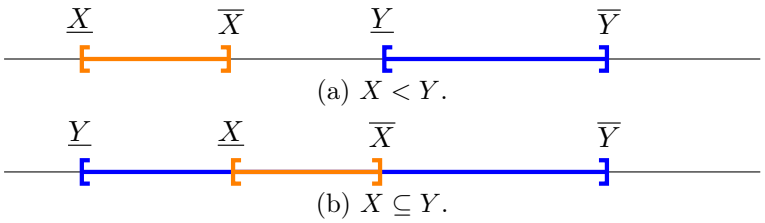
Addition and multiplication both verify the associativity, the commutativity and the existence of a neutral element.

$(I + J) + K = I + (J + K)$	$(IJ)K = I(JK)$	associativity
$I + J = J + I$	$IJ = JI$	commutativity
$0 + I = I$	$1 \cdot I = I$	neutral element

8.4 Order relations

For intervals, we have two partial ordering relations, which are both transitive. They are illustrated by Figure 3.

$$X < Y \iff \overline{X} < \underline{Y}$$
$$X \subseteq Y \iff \underline{Y} \leq \underline{X} \text{ and } \overline{X} \leq \overline{Y}$$



(a) $X < Y$.

(b) $X \subseteq Y$.

Figure 3: Illustration of interval order relations.

8.5 Magnitude

The magnitude $|I|$ of an interval I is the real-valued extension of the absolute value, defined as

$$|I| = \max_{x \in I} |x|.$$

8.6 From interval functions to box functions

Now that we have a basic arithmetic to handle intervals, we would like to define functions over intervals. An *interval function* is a function applied on intervals. If a function $f\colon \mathbb{R} \to \mathbb{R}$ only uses the four basic arithmetic operations, its formula can be applied to get its *natural interval extension* $F\colon \mathbb{IR} \to \mathbb{IR}$ which uses the formulas described in Section 8.2.

Example 0.8.1. If f is defined by $\forall x \in \mathbb{R},\ f(x) = 2x + 3$, then its natural interval extension F is defined by $\forall I \in \mathbb{IR},\ F(I) = 2I + 3$.

When it is defined, we have the inclusion $f(I) \subseteq F(I)$, where $f(I)$ is the image set of f over I and $F(I)$ is the interval function F evaluated at I. In some cases, we even have $f(I) = F(I)$. The tightness of the inclusion depends on the way the expression is written, as shown in Example 0.8.2. Indeed, interval arithmetic is memoryless and ignores the dependence of repeated occurrences of a variable.

Example 0.8.2. For the function f defined by $f(x) = x - x^2 = x\,(1 - x)$, we get the following two interval evaluations for the interval $[0, 2]$:

- $[0, 2] - [0, 2]^2 = [0, 2] - [0, 4] = [-4, 2]$,
- $[0, 2] \cdot (1 - [0, 2]) = [0, 2] \cdot [-1, 1] = [-2, 2]$.

Our motivation to use interval functions is to enclose domains which are not easy to compute. For any function f — not restricted to the four basic operations, we introduce the idea of a *box function* $\square f$ [BKY09]. For a function f we define a box function $\square f$ as a function which verifies the following two properties: inclusion and convergence. For an interval I, $f(I) \subseteq \square f(I)$

(inclusion). For a sequence of intervals $(I_i)_{i\in\mathbb{N}}$, if it is strictly decreasing ($I_0 \supset I_1 \supset I_2 \supset \dots$) with a point p as limit, then the limit of the images $\square f(I_i)$ is $f(p)$ (convergence).

We only handle polynomial functions. Thus, for a polynomial function P, we denote by $\square P(I)$ the evaluation on the interval I using Horner's scheme, whose evaluation order is defined in Section 5. First, we recall the subdistributivity law for intervals [Neu90, Section 1.5].

Proposition 0.8.1. *For $I, J, K \in \mathbb{IR}$:*

$$I(J+K) \subseteq IJ + IK.$$

With Corollary 1, we have a similar statement for polynomials, when they are evaluated with Horner's scheme. This evaluation method removes the ambiguity seen in Example 0.8.2. The gist of the proof relies on the evaluation order of the variable. On the one hand, the polynomial P has a variable occurring several times :

$$P(X) = a_0 + X\,(a_1 + X\,(a_2 + X\,(a_3 + \dots + X\,(a_{d-1} + X\,a_d)\cdots))).$$

On the other hand, the polynomial Q has d variables occurring once :

$$Q(X_1,\dots,X_d) = a_0 + X_1\,(a_1 + X_2\,(a_2 + X_3\,(a_3 + \dots + X_{d-1}\,(a_{d-1} + X_d\,a_d)\cdots))).$$

We take advantage of that by considering $P(X) = Q(X,\dots,X)$.

Corollary 1. *For three polynomials P, P_1, P_2, and an interval I,*

$$P = P_1 + P_2 \implies \square P(I) \subseteq \square P_1(I) + \square P_2(I).$$

Proof. Let d be the degree of P, P_1 and P_2, without loss of generality. We give a proof by induction on d.
Base case: P, P_1 and P_2 are constant polynomials such that $P = P_1 + P_2$. Thus, $P(I) = \square P(I)$, $P_1(I) = \square P_1(I)$ and $P_2(I) = \square P_2(I)$. Subsequently,

$$\square P(I) = P(I) \subseteq P_1(I) + P_2(I) = \square P_1(I) + \square P_2(I).$$

Induction step: assume that for degree at most d polynomials Q, Q_1 and Q_2,

$$Q = Q_1 + Q_2 \implies \square Q(I) \subseteq \square Q_1(I) + \square Q_2(I).$$

Let P be a degree at most $d+1$ polynomial. It can be written as

$$\begin{aligned} P(X) &= P_1(X) + P_2(X) \\ &= \underbrace{XQ_1(X) + q_{0,1}}_{P_1(X)} + \underbrace{XQ_2(X) + q_{0,2}}_{P_2(X)} \\ &= XQ(X) + q_0, \end{aligned}$$

where Q, Q_1 and Q_2 are degree at most d polynomials, and q_0, $q_{0,1}$ and $q_{0,2}$ are intervals. Moreover, $Q = Q_1 + Q_2$ and $q_0 = q_{0,1} + q_{0,2}$. We consider the box function of this polynomial.

$$\begin{aligned} \square P(I) &= I\square Q(I) + q_0 && \text{by application of Horner's scheme,} \\ &\subseteq I\,(\square Q_1(I) + \square Q_2(I)) + q_0 && \text{by induction hypothesis.} \end{aligned}$$

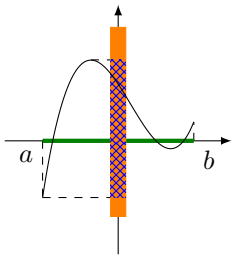


Figure 4: Illustration of $[a, b]$ (green), $P([a, b])$ (blue) and $\square P([a, b])$ (orange).

In Horner'scheme, $\square Q_1(I)$ and $\square Q_2(I)$ are evaluated before the multiplication by I. So, we can apply Proposition 0.8.1.

$$\begin{aligned} \square P(I) &\subseteq I\square Q_1(I) + q_{0,1} + I\square Q_2(I) + q_{0,2} && \text{by subdistributivity with Horner's scheme,} \\ &\subseteq \square P_1(I) + \square P_2(I) && \text{by application of Horner's scheme.} \end{aligned}$$

□

Dyadic numbers $\mathbb{D} = \{m2^{-n} \,|\, m \in \mathbb{Z}, n \in \mathbb{N}\} = \mathbb{Z}\left[\frac{1}{2}\right]$, also called bigfloats, form a good abstraction for floating-point numbers, the latter being a finite subset of the dyadic numbers [YY09]. We can reduce interval arithmetic to computations over $\mathbb{D}$ and the definition of a box function still holds. Even though interval arithmetic is defined for real numbers, it is now widely used as a tool for the computation analysis on devices working in precision p. In practice for finite precision arithmetic, we perform interval arithmetic over floating-point numbers — a finite subset of $\mathbb{D}$ —, so the definition of the box function does not hold. In the IEEE 754 double-precision binary floating-point format (binary64), the set is more precisely $\mathbb{F} = \{m2^{-n} \,|\, m \in [\![-2^{53}, 2^{53}]\!], n \in [\![-1022, 1023]\!]\}$. Nonetheless, we still use the box function notation simply for the inclusion property and because it is easier to compute than the set we are actually interested in. Figure 4 summarizes this relation. We do interval arithmetic on floating-point numbers and denote their set $\mathbb{IF}$.

8.7 Ball arithmetic

Ball arithmetic is a variant of interval arithmetic also known as midpoint-radius interval arithmetic [vdH09]. An interval $[x, y]$ is defined by two endpoints $x < y$ in a totally ordered set R, such that

$$[x, y] = \{z \in R \,|\, x \leq z \leq y\}.$$

A ball $\mathcal{B}(c, r)$ is defined by a center or midpoint c in a normed vector space V over a totally ordered field R and by a radius $r \geq 0$, such that

$$\mathcal{B}(c, r) = \{x \in V \,|\, \|x - c\| \leq r\}.$$

Although balls and intervals are usually interchangeable, ball arithmetic cannot represent intervals like $[1, +\infty[$. Nonetheless, ball arithmetic corresponds to ϵ-δ-calculus and is convenient to represent a value and its uncertainty. Ball arithmetic has been implemented in the Arb library[4] and the `numerix` package of MATHEMAGIX.[5]

[4] `https://arblib.org/`
[5] `http://www.mathemagix.org`

Part I

Curve drawing in $\mathbb{R}^2$

1

Algorithms using the real RAM model

In this chapter, we introduce ideas to improve upon the Marching Squares algorithm. We aim at constructing a drawing with a better algorithmic complexity, while identifying at least the same intersections with the grid. This leads us to Algorithm 4. The goal of this chapter is to get to the structure of this algorithm, therefore we work with real numbers and introduce ideas one by one. In the next chapter, the main ideas will be adapted in order to work with finite precision arithmetic.

1.1 Representing a curve on a fixed grid

We are interested in the drawing of an implicit plane curve defined by a polynomial, in other words we are interested in a curve defined by the implicit equation $P(x, y) = 0$ where P is a bivariate polynomial of partial degree d. By abuse of terminology, we call such a curve a curve of degree d.

We call a *two-dimensional grid* two sets of parallel lines which intersect at a right angle. We call those lines *fibers* of the grid. The intersections are called *vertices*. The segment connecting two adjacent vertices along a line is called an *edge*. A *pixel* is a rectangle whose corners are vertices and which does not contain any vertex inside. These elements are illustrated by Figure 1.1. We define a grid by its sets of lines. So, an $N \times N$ grid has N^2 vertices and $(N-1)^2$ pixels. We call a *drawing* a subset of the pixels of the grid.

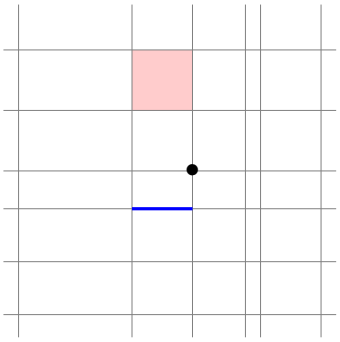


Figure 1.1: A grid in gray, a vertex in black, an edge in blue and a pixel in light red.

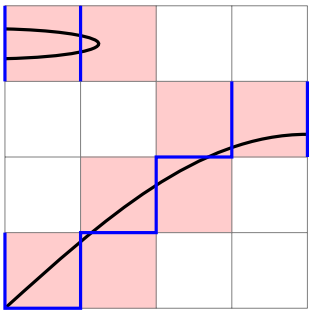
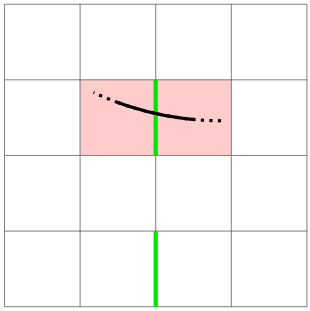
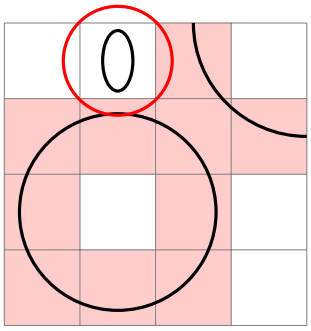


Figure 1.2: A correct pixel drawing and a correct edge drawing of the black curve: pixels are lighted in light red if and only if they intersect the curve, edges are lighted in blue if and only if they intersect the curve.



(a) Pixels are lighted only when an intersection of the curve with an edge is detected.



(b) A piece of the curve (circled in red) inside a false negative pixel.

Figure 1.3: Presence of a false negative pixel if the curve does not intersect the grid.

The *correct pixel drawing* of a curve on a fixed grid corresponds to the subset of pixels of that grid that exactly covers the curve. Each pixel in that set intersects the curve and each pixel not in that set does not intersect the curve. Similarly, the *correct edge drawing* of a curve on a fixed grid corresponds to the subset of edges of that grid that exactly intersect the curve. Each edge in that set intersects the curve and each edge not in that set does not intersect the curve. Figure 1.2 displays the correct pixel drawing and the correct edge drawing with a curve in black, a 5×5 grid, the set of pixels corresponding to the drawing lighted in light red and the set of edges in blue.

The algorithms presented in this chapter detect the **intersection of the curve with edges** of the grid and return the corresponding segments. Pixels intersecting the curve are not directly detected. If an intersection of an edge with the curve is detected the adjacent pixels are lighted and if no intersection is detected no pixel is lighted, as illustrated by Figure 1.3a. Some connected components of the curve are not detected that way. If they lie inside a pixel, they do not intersect its edges and therefore the pixel is not lighted, resulting in a false negative. In Figure 1.3b, such a pixel is circled in red.

The main algorithm of this chapter — Algorithm 4 — is meant to compute a drawing on a grid defined by Chebyshev nodes, that is an $N \times N$ grid with lines verifying $x = c_i$ or $y = c_i$, where the $(c_i)_{[\![0,N-1]\!]}$ are Chebyshev nodes for $N \in \mathbb{N}$. See Figure 1.4 for such a grid. We also assume that d is much smaller than N. For instance, we draw a degree 100 curve on a 1000×1000 grid.

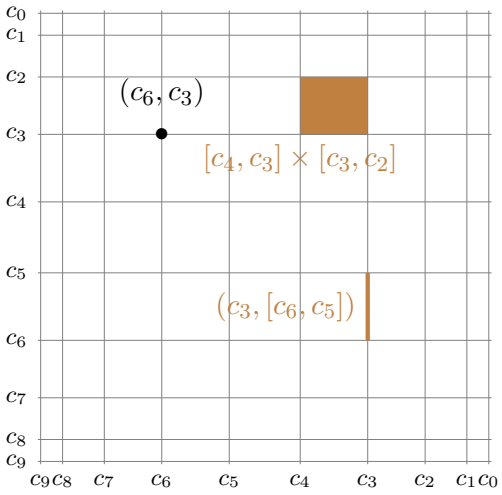


Figure 1.4: Chebyshev grid for $N = 10$, vertical segment $(c_3, [c_6, c_5])$ and pixel $[c_4, c_3] \times [c_3, c_2]$.

1.2 Marching squares-like algorithms

1.2.1 Naive algorithm

A naive way to detect intersections of the curve with the grid would be to evaluate the polynomial on each vertex. If the evaluation on two adjacent vertices have different signs or if one of them is zero, that means that the curve intersects the edge at least once.

Algorithm 1 describes this naive approach. For each point (x_i, y_j) of an $N \times N$ grid, it evaluates the polynomial $P(X, Y) = \sum_{r=0}^{d} \sum_{s=0}^{d} a_{r,s} X^r Y^s$, first with respect to the X variable using Horner's method at x_i. This results in a univariate polynomial $P(x_i, Y)$. It is then evaluated at y_j with the same method. The algorithm returns vertical segments which have an evaluation with different signs at each vertex of the edge or where at least one is equal to zero.

Theorem 1.2.1. *The number of arithmetic operations of Algorithm 1 (*Real_NaiveEvaluation*) is* $O(N^2 d^2)$.

Proof. In Lines 3 to 10, two loops are nested in order to evaluate $P(X, Y) = \sum_{s=0}^{d} \sum_{r=0}^{d} a_{r,s} X^r Y^s$ at the N^2 vertices of the grid. The evaluation of P at (x_i, y_j) corresponds first to its evaluation in X then in Y. The evaluation in X is the evaluation of the $d + 1$ polynomials of degree d, $\sum_r a_{r,s} X^r$. Those evaluations contributes to a complexity of $O(d^2)$ using Horner's method. Then, the resulting $(b_s)_{s \in [\![0,d]\!]}$ are used to evaluate another polynomial of degree d, $\sum_{s=0}^{d} b_s Y^s$. So, the complexity becomes $O(d^2 + d)$. Thus, the complexity for the evaluation of all the vertices is $O(N^2(d^2 + d)) = O(N^2 d^2)$. Then, those evaluations are used to detect intersections of the grid with the curve. In order to perform a test and keep the edges where an intersection is detected (Lines 14 to 16), one multiplication for each of the $N(N - 1)$ vertical edges is required. Thus, the total complexity is $O(N^2 d^2 + N^2) = O(N^2 d^2)$. □

The algorithm returns only vertical segments, but Lines 14 to 16 can be adapted to also return horizontal segments, without affecting the global complexity. Note that in the proof, the complexity due to the evaluation is $O(N^2 d^2)$ and the complexity due to the collection of the returned edges is $O(N^2)$. So, the cost of the evaluation dominates. In the next sections, we present ideas

Algorithm 1 Naive evaluation.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{r,s} X^r Y^s$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$ and two sequences of coordinates $\mathbf{x} = \{x_0, \ldots, x_{N-1}\}$ and $\mathbf{y} = \{y_0, \ldots, y_{N-1}\}$ from $\mathbb{R}$ such that $x_0 < \cdots < x_{N-1}$ and $y_0 < \cdots < y_{N-1}$

Output: The set of vertical edges $\mathcal{E}$ such that for each one of the form $(x_i, [y_j, y_{j+1}])$, $P(x_i, y_j) \cdot P(x_i, y_{j+1}) \leq 0$

1: **procedure** REAL_NAIVEEVALUATION($\mathbf{a}$, $\mathbf{x}$, $\mathbf{y}$)
2: $\quad\mathbf{b} \in \mathbb{R}^{d+1}, \mathbf{val} \in \mathbb{R}^{N\times N}$
3: $\quad$**for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Evaluation on each point
4: $\quad\quad$**for** $j \leftarrow 0$ **to** $N-1$ **do**
5: $\quad\quad\quad$**for** $s \leftarrow 0$ **to** d **do**
6: $\quad\quad\quad\quad b_s \leftarrow \sum_{r=0}^{d} a_{r,s} x_i^r$ ▷ evaluation with respect to X using Horner's method
7: $\quad\quad\quad$**end for**
8: $\quad\quad\quad val_{i,j} \leftarrow \sum_{s=0}^{d} b_s y_j^s$ ▷ evaluation with respect to Y using Horner's method
9: $\quad\quad$**end for**
10: $\quad$**end for**
11: $\quad\mathcal{E} \leftarrow \emptyset$
12: $\quad$**for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Collection of the intervals
13: $\quad\quad$**for** $j \leftarrow 0$ **to** $N-2$ **do**
14: $\quad\quad\quad$**if** $val_{i,j} \cdot val_{i,j+1} \leq 0$ **then**
15: $\quad\quad\quad\quad$Add $(x_i, [y_j, y_{j+1}])$ to $\mathcal{E}$
16: $\quad\quad\quad$**end if**
17: $\quad\quad$**end for**
18: $\quad$**end for**
19: $\quad$**return** $\mathcal{E}$
20: **end procedure**

to decrease the number of evaluations necessary to construct a drawing and thus decrease the total complexity and also use a subdivision approach in order to consider less edges.

1.2.2 Algorithm with partial evaluation

We have just seen in Section 1.2.1 that the cost of the evaluation of the polynomial on each vertex of the grid dominates on the rest of the algorithm. Let us first simply reorder the loops to reduce the complexity from $O(N^2d^2)$ to $O(N^2d)$.

Idea 1

Instead of evaluating the polynomials $(P_s(X))_{s\in\{0,\dots,d\}} = \left(\sum_{r=0}^{d} a_{r,s}X^r\right)_{s\in\{0,\dots,d\}}$ for each point (x_i, y_j), they can be evaluated only once for each vertical fiber $x = x_i$. The result can then be reused for all the vertices on the corresponding fiber. Parentheses are added to the expression of the polynomial which becomes

$$P(X,Y) = \sum_{s=0}^{d}\left(\sum_{r=0}^{d} a_{r,s}X^r\right)Y^s. \tag{1.1}$$

Thus, in Algorithm 2 for each value x_i of X the polynomial P is partially evaluated with respect to X, then for each value y_j of Y the polynomial is evaluated with respect to Y. Finally, the algorithm returns vertical segments which have an evaluation with different signs at each end of the edge or where at least one is equal to zero.

Theorem 1.2.2. *The number of arithmetic operations of Algorithm 2 (*Real_PartialEvaluation*) is* $O(N^2d)$.

Proof. $P(X,Y) = \sum_{s=0}^{d}\left(\sum_{r=0}^{d} a_{r,s}X^r\right)Y^s$ is first partially evaluated for each vertical fiber, then for each horizontal fiber. For each of the N vertical fibers, the $d+1$ partial polynomials $(\sum_{r=0}^{d} a_{r,s}X^r)_{s\in[\![0,d]\!]}$ are evaluated. Those evaluations contributes to a complexity of $O(d^2)$. Then, for that same vertical fiber the results are used to evaluate another polynomial of degree d, $\sum_{s=0}^{d} b_sY^s$, on the N intersections between that vertical fiber and horizontal fibers. So, the complexity for one vertical fiber becomes $O(d^2 + Nd)$. Hence the complexity of the evaluation for all the vertical fibers is $O(N(d^2 + Nd)) = O(N^2d)$, since $d \ll N$. The collection of the edges is similar to the one of Algorithm 1. Thus, the total complexity is $O(N^2d + N^2) = O(N^2d)$. □

Like Algorithm 1, this one returns only vertical segments, but Lines 14 to 16 can be adapted to also return horizontal segments. The complexity is still dominated by the evaluation, but now it is linear in the degree d and quadratic in the resolution N.

1.2.3 Partial evaluation and subdivision algorithm

In Section 1.2.2, we got an algorithm with a complexity of $O(N^2d)$. The factor d comes from Horner's method which is optimal for the evaluation of an arbitrary polynomial. Let us then improve the factor N, which dominates d since we assume that $d \ll N$. To do so, we reduce the number of necessary evaluations.

Algorithm 2 Partial evaluation.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{r,s} X^r Y^s$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$ and two sequences of coordinates $\mathbf{x} = \{x_0, \ldots, x_{N-1}\}$ and $\mathbf{y} = \{y_0, \ldots, y_{N-1}\}$ from $\mathbb{R}$ such that $x_0 < \cdots < x_{N-1}$ and $y_0 < \cdots < y_{N-1}$

Output: The set of vertical edges $\mathcal{E}$ such that for each one of the form $(x_i, [y_j, y_{j+1}])$, $P(x_i, y_j) \cdot P(x_i, y_{j+1}) \leq 0$

1: **procedure** REAL_PARTIALEVALUATION(**a**, **x**, **y**)
2: $\mathbf{b} \in \mathbb{R}^{d+1}, \mathbf{val} \in \mathbb{R}^{N\times N}$
3: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Evaluation
4: **for** $k \leftarrow 0$ **to** d **do** ▷ partial evaluation
5: $b_s \leftarrow \sum_{r=0}^{d} a_{r,s} x_i^r$ ▷ evaluation with respect to X using Horner's method
6: **end for**
7: **for** $j \leftarrow 0$ **to** $N-1$ **do**
8: $val_{i,j} \leftarrow \sum_{s=0}^{d} b_s y_j^s$ ▷ evaluation with respect to Y using Horner's method
9: **end for**
10: **end for**
11: $\mathcal{E} \leftarrow \emptyset$
12: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Collection of the intervals
13: **for** $j \leftarrow 0$ **to** $N-2$ **do**
14: **if** $val_{i,j} \cdot val_{i,j+1} \leq 0$ **then**
15: Add $(x_i, [y_j, y_{j+1}])$ to $\mathcal{E}$
16: **end if**
17: **end for**
18: **end for**
19: **return** $\mathcal{E}$
20: **end procedure**

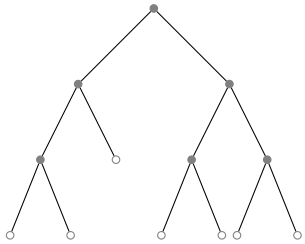


Figure 1.5: Binary tree properties: m leaves (gray circle) and $m-1$ internal nodes (gray disk).

Idea 2

In order to reduce the computation cost, zeros of the function can be found using a divide-and-conquer approach. It can be done by evaluating the function on a interval and then subdividing it if the function vanishes there.

In this section, we assume that we have an oracle $\mathcal{O}$ which, for a polynomial P of degree d and an interval $[\alpha, \beta]$, tells us whether $P([\alpha, \beta])$ contains zero or not with a complexity of $O(d)$. In other words, the oracle $\mathcal{O}$ has the same complexity as the evaluation $P(\gamma)$ of the polynomial P at a point γ. We use this oracle in Algorithm 3 to do the subdivisions. We present implementations to get as close as possible to what the oracle does for the finite precision algorithms in Chapter 2.

The subdivision tree

A partition of an interval $[\alpha, \beta]$ on the real line is a finite sequence $x_0, x_1, x_2, \ldots, x_n$ of real numbers such that $\alpha = x_0 < x_1 < x_2 < \cdots < x_n = \beta$. Every interval of the form $[x_i, x_{i+1}]$ is referred to as a subinterval of the partition.

We recall a properties of binary trees and explain what we call a *subdivision tree*. As illustrated in Figure 1.5 and stated by Proposition 1.2.3, a full binary tree with m leaves has $m-1$ internal nodes. Let us first look at the way the subdivision works on an example. Let $x_0 < x_1 < \cdots < x_7$ be vertices on a fiber. Figure 1.6 shows how the original interval is subdivided recursively throughout the process. The interval $[x_0, x_7]$ is subdivided into $[x_0, x_3]$ and $[x_3, x_7]$, which are respectively subdivided into $[x_0, x_1]$ and $[x_1, x_3]$, and $[x_3, x_5]$ and $[x_5, x_7]$. This continues until the resolution of the fiber is reached. This process can be represented by a binary tree, more specifically a full binary tree. We call such a tree a *subdivision tree* (Definition 1.2.1). If there are N nodes on the fiber, then there are $N-1$ edges or subintervals. Since those edges are represented by leaves of the subdivision tree, this means that this tree has $N-1$ leaves and $N-2$ internal nodes, for a total of $2N-3$ nodes.

Proposition 1.2.3. *A full binary tree with m leaves has $m-1$ internal nodes.*

Definition 1.2.1. A *subdivision tree* is the tree represented by all the subdivisions of an interval given by a set of ordered vertices $(x_i)_{i \in [\![0,N]\!]}$, such that $[x_i, x_j]$ gives $[x_i, x_k]$ and $[x_{k+1}, x_j]$ where $k = \left\lfloor \frac{i+j}{2} \right\rfloor$.

Such a tree represents all the possible evaluations in the subdivision process. However, the actual *evaluation tree* is a subtree of that subdivision tree, as defined by Definition 1.2.1. Let T be its size (number of nodes, leaves included). An example is given by Figure 1.7. If the evaluation

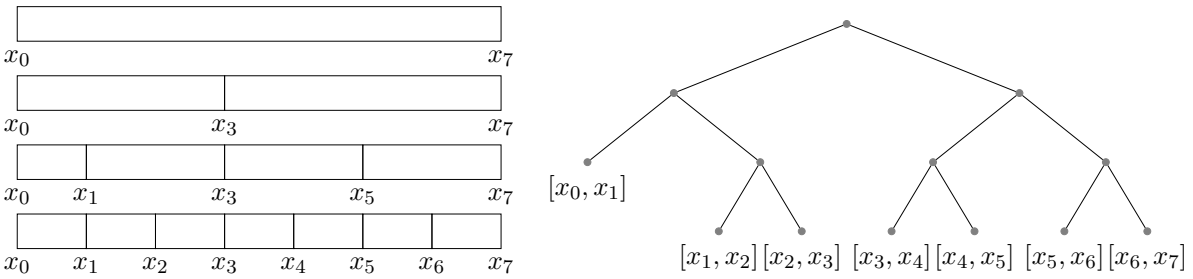


Figure 1.6: On the left hand side, from top to bottom, successive subdivision of the interval $[x_0, x_7]$. On the right hand side, binary tree representing the subdivision process.

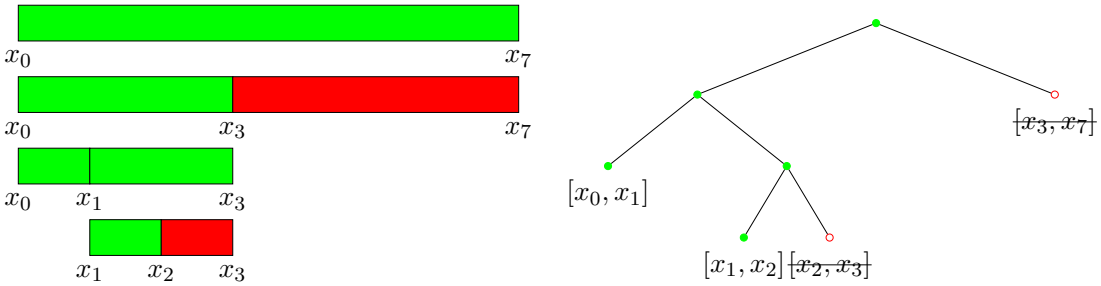


Figure 1.7: Example of an evaluation tree, with evaluations containing zero in green and evaluations containing no zero in red.

on an interval contains a zero, it is subdivided and the two subintervals are also tested. The subdivision stops either when the interval does not contain a zero or when the resolution of the fiber is reached and the interval can no longer be subdivided. We have to evaluate at least the initial interval and the size of the evaluation tree is bounded by the size of the subdivision tree. That gives the result of Lemma 1.2.4.

Definition 1.2.2. An *evaluation tree* is a subset of a subdivision tree, which represents the evaluations of a polynomial necessary to locate the zeros up to the given resolution, on the same set of subintervals. When the evaluation on an interval contains zero the corresponding node is colored in green. When it does not, the node is colored in red and none of its descendants is included in the evaluation tree.

Lemma 1.2.4. *An evaluation tree for N vertices has a size T which verifies $1 \leq T \leq 2N - 3$.*

Proposition 1.2.5. *An evaluation tree of a degree d polynomial for N vertices has a size $T = O(d \log_2(N))$.*

Proof. A polynomial of degree d has at most d real zeros. If it has no real root the subdivision tree is made of a root node only. Let us assume that there is at least one real root. These roots correspond to green leaves of the evaluation tree. Furthermore, each red leaf has a common ancestor with a green leaf. Indeed, by contradiction if a red leaf has no common ancestor with a green leaf, it means that even the root node is not a common ancestor. So, there is no branch from the root node to a green leaf, in other words there is no real root, which was assumed false. That means that there are at most as many red leaves as green leaves, that is $2d$. By counting the number of nodes in each branch to a leaf, one counts each node at least once. By construction a subdivision tree has a height of $\lceil \log_2(N) \rceil$. Thus, each branch of the evaluation

tree contains at most $\lceil \log_2(N) \rceil + 1$. So, there are at most $2d(\lceil \log_2(N) \rceil + 1)$ nodes. □

For the subdivision process, we rely on Subroutine 2, which implements a classical recursive algorithm. Despite that, it is not a usual root isolation algorithm: it does not compute disjoint intervals which contain exactly one root. Given a partition of an interval, it computes exactly the set of subintervals of that partition on which there is at least a root. It takes as input a univariate polynomial, the partition of the initial interval and an interval defined by indices of two vertices. With the oracle $\mathcal{O}$, we can test for an inclusion predicate and an exclusion predicate.

Subroutine 2 Isolation function with subdivision.

Input: A univariate polynomial function P, a set of vertices $x_0 < \cdots < x_{N-1}$ from $\mathbb{R}$ and two integer indices i and j

Output: The set of intervals $\mathcal{I}$ of the form $[x_k, x_{k+1}]$, which verify $0 \in P([x_k, x_{k+1}])$ and $[x_k, x_{k+1}] \subset [x_i, x_j]$

Require: $i < j$

```
1: function Real_Isolate1D(P, x, i, j)
2:     if P([x_i, x_j]) contains 0 then                ▷ Test using the oracle O
3:         if i + 1 < j then                           ▷ Split in two subintervals
4:             k = ⌊(i+j)/2⌋
5:             I_1 ← Real_Isolate1D(P, x, i, k)
6:             I_2 ← Real_Isolate1D(P, x, k + 1, j)
7:             I ← I_1 ∪ I_2
8:             return I
9:         else
10:            return {[x_i, x_j]}
11:        end if
12:    else
13:        return ∅
14:    end if
15: end function
```

Lemma 1.2.6. *The number of arithmetic operations of Subroutine 2 called on* Real_Isolate1D*(P,* **x***, 0, N) is $O(dT)$, when P is of degree d.*

Proof. When called on Real_Isolate1D(P, **x**, 0, N), the algorithm evaluates $P([x_0, x_N])$. If the evaluation contains zero, it subdivides $[x_0, x_N]$ and is called recursively on the subintervals $[x_0, x_k]$ and $[x_{k+1}, x_N]$ where $k = \lfloor \frac{0+N}{2} \rfloor$. The recursion is stopped when the evaluation does not contain zero or when the resolution is reached. In other words, the algorithm evaluates the polynomial exactly once on each interval represented by the evaluation tree. By definition, there are T intervals to test using the oracle $\mathcal{O}$ and each test has a complexity of $O(d)$. So, the algorithm uses $O(dT)$ operations. □

An algorithm doing partial evaluation followed by interval subdivision

Algorithm 3 does a partial evaluation of the polynomial and then uses the subdivision process we have just seen. Thanks to Subroutine 2, the partial polynomials are evaluated on intervals which are subdivided if they contain a zero. It also returns the edges intersected by the curve.

Algorithm 3 Edge enclosing with subdivision.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{r,s} X^r Y^s$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$ and two sequences of coordinates $\mathbf{x} = \{x_0, \dots, x_{N-1}\}$ and $\mathbf{y} = \{y_0, \dots, y_{N-1}\}$ from $\mathbb{R}$ such that $x_0 < \cdots < x_{N-1}$ and $y_0 < \cdots < y_{N-1}$

Output: The set of edges $\mathcal{E}$ of the form $(x_i, [y_j, y_{j+1}])$, which verify $0 \in P((x_i, [y_j, y_{j+1}]))$

1: **procedure** Real_EnclosingEgdes_PartialEval-Subd($\mathbf{a}$, $\mathbf{x}$, $\mathbf{y}$)
2: $\quad \mathbf{b} \in \mathbb{R}^{d+1}$
3: $\quad \mathcal{E} \leftarrow \emptyset$
4: $\quad$ **for** $i \leftarrow 0$ **to** $N-1$ **do**
5: $\quad\quad$ **for** $s \leftarrow 0$ **to** d **do** ▷ Partial evaluation
6: $\quad\quad\quad b_s \leftarrow \sum_{r=0}^{d} a_{r,s} x_i^r$
7: $\quad\quad$ **end for**
8: $\quad\quad Q : Y \longmapsto \sum_{s=0}^{d} b_s Y^s$
9: $\quad\quad \mathcal{I} \leftarrow$ Real_Isolate1D(Q, $\mathbf{y}$, 0, N) ▷ Subdivision
10: $\quad\quad$ **for** $I \in \mathcal{I}$, Add (x_i, I) to $\mathcal{E}$
11: $\quad$ **end for**
12: $\quad$ **return** $\mathcal{E}$
13: **end procedure**

Theorem 1.2.7. *The number of arithmetic operations of Algorithm 3 (*Real_EnclosingEgdes_PartialEval-Subd*) is $O(Nd(d + T_{max}))$, where T_{max} is the size of the largest evaluation tree in the process.*

Proof. Similarly to Algorithm 2, for each of the N vertical fibers the $d+1$ partial polynomials $(\sum_{r=0}^{d} a_{r,s} X^r)_{s \in [\![0,d]\!]}$ are evaluated. Those evaluations contributes to a complexity of $O(d^2)$. Then, Real_Isolate1D is called on the polynomial $\sum_{s=0}^{d} b_s Y^s$. So, from Lemma 1.2.6 the complexity becomes $O(d^2 + dT)$. Hence the complexity of the evaluation for all the fibers is $O(N(d^2 + dT_{max}))$, where T_{max} is the size of the largest evaluation tree over all the vertical fibers. □

For each vertical fiber, the subdivision tree has a size T. So, the largest size T_{max} also verifies $T_{max} = O(d \log_2(N))$, from Corollary 1.2.5. Therefore, the improvement of the previous $O(N^2 d)$ depends on T_{max}. Nonetheless, we have at most $O(d^2 N \log_2(N))$.

Note. In an algorithm which applies a subdivision process along multiple fibers like Algorithm 3, T_{max} is the size of the largest evaluation tree over all of them. We also apply this notation to algorithms using subdivision along stripes and tubes.

Contrary to the previous algorithms, this one can only return vertical edges. In order to get the horizontal edges, one has to call it by swapping the roles of $\mathbf{x}$ and $\mathbf{y}$. In other words, call Real_EnclosingEgdes_PartialEval-Subd($\mathbf{a}$,$\mathbf{y}$,$\mathbf{x}$) and get the output $\mathcal{S}$. Then, one should actually consider $\mathcal{S}' = \{([b,c], a) \,|\, (a, [b,c]) \in \mathcal{S}\}$, which is the set of horizontal edges one is looking for.

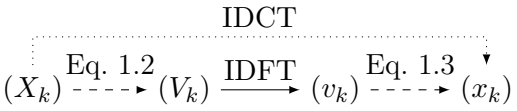


Figure 1.8: Fast IDCT procedure with an IDFT on N real points.

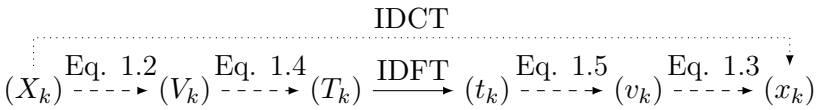


Figure 1.9: Fast IDCT procedure with an IDFT on $N/2$ complex points.

1.2.4 Fast Multipoint Evaluation

This section explains how the fast IDCT is computed by reduction to the IDFT [Mak80]. Direct evaluation methods also exist, but in the implementation of our algorithms we use the IDCT from SciPy which implements the method described in this section.[6]

Reduction of the IDCT to N-point IDFT

The goal is to reduce the computation of (x_k), the IDCT of (X_k), to the computation of (v_k), the IDFT of (V_k) (Figure 1.8).

Let $\omega_M = e^{-j2\pi/M}$. Given $(X_k)_{k\in[\![0,N-1]\!]}$ and $X_N = 0$, (V_k) is defined by

$$V_k = \frac{1}{2}\omega_{4N}^{-k}\left[X_k - jX_{N-k}\right],\ 0 \leq k \leq N-1, \tag{1.2}$$

and (x_k) is retrieved from

$$\begin{cases} x_{2k} = v_k, & 0 \leq k \leq \lfloor \frac{N-1}{2} \rfloor, \\ x_{2k+1} = v_{N-k-1}, & 0 \leq k \leq \lfloor \frac{N}{2} \rfloor - 1. \end{cases} \tag{1.3}$$

Reduction of the IDCT to $(N/2)$-point IDFT

We can further reduce the size of the sequence computed through IDFT. Since (V_k) is a Hermitian symmetric sequence, the number of points for the IDFT can be divided by 2 (Figure 1.9).

(T_k) is computed following the flow graph in Figure 1.10, for $0 \leq k \leq \lfloor N/4 \rfloor$, where $\overline{\bullet}$ denotes the complex conjugate (see Figure 1.10):

$$\begin{aligned} T_k &= \frac{1}{2}\left[\left(V_k + \overline{V_{\frac{N}{2}-k}}\right) + j\omega_N^{-k}\left(V_k - \overline{V_{\frac{N}{2}-k}}\right)\right], \\ \overline{T_{\frac{N}{2}-k}} &= \frac{1}{2}\left[\left(V_k + \overline{V_{\frac{N}{2}-k}}\right) - j\omega_N^{-k}\left(V_k - \overline{V_{\frac{N}{2}-k}}\right)\right]. \end{aligned} \tag{1.4}$$

The (N/2)-point IDFT of (T_k) gives (t_k). The sequence (v_k) is obtained thanks to

$$\begin{aligned} v_{2k} &= Re(t_k) \\ v_{2k+1} &= Im(t_k) \end{aligned} \tag{1.5}$$

[6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.dct.html

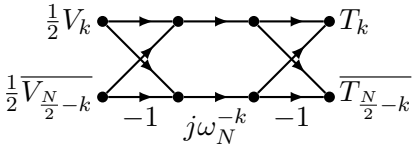


Figure 1.10: Flow graph to compute (T_k) from (V_k).

Using the IDCT **for fast polynomial multipoint evaluation**

Let P be a polynomial of degree d given in the Chebyshev basis $P(x) = \sum_{i=0}^{d} \alpha_i T_i(x)$. Using Eq. (1), the evaluation of P at a Chebyshev node c_k of T_N for $N > d$ satisfies

$$\begin{aligned}\forall k \in [\![0, N-1]\!],\ P(c_k) &= \sum_{i=0}^{d} \alpha_i\, T_i\left(\cos\left(\frac{2k+1}{2N}\pi\right)\right) \\ &= \sum_{i=0}^{d} \alpha_i\, \cos\left(\frac{i(2k+1)}{2N}\pi\right), \text{ with Eq. (1)} \\ &= \frac{\alpha_0}{2} + \left[\frac{\alpha_0}{2} + \sum_{i=1}^{d} \alpha_i\, \cos\left(\frac{i(2k+1)}{2N}\pi\right)\right].\end{aligned}$$

Thus, the IDCT can be used to evaluate a function P simultaneously on a discrete set of N points in a number of operations quasi-linear in N. If P is a polynomial of degree d and $N \geq d$,

$$(P(c_i))_{i\in[\![0,N-1]\!]} = \frac{1}{2}(\alpha_0, \ldots, \alpha_0) + N \cdot \mathrm{IDCT}((\alpha_i)_{i\in[\![0,d]\!]}, N). \tag{1.6}$$

When the polynomial P is given in the monomial basis, one has to first perform a change of basis to take advantage of the multipoint evaluation via the IDCT. The Fast Multipoint Evaluation operator FME is the composition of these operations.

Definition 1.2.3. For a polynomial $P(X) = \sum_{r=0}^{d} a_r X^r$, let us define $\mathrm{FME}((a_r)_{r\in[\![0,d]\!]}, N)$ the Fast Multipoint Evaluation of P at the Chebyshev nodes $(c_i)_{i\in[\![0,N-1]\!]}$, which is computed by Algorithm 3.

Subroutine 3 Fast Multipoint Evaluation.

Input: A polynomial $P(X) = \sum a_r X^r$ with $\mathbf{a} \in \mathbb{R}^{(d+1)}$ and a number of evaluations $N \in \mathbb{N}$
Output: The evaluations of P on the Chebyshev nodes for N

```
1: function REAL_FME(a, N)
2:     α ∈ R^(d+1)
3:     α ← REAL_MONOMIALTOCHEBYSHEV(a)        ▷ Change of basis
4:     b ∈ R^N
5:     b ← IDCT(α, N)
6:     return b
7: end function
```

Using the Fast Fourier Transform, the complexity of the IDCT is $O(N \log_2(N))$. Together with Lemma 0.2.2, this gives the complexity of the FME.

Lemma 1.2.8. *The number of arithmetic operations for the computation of the* FME *described in Subroutine 3 is* $O(d^2 + N\log_2(N))$.

Proof. We first call Real_MonomialToChebyshev on a vector of size $d+1$. Thus, from Lemma 0.2.2 the operation costs $O(d^2)$. After that, the IDCT is called for N points, so that costs $O(N\log_2(N))$. Therefore, the FME uses $O(d^2 + N\log(N))$ arithmetic operations. □

1.2.5 Multipoint partial evaluation and subdivision algorithm

Idea 3

The number of evaluations can be reduced by using multipoint evaluation techniques to compute the partial polynomials. We can simultaneously compute the k-th coefficient of each one.

Algorithm 4 does a fast partial evaluation of the polynomial using the FME and then uses the subdivision process we have seen. Section 3.1 verifies that it is indeed experimentally faster, even though it adds a change of basis step. This is the last algorithm of the chapter. It results from all the previous small improvements made to the naive algorithm. The combination of the fast multipoint evaluation based on the IDCT and the subdivision process is our contribution.

Subroutine 4 Computation of Chebyshev nodes.

Input: An integer N
Output: The Chebyshev nodes $\mathbf{c} \in \mathbb{R}^N$
1: **function** Real_ChebNodes(N)
2: $\quad\mathbf{c} \in \mathbb{R}^N$
3: $\quad$**for** $i \leftarrow 0$ **to** $N-1$ **do**
4: $\quad\quad c_i = \cos\left(\frac{2i+1}{2N}\pi\right)$
5: $\quad$**end for**
6: $\quad$**return c**
7: **end function**

Theorem 1.2.9. *The number of operations of Algorithm 4 is* $O(d^3 + dN\log_2(N) + dNT_{max})$, *where* T_{max} *is the size of the largest evaluation tree in the process.*

Proof. The partial evaluation of $P(X,Y) = \sum_{s=0}^{d}(\sum_{r=0}^{d} a_{r,s}X^r)Y^s$ in X is the evaluation of the $d+1$ polynomials $\sum_{i=0}^{d} a_{i,j}X^i$. In Line 4, each of these polynomials is evaluated via the FME at all the Chebyshev nodes. The cost is $d+1$ times the cost of one FME of size N, that is $O(d(d^2 + N\log_2(N)))$ according to Lemma 1.2.8. In Line 10, in each vertical fiber, the subdivision Algorithm 2 performs $O(T)$ interval evaluations of the partially evaluated univariate polynomial of degree d. Using a classical Horner evaluation, each evaluation is in $O(d)$. The complexity for all the fibers is thus $O(NdT_{max})$. □

Algorithm 4 Fast edge enclosing with subdivision.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{i,j} X^r Y^s$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$ and a resolution N

Output: The set of edges $\mathcal{E}$ of the form $(c_i, [y_j, y_{j+1}])$, which verify $0 \in P((c_i, [y_j, y_{j+1}]))$

1: **procedure** REAL_ENCLOSINGEDGES_MULTIEVAL-SUBD($coefMat$, N)
2: $\mathbf{b} \in \mathbb{IR}^{N\times d}$
3: **for** $k \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{s=0}^{d} b_{i,s} Y^s = P(c_i, Y)$
4: $b_{0,k}, \ldots, b_{N-1,k} \leftarrow$ REAL_FME($(a_{0,k}, \ldots, a_{d,k})$,$N$) ▷ N-point evaluation (Def. 1.2.3)
5: **end for**
6: $\mathcal{S} \leftarrow \emptyset$
7: $\mathbf{c} \leftarrow$ REAL_CHEBNODES(N)
8: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Subdivision in vertical fibers $X = c_i$
9: $Q : Y \longmapsto \sum_{s=0}^{d} b_{i,s} Y^s$
10: $\mathcal{I} \leftarrow$ REAL_ISOLATE1D(Q, $\mathbf{c}$, 0, N) ▷ Subdivision
11: **end for**
12: **return** $\mathcal{E}$
13: **end procedure**

2

Algorithms using floating point arithmetic

In this chapter, we build on the last algorithm. We are now doing the computations with precision p. Therefore, we have to bound the error propagated through the algorithms. This has an impact on the quality of our output. For instance, we may light some pixels that do not contain the curve we want to draw. Section 2.1 details the error of the FME in finite precision arithmetic. Then, Section 2.2 explains how the specifications of the algorithms presented in the previous chapter are changed in this context. Finally, Section 2.3 presents two algorithms for edge drawing and Section 2.4 presents one for pixel drawing.

2.1 Error analysis of the FME

We still want to take advantage of the FME in order to have a fast polynomial evaluation. In this section, we study the error induced by finite precision arithmetic. We assume without loss of generality that $N = 2^n$ is a power of 2. Let j be such that $j^2 = -1$. Let us recall that the non-normalized Inverse Discrete Fourier Transform (IDFT) is defined for a complex vector $z = (z_k)_{k\in[\![0,N-1]\!]}$ by

$$\mathrm{IDFT}(z) = \left(\frac{1}{N}\sum_{i=0}^{N-1} z_k e^{j\frac{2\pi}{N}ik}\right)_{k\in[\![0,N-1]\!]}. \tag{2.1}$$

Inverse Fast Fourier Transform (IFFT) error bound

Based on a recent error bound given for the Fast Fourier Transform (FFT) [BJM+20, Theorem 3.3 and 3.4], we can write a bound on the Inverse FFT in Corollary 2. Let us first recall the result for the FFT and then deduce the bound for the IFFT.

Theorem 2.1.1 ([BJM+20, Theorem 3.4])**.** *Assume radix-2, precision-p arithmetic, with rounding unit $u = 2^{-p}$. Let $\widehat{Z}$ be the computed 2^n-point FFT of $Z \in \mathbb{C}^{2^n}$ and let Z be the exact value. Then*

$$\left\|\widehat{Z} - Z\right\|_2 \leq \|Z\|_2 \left[(1+u)^n(1+g)^{n-2} - 1\right]$$

with

$$g = \frac{\sqrt{2}}{2}u + \rho_\times \left(1 + \frac{\sqrt{2}}{2}u\right)$$
$$\rho_\times = \begin{cases} u\sqrt{5} & \textit{using naive multiplication,} \\ 2u & \textit{using multiplication with fused multiply-add instruction.} \end{cases}$$

Corollary 2. *Assume radix-2, precision-p arithmetic, with rounding unit $u = 2^{-p}$. Let $\widehat{z}$ be the computed 2^n-point IFFT of $Z \in \mathbb{C}^{2^n}$ and let z be the exact value. Then*

$$\|\widehat{z} - z\|_2 \leq \|z\|_2 \left[(1+u)^n (1+g)^{n-2} - 1\right]$$

with

$$g = \frac{\sqrt{2}}{2}u + \rho_\times \left(1 + \frac{\sqrt{2}}{2}u\right)$$
$$\rho_\times = \begin{cases} u\sqrt{5} & \textit{using naive multiplication,} \\ 2u & \textit{using multiplication with fused multiply-add instruction.} \end{cases}$$

For our application, we actually want to bound $\|\widehat{z} - z\|_\infty^\perp$, where

$$\|z\|_\infty^\perp = \max_{i \in [\![0, N-1]\!]} \{\max(|\mathrm{Re}(z_i)|, |\mathrm{Im}(z_i)|)\}.$$

Let us also recall the following inequalities between norms:

$$\|z\|_\infty \leq \|z\|_2 \leq \sqrt{N}\,\|z\|_\infty \text{ and } \|z\|_\infty^\perp \leq \|z\|_\infty \leq \sqrt{2}\,\|z\|_\infty^\perp.$$

Additionally, with Parseval's theorem we have the well-known equality $\|z\|_2 = \frac{1}{\sqrt{N}}\|Z\|_2$ for the Discrete Fourier Transform from Equation 2.1. In order to deduce Corollary 3, note that

$$\|\widehat{z} - z\|_\infty^\perp \leq \|\widehat{z} - z\|_\infty \leq \|\widehat{z} - z\|_2,$$

$$\|z\|_2 = \frac{1}{\sqrt{N}}\,\|Z\|_2$$

and

$$\|Z\|_2 \leq \sqrt{N}\,\|Z\|_\infty \leq \sqrt{2}\,\sqrt{N}\,\|Z\|_\infty^\perp.$$

Corollary 3. *Assume radix-2, precision-p arithmetic, with rounding unit $u = 2^{-p}$. Let $\widehat{z}$ be then computed 2^n-point IFFT of $Z \in \mathbb{C}^{2^n}$ and let z be the exact value. Then*

$$\|\widehat{z} - z\|_\infty^\perp \leq \|Z\|_\infty^\perp \sqrt{2}\left[(1+u)^n (1+g)^{n-2} - 1\right].$$

Fast IDCT error bound

Theorem 2.1.2 shows that the absolute error $\|\widehat{x} - x\|_\infty$ on the output of the fast IDCT can be bounded with respect to u, g (defined in Corollary 2) and $\|X\|_\infty$.

Theorem 2.1.2. *Assume radix-2, precision-p arithmetic, with rounding unit $u = 2^{-p}$. Let $\widehat{x}$ be the computed 2^n-point fast* IDCT *of $X \in \mathbb{C}^{2^n}$ and let x be the exact value. Then*

$$\|\widehat{x} - x\|_\infty \leq e_{\mathrm{IDCT}}(n,p)\|X\|_\infty,$$

where

$$e_{\mathrm{IDCT}}(n,p) = \sqrt{2}\left[\sqrt{2}(1+u)^3(1+g)^2\left((1+u)^{n-1}(1+g)^{n-3}-1\right)\right) + (1+u)^3(1+g)^2 - 1\Big].$$

The following lemmas will be useful for the proof of Theorem 2.1.2. As a reminder, the canonical dot product for complex vectors is defined by $\langle x, y\rangle = \sum \overline{x_i} y_i$.

Lemma 2.1.3 and its generalization Lemma 2.1.4 will allow us to bound the error in the relation between V_k and T_k, seen in Section1.2.4. These quantities will arise in Cauchy-Schwartz inequalities during the proof of Theorem 2.1.2.

Lemma 2.1.3. *For $\omega \in \mathbb{C}$ such as $|\omega| = 1$,*

$$\left\| \begin{bmatrix} 1+j\omega \\ 1-j\omega \end{bmatrix} \right\|_2 = 2$$

Proof. Notice that

$$\begin{aligned} \overline{(1+j\omega)}(1+j\omega) &= (1-j\overline{\omega})(1+j\omega) \\ &= 1 + j\omega - j\overline{\omega} + 1 \end{aligned}$$

and

$$\begin{aligned} \overline{(1-j\omega)}(1-j\omega) &= (1+j\overline{\omega})(1-j\omega) \\ &= 1 - j\omega + j\overline{\omega} + 1. \end{aligned}$$

So,

$$\begin{aligned} \left\| \begin{bmatrix} 1+j\omega \\ 1-j\omega \end{bmatrix} \right\|_2^2 &= \overline{(1+j\omega)}(1+j\omega) + \overline{(1-j\omega)}(1-j\omega) \\ &= 4. \end{aligned}$$

□

Lemma 2.1.4. *For $n \in \mathbb{N}^*$ and $\omega \in \mathbb{C}^n$ such as $|\omega_i| = 1$,*

$$\left\| \begin{bmatrix} 1+j\omega_1 \\ 1-j\omega_1 \\ \vdots \\ 1+j\omega_n \\ 1-j\omega_n \end{bmatrix} \right\|_2 = 2\sqrt{n}$$

Proof. We give a proof by induction on $n \in \mathbb{N}^*$. The base case is verified by Lemma 2.1.3. For the induction, we assume that

$$\left\| \begin{bmatrix} 1+j\omega_1 \\ 1-j\omega_1 \\ \vdots \\ 1+j\omega_n \\ 1-j\omega_n \end{bmatrix} \right\|_2 = 2\sqrt{n}.$$

Thus,

$$\begin{aligned} \left\| \begin{bmatrix} 1+j\omega_1 \\ 1-j\omega_1 \\ \vdots \\ 1+j\omega_n \\ 1-j\omega_n \\ 1+j\omega_{n+1} \\ 1-j\omega_{n+1} \end{bmatrix} \right\|_2^2 &= \left\| \begin{bmatrix} 1+j\omega_1 \\ 1-j\omega_1 \\ \vdots \\ 1+j\omega_n \\ 1-j\omega_n \end{bmatrix} \right\|_2^2 + \left\| \begin{bmatrix} 1+j\omega_{n+1} \\ 1-j\omega_{n+1} \end{bmatrix} \right\|_2^2 \\ &= 4n+4 \\ &= 4(n+1). \end{aligned}$$

The result follows. □

The quantity $\prod_{k=1}^{n}(1+\delta_+^{(k)}) \times \prod_{l=1}^{m}(1+\delta_\omega^{(l)})$ appears in the relative error due to floating point arithmetic, where $\delta_+^{(k)}$, respectively $\delta_\omega^{(l)}$, is the relative error of the k^{th} addition, respectively the relative error of the l^{th} multiplication by a root of unity. Lemma 2.1.5 will allow us to bound this part.

Lemma 2.1.5. *For $n \in \mathbb{N}^*$, $m \in \mathbb{N}^*$, $\delta^+ \in \mathbb{R}^n$, $\delta^\omega \in \mathbb{R}^m$ such as $\left|\delta_k^+\right| \leq u$ and $|\delta_l^\omega| \leq g$,*

$$\left| \left(\prod_{k=1}^{n}(1+\delta_k^+) \times \prod_{l=1}^{m}(1+\delta_l^\omega) \right) - 1 \right| \leq (1+u)^n(1+g)^m - 1.$$

Proof. Let us define Δ such as

$$\begin{aligned} 1+\Delta &= \prod_{k=1}^{n}(1+\delta_k^+) \times \prod_{l=1}^{m}(1+\delta_l^\omega) \\ &= 1 + \delta_1^+ + \cdots + \delta_m^\omega + \cdots + \delta_1^+ \ldots \delta_m^\omega. \end{aligned}$$

Thus,

$$|\Delta| = \left|\delta_1^+ + \cdots + \delta_m^\omega + \cdots + \delta_1^+ \ldots \delta_m^\omega\right| \leq u + \cdots + g + \cdots + u \ldots g.$$

Notice that

$$(1+u)^n(1+g)^m = 1 + u + \cdots + g + \cdots + u \ldots g.$$

Subsequently,

$$\left| \left(\prod_{k=1}^{n}(1+\delta_k^+) \times \prod_{l=1}^{m}(1+\delta_l^\omega) \right) - 1 \right| \leq (1+u)^n(1+g)^m - 1.$$

□

Table 2.1: Summary of the operations for the IDCT and their relative errors.

	Operations	Floating point relative errors
$X_k \to V_k$	$V_k = \frac{1}{2}\omega_N^{-k}(X_k - jX_{N-k})$	g (since X_k real)
$V_k \to T_k$	$T_k = \frac{1}{2}((V_k + \overline{V_{\frac{N}{2}-k}}) + j\omega_N^{-k}(V_k - \overline{V_{\frac{N}{2}-k}}))$ $T_{\frac{N}{2}-k} = \frac{1}{2}((V_k + \overline{V_{\frac{N}{2}-k}}) - j\omega_N^{-k}(V_k - \overline{V_{\frac{N}{2}-k}}))$	$(1+g)(1+u)^3 - 1$
$T_k \to t_k$	$t_k = IFFT(T)_k$	$\sqrt{2}((1+u)^{n-1}(1+g)^{n-3} - 1)$ Cor. 3 for $\frac{N}{2} = 2^{n-1}$ points
$t_k \to v_k$	$v_{2k} = Re(t_k)$ $v_{2k+1} = Im(t_k)$	0
$v_k \to x_k$	$x_{2k} = v_k$ $x_{2k+1} = v_{N-k-1}$	0

We can now prove Theorem 2.1.2 using the error bounds in Table 2.1 for the different steps of the IDCT algorithms described in Figure 1.9.

Proof of Theorem 2.1.2. Let us go through the steps in Table 2.1 from bottom to top.

- $\underline{v_k \to x_k \text{ and } t_k \to v_k}$

  There is no computation in the operations to compute x_k from t_k ($v_k \to x_k$ and $t_k \to v_k$ in Table 2.1). So, we can easily rewrite the error on the output $(x_k)_{k\in[\![0,N-1]\!]}$.

  $$\|\widehat{x} - x\|_\infty = \|\widehat{v} - v\|_\infty = \left\|\widehat{t} - t\right\|_\infty^\perp$$

  Let $\widehat{T}$ (resp. $\widehat{t}$) be the computed value of the first two (resp. three) steps in Table 2.1, assuming precision-p arithmetic. Let us define $t^* = \text{IDFT}(\widehat{T})$ and $t = \text{IDFT}(T)$ the exact values. Taking into account the relative error at each step, we have :

  Then, by triangular inequality

  $$\boxed{\|\widehat{x} - x\|_\infty \leq \left\|\widehat{t} - t^*\right\|_\infty^\perp + \|t^* - t\|_\infty^\perp}$$

- $\underline{T_k \to t_k}$

  The first term is bounded by using Corollary 3 :

  $$\begin{aligned}\left\|\widehat{t} - t^*\right\|_\infty^\perp &\leq \left\|\widehat{T}\right\|_\infty^\perp \sqrt{2}\left[(1+u)^{n-1}(1+g)^{n-3} - 1\right] \\ &\leq \left\|\widehat{T}\right\|_\infty \sqrt{2}\left[(1+u)^{n-1}(1+g)^{n-3} - 1\right].\end{aligned}$$

By linearity, $t^* - t = \text{IDFT}(\widehat{T} - T)$. Moreover, $\max\{Re(e^{j\theta}), Im(e^{j\theta})\} \leq 1$. So for the second term, the triangular inequality on Equation 2.1 gives

$$\|t^* - t\|_\infty^\perp \leq \left\|\widehat{T} - T\right\|_\infty.$$

$\left\|\widehat{T}\right\|_\infty$ and $\left\|\widehat{T} - T\right\|_\infty$ are now each bounded using $\|T\|_\infty$.

In Table 2.1, we can see that $\widehat{T}$ is obtained with two multiplications and three additions:

$$\widehat{T_k} = T_k(1+\delta_1^\omega)(1+\delta_2^\omega)(1+\delta_1^+)(1+\delta_2^+)(1+\delta_3^+).$$

So,

$$\left\|\widehat{T}\right\|_\infty \leq \|T\|_\infty (1+u)^3(1+g)^2.$$

We note also that

$$\widehat{T_k} - T_k = T_k\left[(1+\delta_1^\omega)(1+\delta_2^\omega)(1+\delta_1^+)(1+\delta_2^+)(1+\delta_3^+) - 1\right]$$

and from Corollary 2.1.5

$$\left\|\widehat{T} - T\right\|_\infty \leq \|T\|_\infty \left[(1+u)^3(1+g)^2 - 1\right].$$

Thus,

$$\boxed{\left\|\widehat{t} - t^*\right\|_\infty^\perp + \|t^* - t\|_\infty^\perp \leq \|T\|_\infty \Big[\sqrt{2}\left((1+u)^3(1+g)^2\right)\left((1+u)^{n-1}(1+g)^{n-3} - 1\right) + \left((1+u)^3(1+g)^2 - 1\right)\Big].}$$

- $\underline{V_k \to T_k \text{ and } X_k \to V_k}$

Now, we rewrite T_k with respect to X_k, in order to bound $\|T\|_\infty$.

Since

$$V_k = \frac{1}{2}\omega_{4N}^{-k}\left[X_k - jX_{N-k}\right],$$

we get

$$\begin{aligned} V_{\frac{N}{2}-k} &= \frac{1}{2}\omega_{4N}^{-\frac{N}{2}+k}\left[X_{\frac{N}{2}-k} - jX_{N-\frac{N}{2}+k}\right] = \frac{1}{2}\omega_{4N}^{-\frac{N}{2}}\omega_{4N}^{k}\left[X_{\frac{N}{2}-k} - jX_{\frac{N}{2}+k}\right] \\ &= \frac{1}{2}\omega_8^{-1}\omega_{4N}^{k}\left[X_{\frac{N}{2}-k} - jX_{\frac{N}{2}+k}\right], \end{aligned}$$

so its conjugate is

$$\overline{V_{\frac{N}{2}-k}} = \frac{1}{2}\omega_8\omega_{4N}^{-k}\left[X_{\frac{N}{2}-k} + jX_{N-\frac{N}{2}+k}\right].$$

Consequently, the following sums can be obtained:

$$V_k + \overline{V_{\frac{N}{2}-k}} = \frac{1}{2}\omega_{4N}^{-k}\left[X_k + \omega_8 X_{\frac{N}{2}-k} + j\left(\omega_8 X_{\frac{N}{2}+k} - X_{N-k}\right)\right]$$

and

$$V_k - \overline{V_{\frac{N}{2}-k}} = \frac{1}{2}\omega_{4N}^{-k}\left[X_k - \omega_8 X_{\frac{N}{2}-k} - j\left(\omega_8 X_{\frac{N}{2}+k} + X_{N-k}\right)\right].$$

This gives

$$\begin{aligned} T_k = \frac{1}{4}\omega_{4N}^{-k}\Big[X_k + \omega_8 X_{\frac{N}{2}-k} + \omega_N^{-k}\left(\omega_8 X_{\frac{N}{2}+k} + X_{N-k}\right) \\ + j\left(\omega_8 X_{\frac{N}{2}+k} - X_{N-k} + \omega_N^{-k}\left(X_k - \omega_8 X_{\frac{N}{2}-k}\right)\right)\Big] \end{aligned}$$

and similarly

$$\begin{aligned} \overline{T_{\frac{N}{2}-k}} = \frac{1}{4}\omega_{4N}^{-k}\Big[X_k + \omega_8 X_{\frac{N}{2}-k} - \omega_N^{-k}\left(\omega_8 X_{\frac{N}{2}+k} + X_{N-k}\right) \\ + j\left(\omega_8 X_{\frac{N}{2}+k} - X_{N-k} - \omega_N^{-k}\left(X_k - \omega_8 X_{\frac{N}{2}-k}\right)\right)\Big]. \end{aligned}$$

In order to rewrite T_k and $\overline{T_{\frac{N}{2}-k}}$ as dot products of complex vectors, let us define

$$\widetilde{X_k} = \begin{bmatrix} X_k \\ \omega_8^{-1} X_{\frac{N}{2}-k} \\ \omega_8^{-1} X_{\frac{N}{2}+k} \\ X_{N-k} \end{bmatrix}, \widetilde{\omega_k^1} = \begin{bmatrix} 1 + j\omega_N^{-k} \\ 1 - j\omega_N^{-k} \\ \omega_N^{-k} + j \\ \omega_N^{-k} - j \end{bmatrix} \text{ and } \widetilde{\omega_k^2} = \begin{bmatrix} 1 + j\omega_N^{-k} \\ 1 - j\omega_N^{-k} \\ -\omega_N^{-k} + j \\ -\omega_N^{-k} - j \end{bmatrix}.$$

So, since X is a real vector Cauchy-Schwartz inequality leads to

$$\begin{aligned} T_k &= \frac{1}{4}\omega_{4N}^{-k}\langle\widetilde{X_k}, \widetilde{\omega_k^1}\rangle \leq \frac{1}{4}\omega_{4N}^{-k}\left\|\widetilde{X_k}\right\|_2\left\|\widetilde{\omega_k^1}\right\|_2 \text{and} \\ \overline{T_{\frac{N}{2}-k}} &= \frac{1}{4}\omega_{4N}^{-k}\langle\widetilde{X_k}, \widetilde{\omega_k^2}\rangle \leq \frac{1}{4}\omega_{4N}^{-k}\left\|\widetilde{X_k}\right\|_2\left\|\widetilde{\omega_k^2}\right\|_2. \end{aligned}$$

From Lemma 2.1.4, we have $\left\|\widetilde{\omega_k^1}\right\|_2 = \left\|\widetilde{\omega_k^2}\right\|_2 = 2\sqrt{2}$. Moreover, $\left\|\widetilde{X_k}\right\|_2 \leq \sqrt{4}\|X\|_\infty$.

Thus,

$$\|T\|_\infty \leq \frac{1}{4} \times 4\sqrt{2}\|X\|_\infty.$$

So,

$$\boxed{\|T\|_\infty \leq \sqrt{2}\,\|X\|_\infty}.$$

The result follows from these three steps. □

The ratio $\|\widehat{x} - x\|_\infty / \|X\|_\infty$ is computed for high resolutions in Table 2.2 and it does not exceed 10^{-13}. So, Theorem 2.1.2 is efficient in practice at controlling the error.

N	1024	2048	4096	8192	16384	32768
$\|\widehat{x}-x\|_\infty/\|X\|_\infty$	7.97e-15	8.84e-15	9.72e-15	1.06e-14	1.15e-14	1.23e-14

Table 2.2: IDCT error bounds for $p = 53$ (double precision) using Theorem 2.1.2.

2.1.1 FME error bound

For the fast multipoint evaluation, the polynomial is written in the Chebyshev basis and then the IDCT is applied on these coefficients. The change of basis introduces an error [Hig02, §3.1] which has to be added to the error from the IDCT to get a bound for the FME in Theorem 2.1.6.

Theorem 2.1.6. *Assume radix-2, precision-p arithmetic, with rounding unit $u = 2^{-p}$. Let $\widehat{z}$ be the computed 2^n-point fast evaluation on Chebyshev nodes of the polynomial whose coefficients are $a \in \mathbb{R}^{d+1}$ and let z be the exact value. In other words, $z = \mathrm{FME}((a_0,\dots,a_d),2^n)$ from Algorithm 3 when it works in the real RAM model and $\widehat{z} = \mathrm{FME}((a_0,\dots,a_d),2^n)$ when it works with floating point arithmetic and the* IDCT *is computed with the flowgraph of Figure 1.10. Then*

$$\|z-\widehat{z}\|_\infty \le e_{\mathrm{FME}}(d,n,p)\|a\|_\infty$$

where

$$e_{\mathrm{FME}}(d,n,p) = (d+1)\left[(d+1)\gamma + (1+\gamma)\left(2^n\beta(1+u) + \left(d+\frac{3}{2}\right)u\right)\right]$$

with

$$\gamma = \frac{(d+1)u}{1-(d+1)u},$$
$$\beta = \sqrt{2}\Big[\sqrt{2}(1+u)^3(1+g)^2\left((1+u)^{n-1}(1+g)^{n-3}-1\right) + (1+u)^3(1+g)^2 - 1\Big].$$

Assuming that $du \ll 1$, we can get an order of magnitude for the factor $e_{\mathrm{FME}}(d,n,p)$ in the error bound: $e_{\mathrm{FME}}(d,n,p) = O(d^3u + d2^n nu)$.

Proof. For a the vector consisting of the coefficients of the polynomial in the monomial basis, let us define X the coefficients in the Chebyshev basis and B the change of basis matrix, in other words $X = Ba$. By definition $x = \mathrm{IDCT}(X,N)$. Let us also define $x^* = \mathrm{IDCT}(\widehat{X},N)$ where $\widehat{X}$ is Ba computed with precision-p arithmetic and similarly $\widehat{x}$ is $\mathrm{IDCT}(\widehat{X},N)$ computed with precision-p arithmetic. Finally, we define z, z^* and $\widehat{z}$:

$$z = N\cdot x - \frac{1}{2}(X_0,\dots,X_0),$$
$$z^* = N\cdot x^* - \frac{1}{2}(X_0,\dots,X_0),$$
$$\widehat{z} = N\cdot \widehat{x} - \frac{1}{2}(X_0,\dots,X_0).$$

By norm inequality and triangular inequality,

$$\boxed{\|z-\widehat{z}\|_\infty \le \|z-z^*\|_\infty + \|z^*-\widehat{z}\|_\infty}$$

For the first term,

$$\begin{aligned} z - z^* &= N \cdot \mathrm{IDCT}(X, N) - \frac{1}{2}(X_0, \dots, X_0) - N \cdot \mathrm{IDCT}(\widehat{X}, N) + \frac{1}{2}(X_0, \dots, X_0) \\ &= N \cdot \mathrm{IDCT}(X - \widehat{X}) \text{ by linearity of the IDCT.} \end{aligned}$$

Then by bounding each term of Eq. (10),

$$\left\|\mathrm{IDCT}(X - \widehat{X}, N)\right\|_\infty \leq \frac{d+1}{N}\left\|X - \widehat{X}\right\|_\infty.$$

So,

$$\|z - z^*\|_\infty \leq (d+1)\left\|X - \widehat{X}\right\|_\infty. \tag{2.2}$$

For the second term, we need to bound the maximum value of $|z_k^* - \widehat{z}_k|$ for all k. For that we use the fused multiply-add instruction $\mathrm{FMA}(a, b, c)$ that computes $ab + c$ with a relative error δ with $|\delta| \leq u$.

$$\begin{aligned} z_k^* - \widehat{z}_k = &\ N \cdot x_k^* - \frac{1}{2}\widehat{X_0} - \mathrm{FMA}\left(N, \widehat{x}_k, -\frac{1}{2}\widehat{X_0}\right) \\ = &\ N \cdot x_k^* - \frac{1}{2}\widehat{X}_0 - \left(N \cdot \widehat{x}_k - \frac{1}{2}\widehat{X_0}\right)(1+\delta) \\ = &\ N\left(x_k^* - \widehat{x}_k\right) - N\delta \cdot \widehat{x}_k + \frac{1}{2}\widehat{X_0} \cdot \delta. \end{aligned}$$

Then

$$\|z^* - \widehat{z}\|_\infty \leq N\|x^* - \widehat{x}\|_\infty + Nu \cdot \|\widehat{x}\|_\infty + \frac{1}{2}\left|\widehat{X_0}\right|u.$$

By abuse of notation, we continue to write $\widehat{X}$ the vector padded with zeros in order to apply Theorem 2.1.2 for 2^n points, and we deduce $\|x^* - \widehat{x}\|_\infty \leq \beta\left\|\widehat{X}\right\|_\infty$. Moreover, using the definition of IDCT in Equation (10), we have $\|x^*\|_\infty \leq \frac{d+1}{N}\left\|\widehat{X}\right\|_\infty$, such that $\|\widehat{x}\|_\infty \leq \|x^*\|_\infty + \|\widehat{x} - x^*\|_\infty \leq \left(\frac{d+1}{N} + \beta\right)\left\|\widehat{X}\right\|_\infty$. Thus,

$$\|z^* - \widehat{z}\|_\infty \leq N\left\|\widehat{X}\right\|_\infty\left(\beta(1+u) + \frac{d+1}{N}u\right) + \frac{1}{2}\left|\widehat{X_0}\right|u.$$

So,

$$\boxed{\|z - z^*\|_2 + \|z^* - \widehat{z}\|_2 \leq (d+1)\left\|X - \widehat{X}\right\|_\infty + N\left\|\widehat{X}\right\|_\infty\left(\beta(1+u) + \frac{d+1}{N}u\right) + \frac{1}{2}\left|\widehat{X_0}\right|u}$$

For each component of X, the change of basis is a matrix multiplication where each component is an inner product of size $d+1$. Letting $\widetilde{a}$ be the vector of the $|a_k|$ for k from 0 to d, we introduce $\widetilde{X} = B\widetilde{a}$, and a classical error bound [Hig02, §3.1] on the inner product yields:

$$\left\|X - \widehat{X}\right\|_\infty \leq \left\|\widetilde{X}\right\|_\infty \gamma.$$

Consequently,

$$\left\|\widehat{X}\right\|_\infty \leq \left\|\widetilde{X}\right\|_\infty (1+\gamma).$$

Let us now bound $\left\|\widetilde{X}\right\|_\infty$ from the change of basis where a is a vector of size $d+1$, thus

$$\left\|\widetilde{X}\right\|_\infty \leq (d+1)\max_{i,j}\{B_{i,j}\}\|a\|_\infty$$

where $\max_{i,j}\{B_{i,j}\} = B_{0,0} = 1$, using the relations given by Mason and Handscomb [MH03, section 2.3.1]. Hence,

$$\boxed{\left\|X-\widehat{X}\right\|_\infty \leq (d+1)\|a\|_\infty\gamma \text{ and } \left\|\widehat{X}\right\|_\infty \leq (d+1)\|a\|_\infty(1+\gamma)}$$

Subsequently,

$$\begin{aligned}\|z-\widehat{z}\|_\infty &\leq (d+1)\|a\|_\infty\left[(d+1)\gamma+(1+\gamma)\left(N\beta(1+u)+(d+1)u+\frac{u}{2}\right)\right]\\ &\leq (d+1)\|a\|_\infty\left[(d+1)\gamma+(1+\gamma)\left(N\beta(1+u)+\left(d+\frac{3}{2}\right)u\right)\right].\end{aligned}$$

□

2.1.2 FME with interval coefficients

In the case where the input polynomial has interval coefficients, we need to compute the interval enclosure of its evaluation on the Chebyshev nodes. This leads to a function denoted by □FME that takes as input a polynomial with interval coefficients, and returns a list of intervals that each contains the evaluation of the input polynomial on a Chebyshev node. The function □FME is computed in two steps. First, we compute the change of basis with Subroutine 1. Then, we compute the IDCT on a vector of intervals. A challenge is to keep a tight inclusion and the quasi-linear complexity in N stated by Corollary 4.

Definition 2.1.1. Let $A \in \mathbb{IF}^{d+1}$ be the interval coefficients of a polynomial of degree d and let $X = (X_0, \ldots, X_d)$ be those coefficients in the Chebyshev basis, obtained with Subroutine 1. Let $x \in \mathbb{F}^{d+1}$ be the vector of the centers of the intervals of X and r be the maximum of the radii of these intervals. Let $\widehat{x}$ be the result of the fast IDCT computation applied on x, and $e_{\mathrm{IDCT}}(X)$ is the bound on the error given in Theorem 2.1.2. We also let $E \in \mathbb{IF}^N$ be a vector where all the entries are $[-e_{\mathrm{IDCT}}(X) - \frac{d+1}{N}r, e_{\mathrm{IDCT}}(X) + \frac{d+1}{N}r]$. Finally, we define

$$\square\,\mathrm{FME}(A) = N\cdot(\widehat{x}+E) + \frac{1}{2}(X_0,\ldots,X_0).$$

As a corollary of Theorem 2.1.2 on the error bound on the IDCT, we deduce a bound on the error of the FME function.

Corollary 4. *The function* □FME *is an extension of the function* FME *and uses* $O(d^2 + N\log_2(N))$ *arithmetic operations.*

Proof. First, let $Y \in \mathbb{IF}^N$ be the vector of the interval ranges $\mathrm{IDCT}(X)$. Using interval arithmetic, we have $\mathrm{FME}(A) \subset N \cdot Y + \frac{1}{2}(X_0, \dots, X_0)$ by definition of the FME operator in Section 1.2.4. Then, since the IDCT is linear, we have $\mathrm{IDCT}(X) = \mathrm{IDCT}(x) + \mathrm{IDCT}(X - x)$. From Theorem 2.1.2, we have $\mathrm{IDCT}(x) \subset \widehat{x} + e$. And since all the entries of $X - x$ are bounded by r, using the explicit formula for the IDCT given in Definition 0.4.1, we bound the absolute value of each entry of $\mathrm{IDCT}(X - x)$ by $\frac{d+1}{N} r$. With the notation of Definition 2.1.1, this implies $\mathrm{IDCT}(x) + \mathrm{IDCT}(X - x) \subset \widehat{x} + E$, which concludes the proof for the bound. For the complexity, the dominating parts are the computation of the fast IDCT in $O(N \log_2(N))$ arithmetic operations and the change of basis in $O(d^2)$ operations. □

Subroutine 5 describes the computations of the operator □FME by following Definition 2.1.1.

Subroutine 5 Fast Multipoint Evaluation with interval arithmetic.

Input: A polynomial $P(X) = \sum a_i X^i$ with $\mathbf{a} \in \mathbb{F}^{(d+1)}$ and a number of evaluations $N \in \mathbb{N}$
Output: The evaluations of P on the Chebyshev nodes for N

function FLOAT_FME($\mathbf{a}$, N)
  $\mathbf{X} \in \mathbb{IF}^{(d+1)}$
  $\mathbf{X} = \mathbf{x} + [-\mathbf{r}, \mathbf{r}] \leftarrow$ FLOAT_MONOMIALTOCHEBYSHEV($\mathbf{a}$) ▷ Change of basis
  $r_{max} \leftarrow \max \mathbf{r}$
  $\widehat{\mathbf{x}}, \mathbf{E}, \mathbf{b} \in \mathbb{IF}^N$
  $\widehat{\mathbf{x}} \leftarrow \mathrm{IDCT}(\mathbf{x}, N)$
  $x_{max} \leftarrow \max \widehat{\mathbf{x}}$
  StartUpwardRounding
    $u \leftarrow 2^{-53}$ ▷ Double precision
    $\rho \leftarrow \sqrt{5}u$ ▷ If naive multiplication (no FMA)
    $g \leftarrow \frac{u}{\sqrt{2}} + \rho(1 + \frac{u}{\sqrt{2}})$
    $e \leftarrow \sqrt{2} x_{max} \left[\sqrt{2}(1+u)^3(1+g)^2 \left((1+u)^{n-1}(1+g)^{n-3} - 1\right]\right) + (1+u)^3(1+g)^2 - 1\big]$
    $\mathbf{E} \leftarrow ([-e - \frac{d+1}{N} r_{max}, e + \frac{d+1}{N} r_{max}], \dots, [-e - \frac{d+1}{N} r_{max}, e + \frac{d+1}{N} r_{max}])$
    $\mathbf{b} \leftarrow N(\widehat{\mathbf{x}} + \mathbf{E}) + \frac{1}{2}(X_0, \dots, X_0)$
  StopUpwardRounding
  return b
end function

2.2 Consequences of precision-p arithmetic and interval arithmetic on previous results

2.2.1 Representing a curve on a Chebyshev grid

We have seen in Section 1.1 what a Chebyshev grid is (see Figure 1.4). Since we use the FME in our computations to evaluate our polynomials on Chebyshev nodes, such a grid becomes the natural one to use. Since, we are working with precision-p arithmetic we have to consider the approximations $(\widehat{c_i})_{i \in [\![0, N-1]\!]}$ of the Chebyshev nodes $(c_i)_{i \in [\![0, N-1]\!]}$. Since no confusion is possible in the chapter, by abuse of notation we will also use the notation c_i for the approximations. Furthermore, we assume that at the start of each procedure the N Chebyshev nodes have been computed. That computation is always negligible compared to the rest of the algorithm. As a reminder, $-1 < c_{N-1} < \cdots < c_0 < 1$.

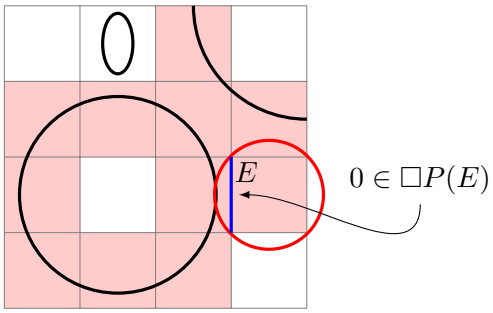


Figure 2.1: A false positive pixel circled in red, because for the blue edge $0 \in \square P(E)$.

2.2.2 Subdivision and evaluation trees with interval arithmetic

With interval arithmetic, if $0 \notin \square P([\alpha, \beta])$, then $0 \notin P([\alpha, \beta])$. However, if $0 \in \square P([\alpha, \beta])$, nothing can be concluded and the interval needs to be subdivided. So, we only have an exclusion predicate. Therefore, in the subdivision algorithm we can be left with an edge which does not intersect the curve. By lighting the pixels adjacent to such an edge, we also get false positives, as illustrated by Figure 2.1. From Section 1.2.3, only Lemma 1.2.4 stays valid. We can no longer state that $T = O(d \log_2(N))$, but that is nonetheless what we expect experimentally. We even expect $T = O(\log_2(N))$, because the factor d comes from the fact that the polynomial has d roots, but only a limited number of them are generally in $[-1, 1]$.

2.3 Two edge enclosing algorithms

Without loss of generality, we use the hypothesis that $N = 2^n$ from Section 2.1 for the analysis of the error on the output. For the two algorithms of this section, we define two types of edges.

Definition 2.3.1. An edge $(x, [y_1, y_2])$ between two consecutive nodes of the grid is called:

- *crossed edge* if the curve intersects $(x, [y_1, y_2])$,
- *candidate edge* if $0 \in \square P(x, [y_1, y_2]) + [-E, E]$.

The expression of E is defined in Lemmas 2.3.3 and 2.3.8, respectively for each algorithm. For Algorithm 5, $E = O\left(de_{\mathrm{FME}}(d, n, p)\|(a_{i,j})\|_\infty\right)$. For Algorithm 6, if the factor $e_{\mathrm{FME}}(d, n, p)$ from Theorem 2.1.6 is small enough, $E = O\left(de_{\mathrm{FME}}(d, n, p)\|(a_{i,j})\|_\infty + d\|(a_{i,j})\|_\infty \frac{1}{(m+1)^{m+1}}\right)$.

Our algorithms return edges from which one can construct a crossing-edge approximation of the curve, as stated by Definition 2.3.2.

Definition 2.3.2. A set of pixels is a *crossing-edge approximation* if the set of edges of its pixels contains all the crossed segments and is contained in the set of candidate segments.

2.3.1 Rewriting the subdivision algorithm

Algorithm 3 becomes Algorithm 6: we replace the polynomial P with its extension $\square P$ (Line 2). Therefore, we no longer rely on the oracle $\mathcal{O}$. The test is implemented using interval arithmetic to evaluate $\square P$. Similarly, from Algorithm 4 to Algorithm 5 the operator FME become $\square$FME (Line 6). The structure of the algorithm does not change much. The operations which

Subroutine 6 Isolation function with subdivision using interval arithmetic.

Input: A univariate polynomial function P, a resolution integer $N > 0$ and two integer indices i and j

Output: The set of intervals $\mathcal{I}$ of the form $[c_{k+1}, c_k]$, which verify $[c_{k+1}, c_k] \subset [c_j, c_i]$ and $0 \in \square P([c_{k+1}, c_k])$

Require: $i < j$

```
1:  function FLOAT_ISOLATE1D(P, N, i, j)
2:      if □P([c_j, c_i]) contains 0 then          ▷ Test with interval arithmetic
3:          if i + 1 < j then                      ▷ Split in two subintervals
4:              k = ⌊(i+j)/2⌋
5:              I_1 ← FLOAT_ISOLATE1D(P, N, i, k)
6:              I_2 ← FLOAT_ISOLATE1D(P, N, k + 1, j)
7:              return I_1 ∪ I_2
8:          else
9:              return {[c_{i+1}, c_i]}
10:         end if
11:     else
12:         return ∅
13:     end if
14: end function
```

have changed have the same complexity as the original ones. Note however that with interval arithmetic, the test of the subdivision algorithm is only for an exclusion predicate.

Lemma 2.3.1. *The number of arithmetic operations of Subroutine 6 called on* FLOAT_ISOLATE1D*$(P, N, 0, N)$ is $O(dT)$, when P is of degree d.*

Proof. Subroutines 2 and 6 have the same structure. The only change is the test in Line 2. In the proof of Subroutine 2 we used the oracle $\mathcal{O}$ with a complexity of $O(d)$. Here, we use interval arithmetic to evaluate a polynomial of degree d. Consequently, the evaluation of $\square P$ has a complexity of $O(d)$ and the test a complexity of $O(1)$. Hence, the complexity is unchanged. □

Theorem 2.3.2. *The number of arithmetic operations of Algorithm 5 (*FLOAT_ENCLOSING-EDGES_MULTIEVAL-SUBD*) is $O(d^3 + dN\log_2(N) + dNT_{max})$, where T_{max} is the size of the largest evaluation tree in the process.*

Proof. Algorithms 4 and 5 have the same structure. The only change is the test in Line 4. In the proof for the complexity of Subroutine 3, we used Lemma 1.2.8. Here, we use Corollary 4, which also gives a complexity of $O(d^2 + N\log_2(N))$ for the $\square$FME. Subsequently, the complexity is unchanged. □

Lemma 2.3.3. *Algorithm 5 does not return an edge $(c_i, [y_1, y_2])$ if $0 \notin \square P(c_i, [y_1, y_2]) + [-E, E]$ where P is of degree d and $E = 2(d+1)e_{\mathrm{FME}}(d, n, p)\|(a_{i,j})\|_\infty$, with e_{FME} defined in Theorem 2.1.6.*

Proof. Algorithm 5 takes $P(X, Y) = \sum_{r,s=0}^{d} a_{r,s}X^rY^s$ as an input. Then, in the for loop of Line 3 it is evaluated to get the $P(c_i, Y) = \sum b_{i,s}Y^s$ for $i \in [\![0, N-1]\!]$. However, FLOAT_FME

Algorithm 5 Fast edge enclosing with subdivision using interval arithmetic.

Input: A bivariate polynomial $P(X,Y) = \sum_{i,j} a_{i,j} X^i Y^j$ with $\mathbf{a} \in \mathbb{F}^{(d+1)\times(d+1)}$ and a Chebyshev grid resolution integer $N > d > 0$.

Output: A set of vertical edges $\mathcal{E}$ containing all the vertical crossed edges and contained in the set of candidate edges (Definition 2.3.1)

1: **procedure** FLOAT_ENCLOSINGEDGES_MULTIEVAL-SUBD($\mathbf{a}$, N)
2: $\mathbf{b} \in \mathbb{IF}^{N\times d}$
3: **for** $k \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{j=0}^{d} b_{i,j} Y^j = P(c_i, Y)$
4: $b_{0,k}, \ldots, b_{N-1,k} \leftarrow$ FLOAT_FME($(a_{0,k}, \ldots, a_{d,k})$,$N$) ▷ N-point evaluation (Def. 2.1.1)
5: **end for**
6: $\mathcal{E} \leftarrow \emptyset$
7: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Subdivision in vertical fibers $X = c_i$
8: $Q : Y \longmapsto \sum_k b_{i,k} Y^k$
9: $\mathcal{I} \leftarrow$ FLOAT_ISOLATE1D(Q, N, 0, $N-1$)
10: **for** $I \in \mathcal{I}$, Add (c_i, I) to $\mathcal{E}$
11: **end for**
12: **return** $\mathcal{E}$
13: **end procedure**

computes the floating points $\left(\widehat{b_{i,s}}\right)_{i\in[\![0,N-1]\!],s\in[\![0,d]\!]}$ and returns the intervals $\left(\widetilde{b_{i,s}}\right)_{i\in[\![0,N-1]\!],s\in[\![0,d]\!]}$. According to Theorem 2.1.6, for a given s,

$$\left\|(\widehat{b_{i,s}})_i - (b_{i,s})_i\right\|_\infty \leq e_{\mathrm{FME}}(d,n,p)\|(a_{i,j})_i\|_\infty.$$

We can even write $\left\|(\widehat{b_{i,s}})_i - (b_{i,s})_i\right\|_\infty \leq e(P,d,n,p)$ and define the upper bound for the polynomial $e(P,d,n,p) = e_{\mathrm{FME}}(d,n,p)\|(a_{r,s})_{r,s}\|_\infty$. For all i and s, there exists $\delta_{i,s}$ such that $\widehat{b_{i,s}} = b_{i,s} + \delta_{i,s}$ and $|\delta_{i,s}| \leq e(P,d,n,p)$.

$$\widehat{P}(c_i,y) = \sum_{s=0}^{d} \widehat{b}_{i,s} y^s = \sum_{s=0}^{d} (b_{i,s} + \delta_{i,s}) y^j s = \sum_{s=0}^{d} b_{i,s} y^j s + \sum_{s=0}^{d} \delta_{i,s} y^s = P(c_i,y) + \sum_{s=0}^{d} \delta_{i,s} y^s.$$

In Line 3, the partial polynomial $\widetilde{P}(c_i,Y)$ with interval coefficients is constructed such that $\widehat{P}(c_i,Y) \in \widetilde{P}(c_i,Y)$, where $\in$ denotes a coefficient-wise set membership:

$$\begin{aligned}\tilde{b}_{i,s} &= \widehat{b}_{i,s} + [-e(P,d,n,p), e(P,d,n,p)] \\ &= b_{i,s} + \delta_{i,s} + [-e(P,d,n,p), e(P,d,n,p)].\end{aligned}$$

Subsequently,

$$\begin{aligned}\widetilde{P}(c_i,y) &= \sum_{s=0}^{d} \widetilde{b}_{i,s} y^s \\ &= P(c_i,y) + \sum_{s=0}^{d} \delta_{i,s} y^s + \sum_{s=0}^{d} [-e(P,d,n,p), e(P,d,n,p)] y^s.\end{aligned}$$

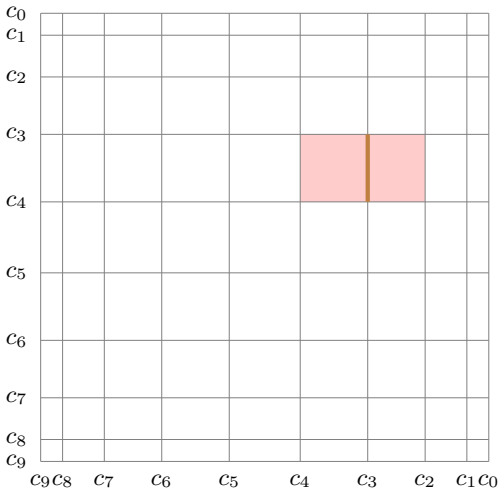


Figure 2.2: If an intersection is detected on $c_3 \times [c_4, c_3]$ (brown line), then the two pixels overlapping $[c_4, c_2] \times [c_4, c_3]$ are lighted (light red).

In the following expression, $\subseteq$ denotes a coefficient-wise inclusion. Since $\delta_{i,s} \in [-e(P,d,n,p), e(P,d,n,p)]$,

$$\widetilde{P}(c_i, y) \subseteq P(c_i, y) + 2\sum_{s=0}^{d} [-e(P,d,n,p), e(P,d,n,p)]y^s.$$

Given that $[y_1, y_2] \subset [-1, 1]$, we deduce from Corollary 1 that

$$\square\widetilde{P}(c_i, [y_1, y_2]) \subseteq \square P(c_i, [y_1, y_2]) + 2(d+1)[-e(P,d,n,p), e(P,d,n,p)].$$

Therefore, if the interval $\square P(c_i, [y_1, y_2]) + [-E, E]$ does not contain a zero, then $\square\tilde{P}(c_i, [y_1, y_2])$ does not either. Thus, the vertical edge $(c_i, [y_1, y_2])$ is not in the output. □

One constructs a pixel drawing by lighting the pixels adjacent to the edges returned by Algorithm 6. When the evaluation on $c_i \times [c_{j+1}, c_j]$ contains a zero, the two pixels $[c_{i+1}, c_i] \times [c_{j+1}, c_j]$ and $[c_i, c_{i-1}] \times [c_{j+1}, c_j]$ are selected. For example, Figure 2.5 shows that if a zero is detected on $c_3 \times [c_4, c_3]$, then the two pixels $[c_4, c_3] \times [c_4, c_3]$ and $[c_3, c_2] \times [c_4, c_3]$ are lighted.

2.3.2 Taylor approximation

An alternative to the subdivision process with interval arithmetic can be a second fast multipoint evaluation in the second direction.

Idea 4

Using Taylor approximation, it is possible to use multipoint evaluation and bound the values in an interval around each vertex.

Bound with Taylor approximation

The Taylor-Lagrange inequality states that for a function f and two reals $a, b \in I$,

$$\left| f(b) - \sum_{k=0}^{m} \frac{1}{k!}(b-a)^k f^{(k)}(a) \right| \leq \max_I \left| f^{(m+1)} \right| \frac{|b-a|^{m+1}}{(m+1)!}.$$

In Algorithm 6, the derivatives are evaluated using the $\square$FME operator at the Chebyshev nodes. It then remains to use the Taylor-Lagrange inequality to bound the values in a neighborhood of each Chebyshev node. Theorem 2.3.4 provides two bounds, the second one is better for points close to -1 or 1.

Theorem 2.3.4. *For a polynomial $P(X) = \sum_{i=0}^{d} a_i X^i$, let $b = \max_{i \in [\![0,d]\!]} |a_i|$, $c \in [-1,1]$ and $r \in [-R, R]$, if $|c + r| \leq 1$*

$$\left| P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c) \right| \leq b\, R^{m+1} \binom{d+1}{m+2}$$

and if $|c \pm R| < 1$

$$\left| P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c) \right| \leq b\, \frac{R^{m+1}}{(1-|c|-R)^{m+2}}.$$

Proof. Let us write the Taylor-Lagrange inequality centered in c

$$\left| P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c) \right| \leq \max_{[c-R,c+R]} \left| P^{(m+1)} \right| \frac{R^{m+1}}{(m+1)!}.$$

Using the formula of an ordinary generating function

$$\sum_{i=0}^{\infty} \binom{i}{l} x^i = \frac{x^l}{(1-x)^{l+1}} \text{ for } x \in]-1, 1[.$$

Thus,

$$\begin{aligned}
\max_{[c-R,c+R]} \left| P^{(m+1)} \right| &\leq \max |a_n| \sum_{i=m+1}^{d} \frac{i!}{(i-m-1)!} (|c|+R)^{i-m-1} \\
&\leq \max |a_n| \frac{(m+1)!}{(|c|+R)^{m+1}} \sum_{i=m+1}^{d} \binom{i}{m+1} (|c|+R)^i \\
&\leq \max |a_n| \frac{(m+1)!}{(1-|c|-R)^{m+2}}.
\end{aligned}$$

Subsequently,

$$\left| P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c) \right| \leq \max |a_n| \frac{R^{m+1}}{(1-|c|-R)^{m+2}} \text{ if } |c \pm R| < 1.$$

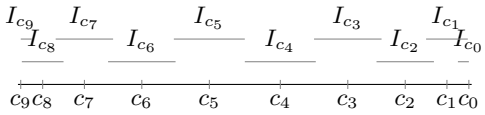


Figure 2.3: Intervals centered at the Chebyshev nodes, represented above the real axis.

Using the hockey-stick identity

$$\sum_{i=r}^{n} \binom{i}{l} = \binom{n+1}{l+1}.$$

The formula of the derivative of a polynomial is

$$P^{(k)}(x) = \sum_{i=k}^{d} a_i \frac{i!}{(i-k)!} x^{i-k}.$$

So,

$$\begin{aligned}
\max_{[c-R,c+R]\cap[-1,1]} \left|P^{(m+1)}\right| &\leq \max|a_n| \sum_{i=m+1}^{d} \frac{i!}{(i-m-1)!} \\
&\leq \max|a_n|(m+1)! \sum_{i=m+1}^{d} \binom{i}{m+1} \\
&\leq \max|a_n|(m+1)! \binom{d+1}{m+2}.
\end{aligned}$$

Consequently, if $|c+r| \leq 1$

$$\left|P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c)\right| \leq \max|a_n| R^{m+1} \binom{d+1}{m+2}.$$

□

An algorithm combining □FME and Taylor approximation

Algorithm 6 does fast partial evaluation of the polynomial using □FME and then uses the fast evaluation techniques once again on the newly formed partial polynomials combined with Taylor approximation. For $i \in [\![0, N-1]\!]$, the error on the evaluation of the partial polynomials is bounded on each interval I_{c_i}, centered in the Chebyshev node c_i and of radius $R = \max\{\frac{c_i - c_{i+1}}{2}, \frac{c_{i-1} - c_i}{2}\}$ when $i \in [\![1, N-2]\!]$. $I_{c_0} = [\frac{c_0 - c_1}{2}, c_0]$ and $I_{c_{N-1}} = [c_{N-1}, \frac{c_{N-2} - c_{N-1}}{2}]$. In Figure 2.3, the intervals are represented above the real axis at different heights not to clutter the figure. One can see that by construction they overlap.

For this, we need two functions given by Algorithms 7 and 8. The first one computes the degree m Taylor approximations of a polynomial at the Chebyshev nodes. The second one evaluates a Taylor approximation by interval arithmetic on the corresponding vertical neighborhood and bounds the result using Theorem 2.3.4.

Subroutine 7 Taylor approximation.

Input: A univariate polynomial $P(X) = \sum_{i=0}^{m} a_i X^i$, a Chebyshev grid resolution integer $N > 0$ and the order of the approximation m

Output: The Taylor approximations of degree m at the $(c_i)_{[\![0,N-1]\!]}$: $Q_i(X) = \sum_{k=0}^{m} \frac{1}{k!} X^k P^{(k)}(c_i)$

1: **function** FLOAT_TAYLORAPPROXIMATIONS(P, N, m)
2: $\mathbf{p} \in \mathbb{IF}^{(m+1)\times N}$
3: **for** $k \leftarrow 0$ **to** m **do** ▷ Computation of the derivatives
4: $\sum_{i=0}^{d-k} b_i X^i \leftarrow \mathrm{diff}(\sum_{i=0}^{d} a_i X^i, k)$ ▷ $\sum_{j=0}^{d-k} b_j X^j = P^{(k)}(X)$
5: $p_{k,0}, \ldots, p_{k,N-1} \leftarrow$ FLOAT_FME($(b_0, \ldots, b_{d-k})$,N) ▷ $p_{k,j} = P^{(k)}(c_i)$
6: **end for**
7: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Construction of the Taylor approximations
8: $Q_i \leftarrow \sum_{l=0}^{m} \frac{p_{l,i}}{l!} X^l$
9: **end for**
10: **return** $Q_0, \ldots, Q_{N-1}$
11: **end function**

Lemma 2.3.5. *The number of arithmetic operations of Subroutine 7 (*FLOAT_TAYLORAPPROXIMATIONS*) is* $O(md^2 + mN\log_2(N))$.

Proof. For each degree $k \in [\![0, m]\!]$ of the approximation, the polynomial is derived, then the $\square$FME is applied on the obtained polynomial. The derivation of order k of a polynomial of degree d has a complexity $O(d)$. The resulting polynomial has at most a degree d. Thus, the $\square$FME has a complexity of $O(d^2 + N\log_2(N))$ from Corollary 4. Since we do this $m+1$ times, this whole step has a complexity of $O(md^2 + mN\log_2(N))$. Then, for each Chebyshev node the coefficients of a degree m polynomial are computed. This results in a complexity of $O(mN)$. $\square$

Subroutine 8 Evaluation around Chebyshev node.

Input: A univariate polynomial $T(X) = \sum_{i=0}^{m} t_i X^i$, its degree m, the index j of the Chebyshev nodes around which it should be evaluated, a positive real b and an integer N

Output: The local evaluation around (c_j) with a radius $R = \max\{\frac{c_j - c_{j+1}}{2}, \frac{c_{j-1} - c_j}{2}\}$

1: **function** FLOAT_EVALAROUNDCHEBNODE(T, m, j, b, N)
2: **if** $j < N/2$ **then** $R \leftarrow \frac{c_j - c_{j+1}}{2}$ **else** $R \leftarrow \frac{c_{j-1} - c_j}{2}$ **end if**
3: $\beta \leftarrow bR^{m+1} \min\left\{\frac{1}{(1-|c_j|-R)^{m+2}}, \binom{d+1}{m+2}\right\}$ ▷ see Thm. 2.3.4
4: **switch** j **do**
5: **case** 0
6: $I \leftarrow \square T([-R, 0]) + [-\beta, \beta]$
7: **case** $N-1$
8: $I \leftarrow \square T([0, R]) + [-\beta, \beta]$
9: **default**
10: $I \leftarrow \square T([-R, R]) + [-\beta, \beta]$
11: **end switch**
12: **return** I
13: **end function**

Lemma 2.3.6. *The number of arithmetic operations of Subroutine 8 (*Float_EvalAround-ChebNode*) is* $O(m)$.

Proof. The test and the computation of R costs $O(1)$. Therefore, computing R^{m+1}, $\frac{1}{(1-|c_j|-R)^{m+2}}$ or $\binom{d+1}{m+2}$ also costs $O(m)$. Consequently, the same goes for β. Finally, the evaluation of $\square T$ where T is of degree m costs also $O(m)$. Therefore, the global complexity is $O(m)$. □

Algorithm 6 Fast edge enclosing with Taylor approximation.

Input: A bivariate polynomial $P(X,Y) = \sum_{i,j} a_{i,j} X^i Y^j$ with $\mathbf{a} \in \mathbb{F}^{(d+1)\times(d+1)}$, a Chebyshev grid resolution integer $N > 0$ and the order m of the Taylor approximation.

Output: A set of vertical edges $\mathcal{E}$ containing all the vertical crossed edges and contained in the set of candidate edges (Definition 2.3.1)

Require: $4d < N$

1: **procedure** Float_EnclosingEdges_MultiEval-TaylorApprox($\mathbf{a}$, N, m)
2: $\mathbf{b} \in \mathbb{IF}^{N\times(d+1)}$
3: **for** $j \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{j=0}^{d} b_{i,j} Y^j = P(c_i, Y)$
4: $b_{0,j}, \ldots, b_{N-1,j} \leftarrow$ Float_FME($(a_{0,j}, \ldots, a_{d,j})$,$N$) ▷ N-point evaluation (Def. 2.1.1)
5: **end for**
6: $\mathcal{E} \leftarrow \emptyset$
7: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Processing vertical fiber $X = c_i$
8: $Q : Y \longmapsto \sum_{t=0}^{d} b_{i,t} Y^t$
9: $T_{i,0}, \ldots, T_{i,N-1} \leftarrow$ Float_TaylorApproximations(Q,N,m)
10: $b_{i,max} \leftarrow \max_{0\le j\le d} |b_{i,j}|$
11: **for** $j \leftarrow 0$ **to** $N-1$ **do** ▷ Evaluation of $P(c_i, I_{c_j})$
12: $I_{i,j} \leftarrow$ Float_EvalAroundChebNode($T_{i,j}$, m, j, $b_{i,max}$, N)
13: **switch** j **do**
14: **case** 0
15: **if** $0 \in I_{i,j}$ **then** Add $(c_i, [c_{j+1}, c_j])$ to $\mathcal{E}$
16: **case** $N-1$
17: **if** $0 \in I_{i,j}$ **then** Add $(c_i, [c_j, c_{j-1}])$ to $\mathcal{E}$
18: **default**
19: **if** $0 \in I_{i,j}$ **then** Add $(c_i, [c_{j+1}, c_j])$ and $(c_i, [c_j, c_{j-1}])$ to $\mathcal{E}$
20: **end switch**
21: **end for**
22: **end for**
23: **return** $\mathcal{E}$
24: **end procedure**

Theorem 2.3.7. *The number of arithmetic operations of Algorithm 6 (*Float_Enclosing-Edges_MultiEval-TaylorApprox*) is* $O(md^2N + mN^2\log_2(N) + d^3)$.

Proof. The partial evaluation of $P(X,Y) = \sum_{s=0}^{d}(\sum_{r=0}^{d} a_{r,s}X^r)Y^s$ in X is the evaluation of the $d+1$ polynomials $\sum_{r=0}^{d} a_{r,s}X^r$. In Line 4, each of these polynomials is evaluated via the $\square$FME at all the Chebyshev nodes. The cost is $d+1$ times the cost of one $\square$FME of size N, that is $O(d(d^2 + N\log_2(N)))$ according to Corollary 4. In the outer for loop of Line 7, for each of

the N vertical fibers, degree m Taylor approximations are computed with Subroutine 7 at all Chebyshev nodes for a cost of $O(md^2 + mN\log_2(N))$ according to Lemma 2.3.5. In Line 10, the computation of the maximum absolute value of the coefficients of $P(c_i, Y)$ is in $O(d)$. In the for inner loop of Line 11, for each c_j the Taylor approximations are evaluated with Subroutine 8 for a cost of $O(m)$ according to Lemma 2.3.6. Then, the presence of zero in the interval $I_{i,j}$ is tested in order to collected edges with a constant cost. Subsequently, the cost of the inner loop is $O(mN)$. So, the whole outer loop has a cost of $O(N(md^2 + mN\log_2(N) + d + mN))$. Thus, the cost of the algorithm is $O(md^2N + mN^2\log_2(N) + d^3)$. □

Lemma 2.3.8. *Algorithm 6 does not return an edge $(c_i, [y_1, y_2])$ if $0 \notin \square P(c_i, [y_1, y_2]) + [-E, E]$ where*

$$\begin{aligned} E =& 8(d+1)e_{\mathrm{FME}}(P,d,n,p)\|(a_{i,j})\|_\infty + (d+1)^2\|(a_{i,j})\|_\infty \left(\frac{e}{2(m+1)}\right)^{m+1} \\ &+ 2e_{\mathrm{FME}}(P,d,n,p)^2\|(a_{i,j})\|_\infty, \end{aligned}$$

with e_{FME} defined in Theorem 2.1.6, assuming that $4d < N$. For a given c_j, the enclosing interval $[y_1, y_2]$ is defined in Subroutine 8.

Proof. Algorithm 6 takes $P(X,Y) = \sum_{r,s=0}^{d} a_{r,s}X^rY^s$ as an input. Then, in the for loop of Line 3 it is evaluated to get the $P(c_i, Y) = \sum b_{i,s}Y^s$ for $i \in [\![0, N-1]\!]$. However, the computation returns $\left(\sum \widehat{b}_{i,s}Y^s\right)_{i\in[\![0,N-1]\!]}$. According to Theorem 2.1.6, for a given s,

$$\left\|(\widehat{b_{i,s}})_i - (b_{i,s})_i\right\|_\infty \leq e_{\mathrm{FME}}(d,n,p)\|(a_{r,s})_r\|_\infty.$$

We can even write $\left\|(\widehat{b_{i,s}})_i - (b_{i,s})_i\right\|_\infty \leq e(P,d,n,p)$ and define the upper bound for the polynomial $e(P,d,n,p) = e_{\mathrm{FME}}(d,n,p)\|(a_{r,s})_{r,s}\|_\infty$.

Let us make some remarks that will help us to bound the error. For a univariate polynomial Q of degree d, such that $Q(y) = \sum_{s=0}^{d} q_s y^s$, we have

$$Q^{(k)}(y) = \sum_{s=k}^{d} q_s s\cdots(s-k+1)y^{s-k}.$$

Consequently,

$$\underline{\underline{\left|Q^{(k)}(y)\right| \leq \frac{d!}{(d-k)!}\sum_{s=k}^{d}|q_s||y|^{s-k}.}}$$

Also remark that

$$\frac{d!}{(d-k)!k!} = \frac{d(d-1)\cdots(d-k+1)}{k(k-1)\cdots 1} \leq d^k$$

and that

$$R \leq \max\left|\frac{c_i - c_{i+1}}{2}\right| < \frac{1}{2}\frac{\pi}{N}.$$

Therefore, since $\pi d < 4d < N$,

$$dR \leq \frac{\pi d}{2N} < \frac{N}{2N} = \frac{1}{2}.$$

So,

$$\underline{\underline{\frac{d!}{(d-k)!k!}R^k < d^k R^k < \left(\frac{1}{2}\right)^k.}}$$

Finally, the Stirling inequality formula states that for $n \geq 1$

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} < n!.$$

Thus,

$$\underline{\underline{\left(\frac{m+1}{e}\right)^{m+1} < (m+1)!.}}$$

We have to account for three error terms due to the use of the Taylor approximation, the FME operations for the evaluation in X and in Y. In the following expressions, for a given c_j, r is a value in $[-R, R]$ with $R = \max\left\{\frac{c_j - c_{j+1}}{2}, \frac{c_{j-1} - c_j}{2}\right\}$ (it is exactly defined in Subroutine 8). For conciseness, we define $P_{c_i}^{(k)}(y) = \frac{\mathrm{d}^k}{\mathrm{d}y^k}P(c_i, y)$.

- Taylor approximation
  With Taylor-Lagrange inequality, we have

$$\left|P(c_i, c_j + r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P_{c_i}^{(k)}(c_j)\right| \leq \max_{y \in [c_j - R, c_j + R]} \left|P_{c_i}^{(m+1)}(y)\right| \frac{R^{m+1}}{(m+1)!}.$$

  Note that

$$P(c_i, y) = \sum_{s=0}^{d} \underbrace{\left[\sum_{r=0}^{d} a_{r,s} c_i^r\right]}_{q_s} y^s$$

  and since $c_i \in [-1, 1]$

$$|q_s| \leq (d+1)\|(a_{i,j})_{i,j}\|_\infty.$$

  Given that $[c_j - R, c_j + R] \subset [-1, 1]$, with previous remarks, we get

$$\begin{aligned}\max_{y \in [c_j - R, c_j + R]} \left|P_{c_i}^{(m+1)}(y)\right| \frac{R^{m+1}}{(m+1)!} &\leq (d+1)^2 \underbrace{d \cdots (d-m)}_{\leq d^{m+1}} \|(a_{i,j})_{i,j}\|_\infty \frac{\overbrace{R^{m+1}}^{\leq \left(\frac{\pi}{2N}\right)^{m+1}}}{\underbrace{(m+1)!}_{\leq \left(\frac{e}{m+1}\right)^{m+1}}} \\ &\leq (d+1)^2 \|(a_{i,j})_{i,j}\|_\infty \left(\frac{ed\pi}{2(m+1)N}\right)^{m+1}.\end{aligned}$$

  Therefore,

$$\boxed{P(c_i, c_j + r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P_{c_i}^{(k)}(c_j) \in (d+1)^2 \|(a_{i,j})_{i,j}\|_\infty \left(\frac{e}{2(m+1)}\right)^{m+1} [-1, 1].}$$

- First FME
  We denote by $\widehat{P}$ the approximation of P when its first variable is evaluated with the FME. For all i and s there exists $\delta_{i,s}$ such that $|\delta_{i,s}| \leq e(P,d,n,p)$ and

$$\widehat{P}(c_i,y) = \sum_{s=0}^{d} \widehat{b}_{i,s} y^s = \sum_{s=0}^{d} (b_{i,s} + \delta_{i,s}) y^j s = \sum_{s=0}^{d} b_{i,s} y^j s + \sum_{s=0}^{d} \delta_{i,s} y^s = P(c_i,y) + \sum_{s=0}^{d} \delta_{i,s} y^s.$$

  In Line 4, the partial polynomial $\widetilde{P}(c_i,Y)$ with interval coefficients is constructed such that $\widehat{P}(c_i,Y) \in \widetilde{P}(c_i,Y)$, where $\in$ denotes a coefficient-wise set membership:

$$\begin{aligned}\tilde{b}_{i,s} &= \widehat{b}_{i,s} + [-e(P,d,n,p), e(P,d,n,p)] \\ &= b_{i,s} + \delta_{i,s} + [-e(P,d,n,p), e(P,d,n,p)].\end{aligned}$$

  Subsequently,

$$\begin{aligned}\widetilde{P}(c_i,y) &= \sum_{s=0}^{d} \widetilde{b}_{i,s} y^s \\ &= P(c_i,y) + \sum_{s=0}^{d} \delta_{i,s} y^s + \sum_{s=0}^{d} [-e(P,d,n,p), e(P,d,n,p)] y^s.\end{aligned}$$

  Since $\delta_{i,s} \in [-e(P,d,n,p), e(P,d,n,p)]$ and with $\subseteq$ which denotes a coefficient-wise inclusion,

$$\widetilde{P}(c_i,y) \subseteq P(c_i,y) + 2\sum_{s=0}^{d} [-e(P,d,n,p), e(P,d,n,p)] y^s.$$

  The error on the Taylor approximation obtained with $\widehat{P}$ can then be bounded using this inclusion, the previous remarks and the fact that $c_j \in [-1,1]$.

$$\begin{aligned}\sum_{k=0}^{m} \frac{1}{k!} r^k P_{c_i}^{(k)}(c_j) - \sum_{k=0}^{m} \frac{1}{k!} r^k \widetilde{P}_{c_i}^{(k)}(c_j) &\subseteq \sum_{k=0}^{m} \frac{1}{k!} r^k 2 \sum_{s=k}^{d} e(P,d,n,p)[-1,1] \frac{d!}{(d-k)!} c_j^s \\ &\subseteq 2\sum_{k=0}^{m} \frac{d!}{(d-k)!k!} r^k \sum_{s=0}^{d} e(P,d,n,p)[-1,1] \\ &\subseteq 2(d+1)e(P,d,n,p) \sum_{k=0}^{m} \frac{d!}{(d-k)!k!} R^k [-1,1] \\ &\subseteq 2(d+1)e(P,d,n,p) \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k [-1,1].\end{aligned}$$

  Hence,

$$\boxed{\sum_{k=0}^{m} \frac{1}{k!} r^k P_{c_i}^{(k)}(c_j) - \sum_{k=0}^{m} \frac{1}{k!} r^k \widetilde{P}_{c_i}^{(k)}(c_j) \subseteq 4(d+1)e(P,d,n,p)[-1,1].}$$

- Second FME
  We denote by $\widetilde{\widetilde{P}}^{(k)}$ the approximation of $P^{(k)}$ when its second variable is evaluated with

the FME, from the computation of $\widetilde{P}^{(k)}$. For all i, j and k there exists $\delta_{i,j,k}$ such that $|\delta_{i,j,k}| \leq e(\widetilde{P}, d, n, p)$ and

$$\widetilde{\widetilde{P}}^{(k)}_{c_i}(c_j) = \widetilde{P}^{(k)}_{c_i}(c_j) + \sum_{s=0}^{d} \delta_{i,j,k} y^s.$$

Moreover, from Section 2.1.2

$$\begin{aligned} e(\widetilde{P}, d, n, p) &= e_{\mathrm{FME}}(d, n, p) \left\| (\widehat{b}_{i,s})_i \right\|_\infty + (d+1) e(P, d, n, p) \\ &= e_{\mathrm{FME}}(d, n, p) \|(b_{i,s} + \delta_{i,s})_i\|_\infty + (d+1) e(P, d, n, p). \end{aligned}$$

Since

$$|b_{i,s}| = \left| \sum_{r=0}^{d} a_{r,s} c_i^r \right| \leq (d+1) \|a\|_\infty$$

and

$$|\delta_{i,s}| \leq e(P, d, n, p) = e_{\mathrm{FME}}(d, n, p) \|a\|_\infty,$$

by triangular inequality we have

$$e(\widetilde{P}, d, n, p) \leq 2(d+1) e_{\mathrm{FME}}(d, n, p) \|a\|_\infty + e_{\mathrm{FME}}(d, n, p)^2 \|a\|_\infty.$$

So,

$$\begin{aligned} \sum_{k=0}^{m} \frac{1}{k!} r^k \widetilde{P}^{(k)}_{c_i}(c_j) - \sum_{k=0}^{m} \frac{1}{k!} r^k \widetilde{\widetilde{P}}^{(k)}_{c_i}(c_j) &\subseteq \sum_{k=0}^{m} \frac{1}{k!} r^k e(\widetilde{P}, d, n, p)[-1, 1] \\ &\subseteq e(\widetilde{P}, d, n, p) \sum_{k=0}^{m} r^k [-1, 1]. \end{aligned}$$

If $1 < d$, then $\pi < 4d < N$, so

$$\sum_{k=0}^{m} |r|^k \leq \sum_{k=0}^{\infty} R^k \leq \sum_{k=0}^{\infty} \left(\frac{\pi}{2N} \right)^k = \frac{1}{1 - \frac{\pi}{2N}} \leq 2.$$

Thus,

$$\boxed{\sum_{k=0}^{m} \frac{1}{k!} r^k \widetilde{P}^{(k)}_{c_i}(c_j) - \sum_{k=0}^{m} \frac{1}{k!} r^k \widetilde{\widetilde{P}}^{(k)}_{c_i}(c_j) \subseteq 2e(\widetilde{P}, d, n, p)[-1, 1].}$$

The algorithms computes $\sum_{k=0}^{m} \frac{1}{k!} [y_1, y_2]^k \widetilde{\widetilde{P}}^{(k)}_{c_i}(c_j)$. Thus, with Corollary 1 these three results prove the lemma.

□

One constructs a pixel drawing by lighting the pixels adjacent to the edges returned by Algorithm 6. When the evaluation on $c_i \times I_{c_j}$ contains a zero, the four pixels $[c_{i+1}, c_i] \times [c_{j+1}, c_j]$, $[c_{i+1}, c_i] \times [c_j, c_{j-1}]$, $[c_i, c_{i-1}] \times [c_{j+1}, c_j]$ and $[c_i, c_{i-1}] \times [c_j, c_{j-1}]$ are selected. For example, Figure 2.5 shows that if a zero is detected on $c_3 \times I_{c_3}$, then the four pixels $[c_4, c_3] \times [c_4, c_3]$, $[c_4, c_3] \times [c_3, c_2]$, $[c_3, c_2] \times [c_4, c_3]$ and $[c_3, c_2] \times [c_3, c_2]$ to be lighted.

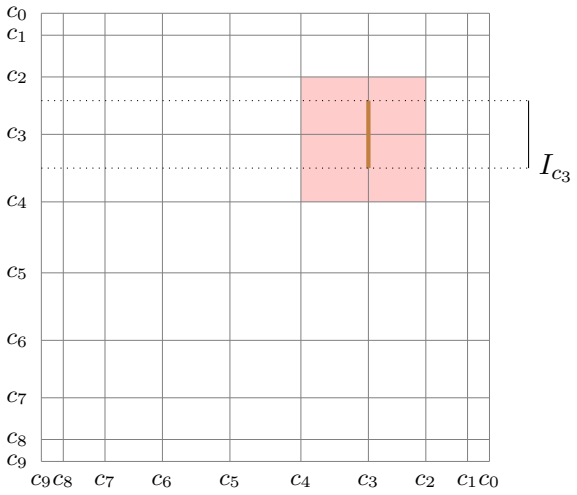


Figure 2.4: If an intersection is detected on $c_3 \times I_{c_3}$ (brown line), then the four pixels overlapping $[c_4, c_2] \times [c_4, c_2]$ are lighted (light red).

2.4 A pixel enclosing algorithm

By going one step further, we can even capture features of the curves inside pixels, not only along the fibers. Doing so, we are able to detect the false negative pixels of the curve missed by previous algorithms. Those pixels contain connected components which lie inside pixels. So the drawing is now actually an enclosure of the curve. The challenge is to get it as tight as possible. We improve upon Algorithm 5. Instead of evaluating the polynomial at points and then doing subdivisions along fibers of the form $x = c_i$, we now evaluate the polynomial on intervals and then do subdivisions along stripes of the form $x \in I_{c_i}$. More specifically, we evaluate the first variable on intervals around the Chebyshev nodes (c_i) using multipoint evaluations and Taylor approximation, then we do subdivisions along the second variable for polynomial with interval coefficients. Algorithm 7 uses these evaluations to select the pixels with potential zeros. When the evaluation on $I_{c_i} \times I$ contains a zero, it selects the two pixels , $[c_{i+1}, c_i] \times I$ and $[c_i, c_{i-1}] \times I$. For example, Figure 2.5 shows that if a zero is detected on $I_{c_3} \times [0.4, 0.6]$, then Algorithm 7 returns the two pixels $[c_4, c_3] \times [0.4, 0.6]$ and $[c_3, c_2] \times [0.4, 0.6]$ to be lighted.

Theorem 2.4.1. *The number of arithmetic operations of Algorithm 7 (*Float_Enclosing-Pixels_TaylorApprox-Subd*) is $O(md^3 + mdN \log_2(N) + dNT_{max})$, where T_{max} is the size of the largest evaluation tree in the process.*

Proof. Algorithm 7 has the same structure as Algorithms 4 and 5. The partial evaluation of $P(X, Y) = \sum_{s=0}^{d} (\sum_{r=0}^{d} a_{r,s} X^r) Y^s$ in X is the evaluation of the $d+1$ polynomials $\sum_{r=0}^{d} a_{r,s} X^r$. For each one, the Taylor approximation of degree m is computed using Algorithm 7 with a cost of $O(md^2 + mN \log_2(N))$ according to Lemma 2.3.5. The results are used to compute the evaluation around each of the N Chebyshev nodes. With Algorithm 8 each evaluation costs $O(m)$ according to Lemma 2.3.6, so for all the nodes the cost is $O(mN)$. Thus, for one partial polynomial the cost is $O(md^2 + mN \log_2(N) + mN) = O(md^2 + mN \log_2(N))$. Therefore, for all the partial polynomials the partial evaluations costs $O(md^3 + mdN \log_2(N))$. In Line 14, in each vertical stripe, the subdivision function Subroutine 2 performs $O(T)$ interval evaluations of the partially evaluated univariate polynomial of degree d. Then the intervals $\mathcal{I}$ are used to

Algorithm 7 Fast pixel enclosing with subdivision.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{r,s} X^r Y^s$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$, a Chebyshev grid resolution integer $N > 0$ and the order m of the Taylor approximation.
Output: A set of pixels $\mathcal{P}$, enclosing the curve represented by P

1: **procedure** FLOAT_ENCLOSINGPIXELS_TAYLORAPPROX-SUBD($\mathbf{a}$, N, m)
2: $\quad\mathbf{b} \in \mathbb{IF}[X]^{N\times(d+1)}$
3: $\quad$**for** $s \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{s=0}^{d} b_{i,s} Y^t = P(I_{c_i}, Y)$
4: $\quad\quad P_s \leftarrow \sum_{r=0}^{d} a_{r,s} X^r$
5: $\quad\quad T_{0,s}, \ldots, T_{N-1,s} \leftarrow$ FLOAT_TAYLORAPPROXIMATIONS(P_s, N, m)
6: $\quad\quad a_{max} \leftarrow \max_{0\le r\le d} |a_{r,s}|$
7: $\quad\quad$**for** $i \leftarrow 0$ **to** $N-1$ **do**
8: $\quad\quad\quad b_{i,s} \leftarrow$ FLOAT_EVALAROUNDCHEBNODE($T_{i,s}$, m, i, a_{max}, N)
9: $\quad\quad$**end for**
10: $\quad$**end for**
11: $\quad\mathcal{P} \leftarrow \emptyset$
12: $\quad$**for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Subdivision in vertical tubes $X = I_{c_i}$
13: $\quad\quad Q : Y \longmapsto \sum_{s=0}^{d} b_{i,s} Y^s$
14: $\quad\quad\mathcal{I} \leftarrow$ FLOAT_ISOLATE1D(Q, N, 0, $N-1$)
15: $\quad\quad$**switch** i **do**
16: $\quad\quad\quad$**case** 0
17: $\quad\quad\quad\quad$**for** $I \in \mathcal{I}$, Add $([c_{i+1}, c_i], I)$ to $\mathcal{P}$
18: $\quad\quad\quad$**case** $N-1$
19: $\quad\quad\quad\quad$**for** $I \in \mathcal{I}$, Add $([c_i, c_{i-1}], I)$ to $\mathcal{P}$
20: $\quad\quad\quad$**default**
21: $\quad\quad\quad\quad$**for** $I \in \mathcal{I}$, Add $([c_{i+1}, c_i], I)$ and $([c_i, c_{i+1}], I)$ to $\mathcal{P}$
22: $\quad\quad$**end switch**
23: $\quad$**end for**
24: $\quad$**return** $\mathcal{P}$
25: **end procedure**

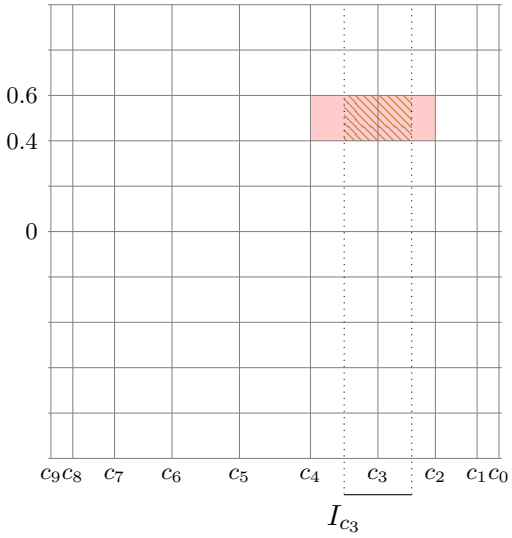


Figure 2.5: If an intersection is detected on $I_{c_3} \times [0.4, 0.6]$ (brown hatched lines), then the two pixels overlapping $[c_4, c_2] \times [0.4, 0.6]$ are lighted (light red).

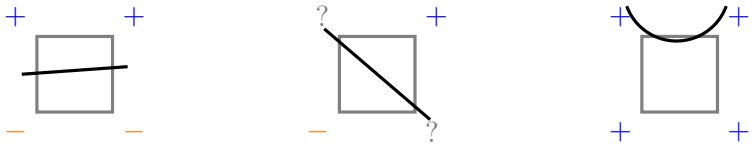


Figure 2.6: Three examples of sign configurations. The pixel is in gray and the curve crossing or passing nearby is in black. The sign of the evaluation is at each corner: + (strictly positive), − (strictly negative) and ? (contains zero). The first two pixels are black and the third one is red.

collect pixels with a constant cost. The complexity for all the fibers is thus $O(NdT)$. Hence the global complexity of $O(md^3 + mdN \log_2(N) + dNT_{max})$. □

Algorithm 7 returns pixels which enclose the curve. With Algorithm 8 it is possible to post-process that output and classify the pixels. We color pixels crossed by the curve for sure in black, pixels not crossed by the curve for sure in white and undecided pixels in red. We call that a *colored pixel drawing*. The algorithm detects whether the curve crosses the pixel, by detecting whether the polynomial P representing the curve changes signs from one vertex to another. If the evaluations $\square P(v_1)$ and $\square P(v_2)$ at two different vertices, v_1 and v_2, of a pixel have different signs, then P vanishes on the pixel. In other words, the curve crosses the pixel, so it is black. Otherwise, the pixel is red. This choice of colors is similar to what Tupper does [Tup01]. Figure 2.6 illustrates three cases. The first two pixels are black because they both have at least one vertex with a strictly positive evaluation and one with a strictly negative evaluation. The third one is red, because it does not satisfy this condition. In Figure 2.7, the sign of the evaluation at a vertex is represented with colored dots. We use the same scheme as in Figure 2.6, the color of a vertex depends on the sign of its evaluation: blue if it is strictly positive, yellow if is it strictly negative and gray if it contains zero. Based on that, the pixels are colored in black, red or white.

Lemma 2.4.2. *The number of arithmetic operations of Algorithm 8 is* $O(d^3 + dN \log_2(N) + dS)$,

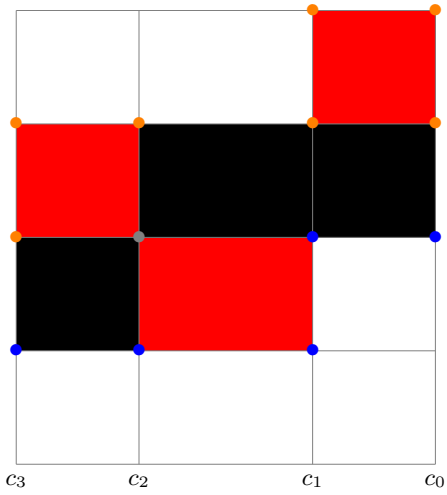


Figure 2.7: A colored pixel drawing. Vertices are colored with respect to the sign of the evaluation at that point: strictly positive (•), strictly negative (•) or the evaluation contains zero (•). A pixel is black when two vertices of the pixel are respectively strictly positive and strictly negative. A pixel is red when it is part of the enclosure, but is not black. A pixel is white when it is not part of the enclosure.

Algorithm 8 Fast colored pixel drawing.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{r,s} X^r Y^s$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$ and a set of pixels $\mathcal{P}$.
Output: A set of black pixels $\mathcal{P}_b$ and a set of red pixels $\mathcal{P}_r$

1: **procedure** FLOAT_COLOREDPIXELS($\mathbf{a}$, $\mathcal{P}$)
2: $\quad\mathbf{b} \in \mathbb{IF}^{N\times(d+1)}$
3: $\quad$**for** $s \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{s=0}^{d} b_{i,s} Y^s = P(c_i, Y)$
4: $\quad\quad b_{0,j}, \ldots, b_{N-1,j} \leftarrow$ FLOAT_FME($(a_{0,j}, \ldots, a_{d,j})$,$N$)
5: $\quad$**end for**
6: $\quad$**for** $i \leftarrow 0$ **to** $N-1$ **do**
7: $\quad\quad P_i : Y \longmapsto \sum_{s=0}^{d} b_{i,s} Y^s$
8: $\quad$**end for**
9: $\quad\mathcal{P}_b, \mathcal{P}_r \leftarrow \emptyset, \emptyset$
10: $\quad$**for all** $[c_i, c_{i-1}] \times [y_1, y_2] \in \mathcal{P}$ **do** ▷ Decide if a pixel is black or red
11: $\quad\quad V \leftarrow \{\square P_i(y_1), \square P_i(y_2), \square P_{i-1}(y_1), \square P_{i-1}(y_2)\}$
12: $\quad\quad b_+ \leftarrow \exists\, v \in V,\ v > 0$
13: $\quad\quad b_- \leftarrow \exists\, v \in V,\ v < 0$
14: $\quad\quad$**if** b_+ and b_- **then**
15: $\quad\quad\quad$Add $[c_i, c_{i-1}] \times [y_1, y_2]$ to $\mathcal{P}_b$
16: $\quad\quad$**else**
17: $\quad\quad\quad$Add $[c_i, c_{i-1}] \times [y_1, y_2]$ to $\mathcal{P}_r$
18: $\quad\quad$**end if**
19: $\quad$**end for**
20: $\quad$**return** $\mathcal{P}_b, \mathcal{P}_r$
21: **end procedure**

when it is called on FLOAT_COLOREDPIXELS*(P, $\mathcal{P}$) with P of partial degree d and $\mathcal{P}$ of size S.*

Proof. The first step is to compute the partial polynomials. The $d+1$ partial polynomials are evaluated using the $\square$FME. So, for all of them the cost is $O(d(d^2 + N\log_2(N)))$. Then for each pixel of $\mathcal{P}$, four polynomial evaluations of degree d are performed at Line 11. Each one has a complexity of $O(d)$. In Lines 12 and 13, the sign of four values are tested twice with a constant cost. The final boolean evaluation also has a constant cost. So, the complexity is $O(d)$. So that cost of the decision step is $O(Sd)$. $\square$

In practice, we get an enclosure of the curve with Algorithm 7 and apply Algorithm 8 to that output. So, using the notations of Theorem 2.4.1 the output $\mathcal{P}$ is of size $S = O(dT_{max})$ (where T_{max} is defined in Note of Section 1.2.3). Thus, the cost of Algorithm 7 dominates the cost of Algorithm 8, hence the latter gives the complexity of the whole process.

Corollary 5. *When Algorithm 7 is called on* FLOAT_ENCLOSINGPIXELS_TAYLORAPPROXSUBD*(P, N, m) to get $\mathcal{P}$ and then Algorithm 8 is called on* FLOAT_COLOREDPIXELS*(P, $\mathcal{P}$), the number of arithmetic operations of the whole process is $O(md^3 + mdN\log_2(N) + dNT_{max})$, where P is a bivariate polynomial of partial degree d and where T_{max} is the size of the largest evaluation tree in the process.*

3

Experiments

We have presented algorithms to draw the enclosing edges and the enclosing pixels of an algebraic curve in Chapter 2. The procedures have long and explicit names. In this chapter, we associate those algorithms with a pixel coloring method and label the pair as Algorithm ES, Algorithm ET and Algorithm PS. The first two compute the enclosing edges with subdivision and Taylor approximation. The last one computes the enclosing pixels. In this chapter, we use the names of the algorithms to refer to their implementation, and occasionally to the algorithms themselves.

Algorithm ES
FLOAT_ENCLOSING**E**DGES_MULTIEVAL-**S**UBD (2-pass version)
pixel selection (see Figure 2.2) and coloring

Algorithm ET
FLOAT_ENCLOSING**E**DGES_MULTIEVAL-**T**AYLORAPPROX (2-pass version)
pixel selection (see Figure 2.4) and coloring

Algorithm PS
FLOAT_ENCLOSING**P**IXELS_TAYLORAPPROX-**S**UBD
FLOAT_COLOREDPIXELS

For Algorithm ES and Algorithm ET, we do not provide an algorithm in pseudo-code explaining how pixels are selected and colored. We simply do that here in natural language. Both algorithms return edges. The pixels adjacent to those edges are lighted according to Figure 2.2 or Figure 2.4. The scheme used in FLOAT_COLOREDPIXELS is also used to determine the color of the pixels: if the evaluation of the polynomial defining the curve has different signs at two vertices of the same pixel, then that pixel is black, otherwise it is red.

Note. Red and black pixels have the same meaning in the drawings of these three algorithms. However, white pixels do not. On the one hand, white pixels from Algorithm ES and Algorithm ET may contain connected components of the curve inside them, on the other hand, white pixels from Algorithm PS do not contain the curve at all. Therefore, only the latter returns a certified drawing.

First, we verify that the speed-up of the FME is practical and improves the computation time (Section 3.1). Then, we compare Algorithm ES and Algorithm ET and check that their computation times correspond to their theoretical complexities (Section 3.2). Finally, we compare

all three implementations to state-of-the-art software. We extend the experimentation to higher resolution drawings for Algorithm ES (Section 3.3).

Our algorithms are implemented in Python and use the interval arithmetic implementation of Arb for Python [Joh13]. For Algorithm ET, one can choose the order m of the Taylor approximation. The higher the order of the approximation is, the higher the degree of the polynomial is. Subsequently, the higher m is, the longer it takes to evaluate the approximation. In our experiments, increasing m made the computation time longer, without any significant improvement of the drawings. So, we fix $m = 3$. A piece of software is available at `https://gitlab.inria.fr/nherathm/certified2dvisualization`.

Among the state-of-the-art implementations for drawing implicit curves, we have selected `ImplicitEquations` [Imp], the marching squares from `scikit` and the implicit function plotting of MATLAB. `ImplicitEquations` is based on an algorithm by Tupper [Tup01] and returns an enclosure of the algebraic curve. Therefore, we can compare it to Algorithm PS. Our choice of colors for the drawings come from that algorithm. An implementation of the marching squares algorithm [LC87] is provided by `skimage.measure.find_contours` [vdWSN+14]. It takes a scalar field as an input. We provide it the evaluations of our polynomial on the grid that we compute using `polyval1d` from the `numpy` package [HMvdW+20]. So, timings in Section 3.3 correspond to the call to both methods. We also examine the behavior of `fimplicit` from MATLAB. We were not able to find any documentation on the algorithm, but we suspect that it also relies on an evaluation and a marching square algorithm. All the measures are CPU times in seconds.

Our dataset is composed of slightly modified Kac and KSS polynomials of the form

$$\sum_{0 \leq r+s \leq d} w_{r,s} a_{r,s} X^r Y^s$$

of total degree d where the $a_{r,s}$ are *uniformly distributed* in $[\![-100, 100]\!]$ and where $w_{r,s}$ depends on the family and is defined in Section 6. Our input polynomials are named *random_ d_ weight* where d is the total degree and *weight* is either "kac" or "kss" depending on the family of the polynomial.

3.1 Speed-up of the FME

We introduce Idea 3 to get from Algorithm 3 to Algorithm 4 by claiming that the multipoint evaluation speeds up the computation despite the introduction of a change of basis step in the algorithm. We verify that here.

Figure 3.1 displays the computation times of the partial evaluation of our polynomials using $\square$FME, with a complexity of $O(d^2 + N \log_2(N))$ from Lemma 1.2.8, and using Horner's method, with a complexity of $O(d^2 N)$. A given polynomial corresponds to one color, the dashed lines correspond to the $\square$FME and the solid lines to Horner's method. For the range of values we are interested in, specifically $10 \leq d \leq 100$ and $5{,}000 < N$ the $\square$FME is always faster. This justifies the use of the FME despite the preliminary change of basis, which contributes to the complexity with the term d^2.

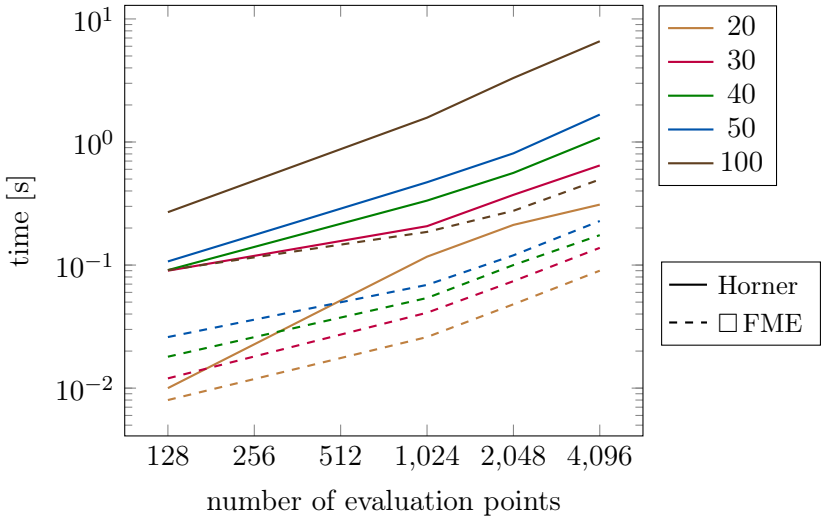


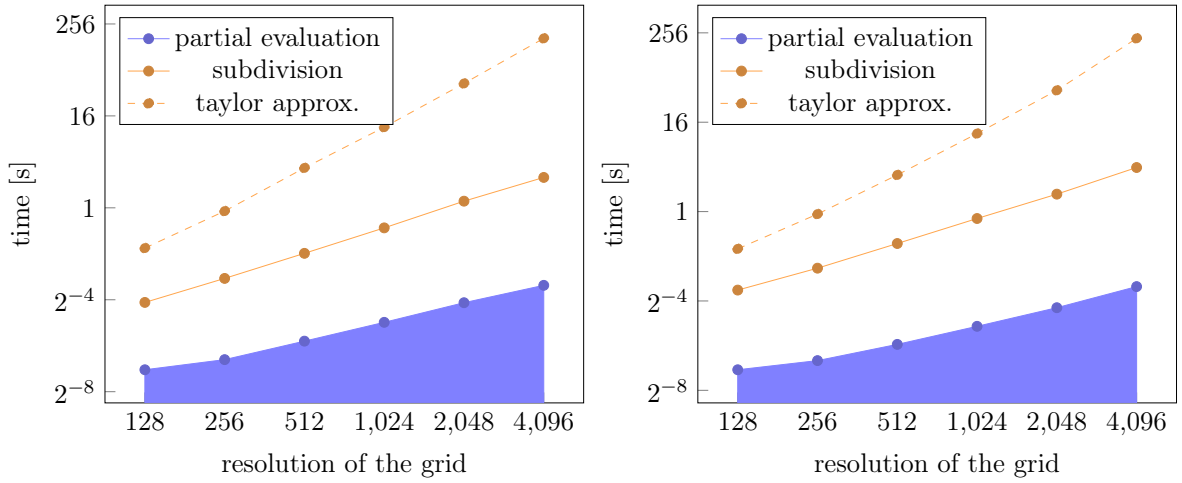
Figure 3.1: Computation times of the □FME and the interval Horner evaluation with respect to the number of evaluation points on a log-log plot. The colors in the legend indicate the degree of Kac polynomials. Dotted lines correspond to the □FME and solid lines to the interval Horner scheme.

3.2 Comparison of our two approaches

Let us now compare our two methods, Algorithm 6 (Float_EnclosingEdges_MultiEval-TaylorApprox) and Algorithm 5 (Float_EnclosingEdges_MultiEval-Subd). They are composed of two main steps:

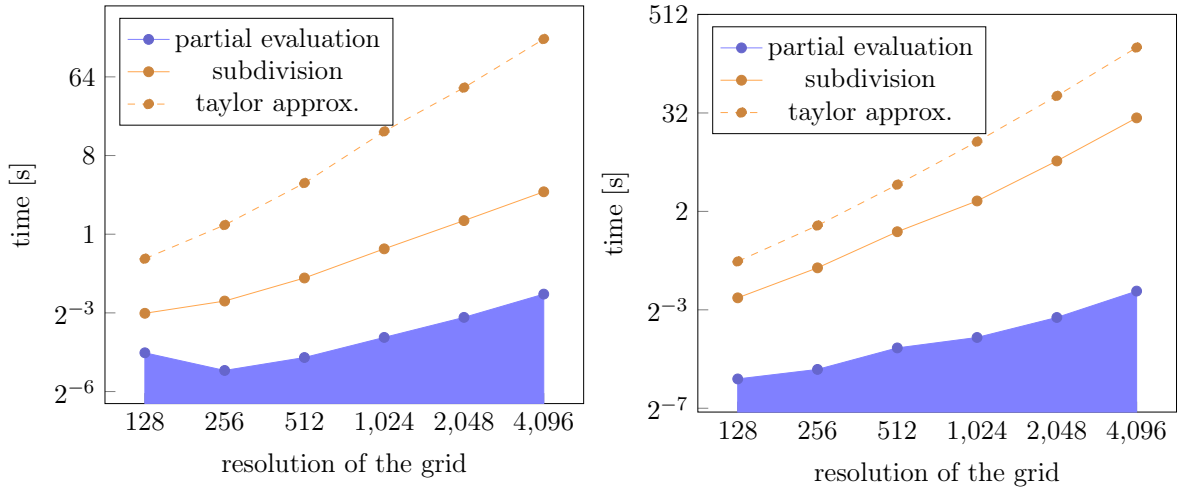
- a partial evaluation and
- a Taylor approximation or a subdivision process.

We record the computation times of those specific steps and exclude the loading of the polynomial and the drawing. Figures 3.2 and 3.3 show log-log graphs for different input polynomials. They display cumulative times for the two steps. Both methods share a common partial evaluation step which is represented in blue, the second step is in orange. The slopes of these plots give the power in the complexities with respect to N. In these examples, the subdivision is faster than the Taylor approximation. Moreover, except for Figure 3.3b, the slopes are around 2 for Algorithm 6 and around 1 for Algorithm 5. The reason *random_40_kss* does not behave as expect is due to the weights $w_{i,j}$ of KSS polynomials when the degree d exceeds 30 and is further detailed in Section 3.3. The other slopes confirm the expected result from the complexity analysis of Theorems 2.3.7 and 2.3.2. Indeed, since $d^2 \lesssim N \log_2(N)$ and assuming $T = O(\log_2(N))$, the complexities are respectively $O(\boldsymbol{N^2} \log_2(N))$ and $O(d\boldsymbol{N} \log_2(N))$.



(a) *random_20_kac.* (b) *random_20_kss.*

Figure 3.2: Cumulative computing time for a polynomial of total degree 20 weighted according to the two families.



(a) *random_40_kac.* (b) *random_40_kss.*

Figure 3.3: Cumulative computing time for a polynomial of total degree 40 weighted according to the two families.

3.3 Comparison to state-of-the-art implementations

3.3.1 Comparison at relatively high resolutions

Let us now compare our methods to some state-of-the-art implementations. We choose to measure the computation times that it takes to run the different programs, compute the drawings and save them in PNG files, without displaying them. In Tables 3.1, 3.2, 3.3, 3.4 and 3.5, all the implementations are tested respectively for *random_20_kac*, *random_40_kac*, *random_100_kac*, *random_20_kss* and *random_40_kss*. They all lead to the same conclusions about the different programs. As a reminder, `numpy + sckikit` and MATLAB compute an approximated drawing, while `ImplicitEquations`, Algorithm ET, Algorithm ES and Algorithm PS provide different levels of guarantees on the drawing. The latter three are our implementations and are highlighted in orange in the tables. For `ImplicitEquations` and MATLAB, we stopped the computation after 900 seconds at some resolutions for each example polynomial. As stated in Section 3.2, Algorithm 6 is slower than Algorithm 5, leading Algorithm ET to be slower than Algorithm ES. That is verified by these experiments as well. Moreover, in these tables, the methods using the subdivision (Algorithm ES and Algorithm PS) are slightly faster than scikit up to a resolution of 1024, despite the fact that our code may be slowed by Python limitations. Our programs perform worse for KSS polynomial. The reason is twofold. One the one hand, because of the size of the coefficients of KSS polynomials, our methods are unable to control the uncertainty well enough to eliminate a large number of pixel. We are then left with a large number of red pixels. One the other hand, for high resolutions, the bottleneck of our programs is the implementation of the visualization, which does not scale well currently and needs further optimization. This is aggravated by the large number of undecided pixels for KSS polynomials.

3.3.2 Quality of the output drawings

Figure 3.4 presents the plots of *random_20_kac* with Algorithm PS and `ImplicitEquations`. Our algorithm returns a tighter enclosure. Figure 3.5 presents the plots of the KSS polynomial *random_20_kss* for the other implementations, at a higher resolution of $N = 1024$. Compared to the plot of Figure 3.4a, the non-uniformity of the Chebyshev grid is no longer visible at this resolution and the outputs look similar.

Table 3.1: Computation times for *random_20_kac* (in seconds), with our implementations highlighted in orange.

N	128	256	512	1,024	2,048	4,096
`numpy + scikit`	5.8	5.7	5.8	5.9	6.3	7.3
MATLAB	12	19	49	167	636	>900
`ImplicitEquations`	281	599	>900	>900	>900	>900
ET	3.3	4	6.4	15	46	170
ES	4.2	3.3	3.7	4.3	5.7	9.1
PS	3.2	3.5	4.1	5.1	7.1	11

Table 3.2: Computation times for *random_40_kac* (in seconds), with our implementations highlighted in orange.

N	128	256	512	1,024	2,048	4,096
`numpy + scikit`	5.8	6.0	6.0	6.1	6.3	7.5
MATLAB	20	49	169	651	>900	>900
`ImplicitEquations`	>900	>900	>900	>900	>900	>900
ET	3.6	4.2	6.8	15	47	177
ES	3.1	3.3	3.7	4.3	5.6	8.3
PS	3.2	3.4	3.9	4.6	6	9

Table 3.3: Computation times for *random_100_kac* (in seconds), with our implementations highlighted in orange.

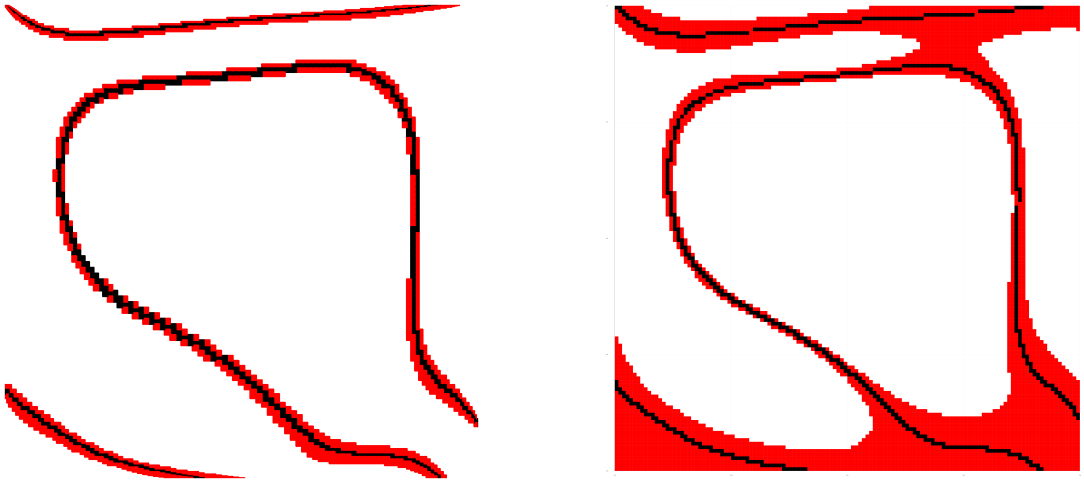
N	128	256	512	1,024	2,048	4,096
`numpy + scikit`	6.2	5.7	5.8	6.2	6.6	9.4
MATLAB	79	283	>900	>900	>900	>900
`ImplicitEquations`	>900	>900	>900	>900	>900	>900
ET	4.7	5.8	9.4	19	56	208
ES	4.1	4.3	4.7	5.4	6.7	9.5
PS	4.3	4.5	4.9	5.6	7.2	10

Table 3.4: Computation times for *random_20_kss* (in seconds), with our implementations highlighted in orange.

N	128	256	512	1,024	2,048	4,096
`numpy + scikit`	5.8	5.8	5.8	5.9	6.2	7.1
MATLAB	11	18	48	165	638	>900
`ImplicitEquations`	693	>900	>900	>900	>900	>900
ET	3.4	4.3	7.5	19	69	>900
ES	3.1	3.4	3.9	4.8	6.4	10
PS	4.6	3.7	4.3	5.3	7.7	12

Table 3.5: Computation times for *random_40_kss* (in seconds), with our implementations highlighted in orange.

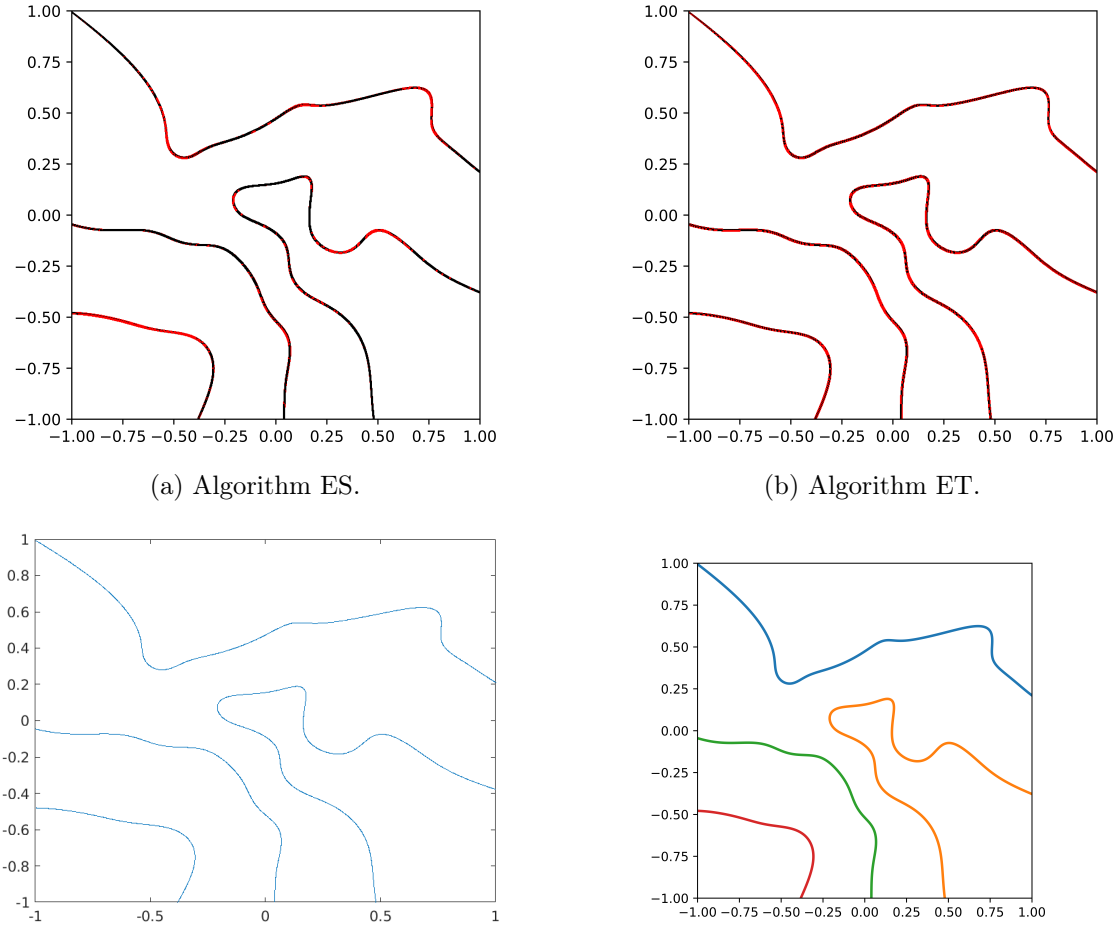
N	128	256	512	1,024	2,048	4,096
`numpy + scikit`	5.8	5.8	5.8	6.0	6.3	7.6
MATLAB	19	50	171	656	>900	>900
`ImplicitEquations`	>900	>900	>900	>900	>900	>900
ET	3.8	5.3	11	32	119	>900
ES	3.6	4.4	6.5	14	41	150
PS	4.9	4.4	5.9	8.9	19	43



(a) Algorithm PS.

(b) `ImplicitEquations`.

Figure 3.4: Drawings of *random_20_kac* for $N = 128$.



(a) Algorithm ES.

(b) Algorithm ET.

(c) MATLAB.

(d) `scikit`.

Figure 3.5: Drawings of *random_20_kss* for $N = 1,024$.

Table 3.6: Computation times of **Algorithm ES** for different families of polynomials with respect to the resolution (in seconds).

N	128	256	512	1,024	2,048	4,096	8,192	16,384	32,768
random_20_kac	3.0	3.4	3.7	4.3	5.5	8.1	13	24	47
random_30_kac	3.0	3.3	3.5	4.1	5.0	7.0	11	20	38
random_40_kac	3.1	3.3	3.6	4.3	5.4	8.1	13	24	47
random_50_kac	5.3	4.2	4.4	5.1	6.2	8.5	13	23	44
random_100_kac	4.1	4.3	4.7	5.3	6.7	9.4	15	26	49
random_20_kss	3.1	3.4	3.9	4.8	6.4	9.9	17	32	64
random_30_kss	3.2	3.5	4.2	5.3	8.2	16	43	185	>900
random_40_kss	4.2	4.5	5.1	6.2	8.7	15	36	145	718

Table 3.7: Computation times of **Algorithm PS** for different families of polynomials with respect to the resolution (in seconds).

N	128	256	512	1,024	2,048	4,096	8,192	16,384	32,768
random_20_kac	3.1	3.5	3.9	4.7	6.4	10	17	31	59
random_30_kac	3.1	3.3	3.7	4.3	5.4	7.7	12	22	40
random_40_kac	3.2	3.5	3.9	4.6	6.0	9.1	15	27	50
random_50_kac	3.0	3.3	3.6	4.2	5.5	8.2	14	25	47
random_100_kac	4.3	4.6	5.0	5.8	7.3	10	17	30	55
random_20_kss	3.3	3.7	4.3	5.4	7.7	12	21	39	76
random_30_kss	4.8	3.9	4.6	6.0	8.8	15	29	73	322
random_40_kss	3.4	4.2	6.4	14	42	150	>900	>900	>900

3.3.3 Comparison at very high resolution

The marching squares from `scikit + numpy`, Algorithm ES and Algorithm PS are the fastest at all the resolutions that we have tested. So, we test them at even higher resolutions in Tables 3.6, 3.7 and 3.8. For Kac polynomials, the two methods are comparable up to a resolution of $16,384$ and our method becomes faster for higher resolutions. On the other hand, our method faces stability issues for KSS polynomials when the resolution and the degree increase: degree 20 and 30 polynomials are computed in similar times, but not the degree 40 one. This is explained by the sensitivity of the IDCT to the size of the input coefficients. The error bound presented in Table 2.2 becomes insufficient to control the drawing. Indeed, for *random_40_kss* with $N = 8,192$, Theorem 2.1.6 yields $\|x - \widehat{x}\|_\infty \lesssim 1$, because the magnitude of the coefficients ranges from 1 to 10^{19} due to the KSS weights.

For completeness, we also included in Table 3.9 the timings for Isotop,[7] a state-of-the-art software based on CAD [CLP+10], that computes a drawing with a guaranteed topology and a low-resolution.

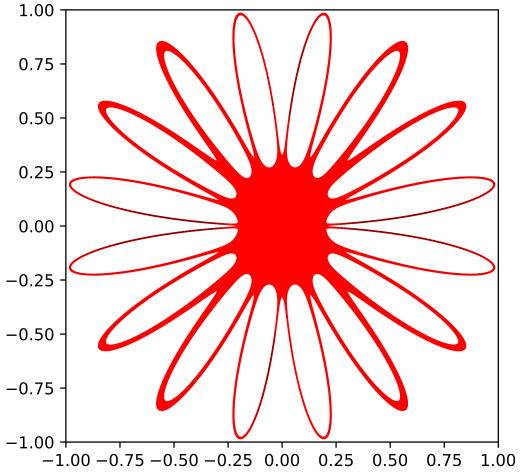
[7] https://isotop.gamble.loria.fr

Table 3.8: Computation times of **`scikit + numpy`** for different families of polynomials with respect to the resolution (in seconds).

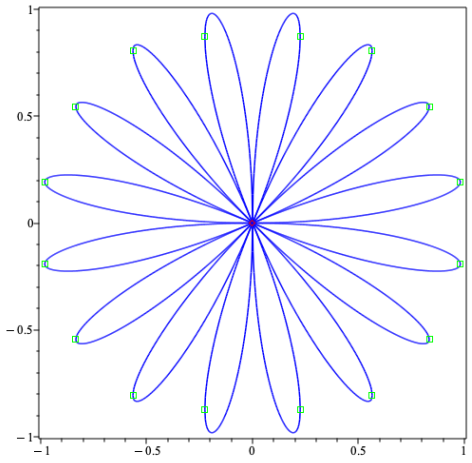
N	128	256	512	1,024	2,048	4,096	8,192	16,384	32,768
random_20_kac	5.9	5.8	5.8	5.9	6.2	7.2	11	29	83
random_30_kac	6.6	5.8	5.8	5.9	6.2	7.4	12	32	100
random_40_kac	6.5	5.9	5.8	5.8	6.3	7.6	12	35	116
random_50_kac	3.2	3.4	3.8	4.5	6.0	9.0	15	28	53
random_100_kac	6.4	5.7	5.8	6.0	6.5	8.5	17	49	232
random_20_kss	6.6	5.8	5.9	5.8	6.2	7.3	11	29	78
random_30_kss	6.7	5.9	5.8	5.8	6.3	7.4	12	32	102
random_40_kss	4.8	4.4	5.8	8.7	17	43	133	511	>900

Table 3.9: Computation times of Isotop for different families of polynomials (in seconds).

	Isotop ($N \sim 1,000$)	PS ($N = 16,384$)
random_20_kac	2.3	31
random_30_kac	18	22
random_40_kac	81	27
random_50_kac	1,603	25
random_100_kac	>2,000	30
random_20_kss	12	39
random_30_kss	183	73
random_40_kss	688	>900



(a) Algorithm PS ($N = 16,384$).



(b) Isotop ($N \approx 1,000$).

Figure 3.6: Drawings of $\text{dfold}_{8,1}$.

3.3.4 Drawing a singular curve

We have also tested the curve $\text{dfold}_{8,1}$ from Challenge 13 from Oliver Labs [Lab09], defined by

$$\begin{aligned} P(x,y) = &- x^{18} - 9x^{16}y^2 - 36x^{14}y^4 - 84x^{12}y^6 - 126x^{10}y^8 - 126x^8y^{10} \\ &- 84x^6y^{12} - 36x^4y^{14} - 9x^2y^{16} - y^{18} + 64x^{14}y^2 - 896x^{12}y^4 \\ &+ 4032x^{10}y^6 - 6400x^8y^8 + 4032x^6y^{10} - 896x^4y^{12} + 64x^2y^{14}. \end{aligned}$$

It contains an 8-fold singularity and has high tangencies between branches. For this curve, Isotop takes about one second while our subdivision method takes $16s$, $68s$ and $375s$ for respective resolutions of $1,024$, $4,096$ and $16,384$, in particular, it fails to discard pixels around the singularity.

Our approach combining FME and subdivision proves to be competitive in our experiments. Even though our guarantee is a weaker than the one provided by `ImplicitEquations`, our algorithm is faster. For all polynomials but *random_40_kss*, it is also significantly faster than the marching squares from scikit for high resolutions ($N \geq 16,384$), and has similar timings for the lower resolutions. The speed limitation of our implementation could be due to the Python language. We also think that the techniques that we presented could be used to speed-up existing algorithms such as marching squares or subdivision approaches.

Part II

Surface drawing in $\mathbb{R}^3$

1

An edge enclosing algorithm in 3D

We show how ideas used for plane curves in Part I can be extended to the three-dimensional case. We use fast evaluation techniques for the two first directions and reserve subdivision for the last one.

In this chapter, we show how to combine multipoint evaluation techniques and root isolation via subdivision in order to lower the number of evaluations in the marching cubes algorithm. The grid is made of fibers in the three directions. We locate the edges intersected by the surface with our algorithm. Then, we can identify a candidate subset of the voxels of the grid. By limiting the evaluation to that subset, it is possible to reduce the cost of the evaluation in the marching cubes algorithm.

1.1 Finding intersections with the fibers

We now consider a three-dimensional Chebyshev grid. Algorithm 9 computes intersections of a surface with the vertical fibers of the grid. The surface is given by an implicit equation of the form $P(x, y, z) = 0$. The polynomial $P(X, Y, Z)$ is first partially evaluated on Chebyshev nodes with respect to the first variable X using the FME. The resulting N bivariate polynomials $P(c_i, Y, Z)$ are then similarly evaluated with respect to the second variable Y. Then for each polynomial $P(c_i, c_j, Z)$ it is possible to find the vertical edges intersecting the surface with a subdivision process. That only gives intersections with vertical fibers. The algorithm should be applied three times to get the fibers in each direction, by permuting the roles of the variables each time: call it on $P(X, Y, Z)$ to get the vertical fibers, on $Q(X, Y, Z) = P(Z, Y, X)$ to get the fibers in the next direction and on $R(X, Y, Z) = P(X, Z, Y)$ to get the fibers in the last direction.

Theorem 1.1.1. *The number of operations of Algorithm 9 (*Float_EnclosingEdges_MultiEval-MultiEval-Subd*) is* $O(d^3N + dN^2(\log_2(N) + T)$. *If the surface intersects the grid* $\log_2(N) \leq T \leq N$, *otherwise* $0 \leq T \leq N$.

Proof. The partial evaluation of $P(X, Y, Z) = \sum_{t=0}^{d}(\sum_{s=0}^{d}(\sum_{r=0}^{d} a_{r,s,t}X^r)Y^s)Z^t)$ in X is the evaluation of the $(d+1)\times(d+1)$ polynomials $\sum_{r=0}^{d} a_{r,s,t}X^r$. In Line 5, each of these polynomials is evaluated via the FME at all the Chebyshev nodes. It returns the bivariate polynomials $\sum_{t=0}^{d}(\sum_{s=0}^{d} d_{i,s,t}Y^s)Z^t)$. The cost is $(d + 1) \times (d + 1)$ times the cost of one FME of size N, that is $O(d^2(d^2 + N\log_2(N)))$ according to Lemma 1.2.8. Then, its partial evaluation in Y is the evaluation of the $(d + 1) \times N$ polynomials $\sum_{s=0}^{d} d_{i,s,t}Y^r$. They are also evaluated using the

Algorithm 9 Fast 3D edge enclosing with subdivision.

Input: A bivariate polynomial $P(X,Y,Z) = \sum_{r,s,t} a_{r,s,t} X^r Y^s Z^t$ with $\mathbf{a} \in \mathbb{F}^{(d+1)\times(d+1)\times(d+1)}$ and a resolution N

Output: A set of edges $\mathcal{E}$, enclosing the intersections of the grid with the surface represented by P

1: **procedure** FLOAT_ENCLOSINGEDGES_MULTIEVAL-MULTIEVAL-SUBD($\mathbf{a}$, N)
2: $\mathbf{b} \in \mathbb{IF}^{N\times(d+1)}, \mathbf{d} \in \mathbb{IF}^{N\times N}$
3: **for** $t \leftarrow 0$ **to** d **do**
4: **for** $s \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{s,t} b_{i,s,t} Y^s Z^t = P(c_i, Y, Z)$
5: $b_{0,s,t}, \ldots, b_{N-1,s,t} \leftarrow$ FLOAT_FME($(a_{0,s,t}, \ldots, a_{d,s,t})$,$N$)
6: **end for**
7: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Partial evaluations $\sum_t d_{i,j,t} Z^t = P(c_i, c_j, Z)$
8: $d_{i,0,t}, \ldots, d_{i,N-1,t} \leftarrow$ FLOAT_FME($(b_{i,0,t}, \ldots, b_{i,d,t})$,$N$)
9: **end for**
10: **end for**
11: $\mathcal{S} \leftarrow \emptyset$
12: $\mathbf{q} \in \mathbb{IF}^{d+1}, \mathbf{p} \in \mathbb{IF}^{(m+1)\times N}$
13: **for** $i \leftarrow 0$ **to** $N-1$ **do** ▷ Subdivision in vertical fibers $X = x_i$ and $Y = y_j$
14: **for** $j \leftarrow 0$ **to** $N-1$ **do**
15: $Q : Z \longmapsto \sum_t d_{i,j,t} Z^t$
16: $S_Z \leftarrow$ FLOAT_ISOLATE1D(Q, N, 0, $N-1$)
17: **for** $I \in \mathcal{I}$, Add (c_i, c_j, I) to $\mathcal{E}$
18: **end for**
19: **end for**
20: **return** $\mathcal{E}$
21: **end procedure**

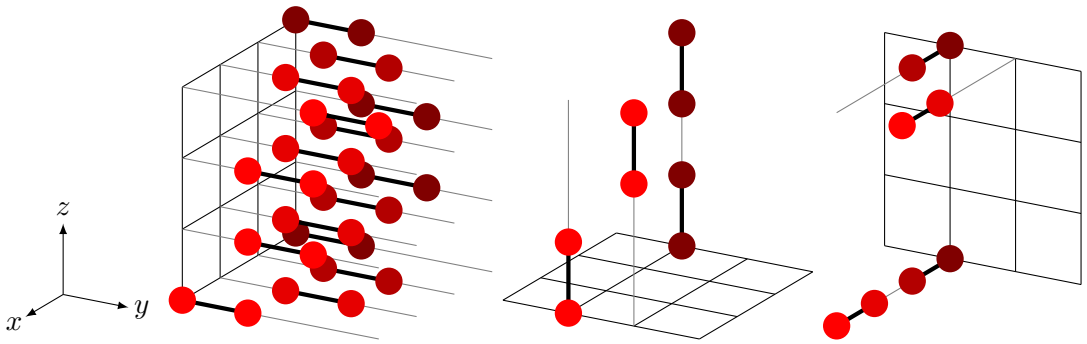


Figure 1.1: Detection of vertices which contribute to the output of the Marching Cubes algorithm with color depending on the coordinate along the x-axis (●●●●) for clarity.

FME in Line 8, so the cost is $O(dN(d^2 + N\log_2(N))$. In Line 16, in each vertical fiber, the subdivision process with Algorithm 2 does $O(T)$ interval evaluations of the partially evaluated univariate polynomial of degree d. Using a classical Horner evaluation, each evaluation is in $O(d)$. The complexity for all the fibers is thus $O(N^2dT)$. Hence, the global complexity is $O((d^2 + N\log_2(N))(d^2 + dN) + N^2dT) = O(d^3N + dN^2(\log_2(N) + T))$. □

1.2 Enclosure from fiber intersection

The marching cubes algorithm constructs a discrete representation of an implicit surface inside voxels which have edges where the function changes sign. By applying Algorithm 9 in each direction, we get that set of edges. In Figure 1.1, this operation is illustrated with an example of output edges in black, the color of the vertices depends on the x-coordinate only for the visual clarity of the figure. We can then deduce the set of voxels which contributes to the final representation. For the same example, we get the subset of voxels represented in Figure 1.2. One can then apply the marching cubes algorithm to this subset instead of the whole grid. We expect this procedure to reduce the complexity, which would otherwise be $O(N^3)$.

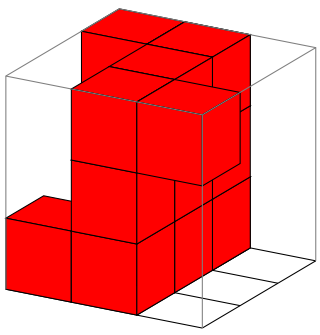


Figure 1.2: Subset of voxels.

2

A voxel enclosing algorithm

In this chapter, we propose a voxel enclosing algorithm for the drawing of surfaces. It can capture features of a surface inside voxels, not only along the fibers. The polynomial which defines the surface is evaluated on intervals for the first two variables and then the algorithm does subdivision along vertical tubes of the form $I_{c_i} \times I_{c_j}$. More specifically, we evaluate the first variable on intervals around the Chebyshev nodes (c_i) using multipoint evaluations and Taylor approximation, then we evaluate the second variable similarly around the nodes (c_j), and finally we do subdivisions along the third variable for a polynomial with interval coefficients.

Let us look at the exact algorithms that we use. In order to enclose the surface, we first partially evaluate the polynomial $P(X, Y, Z)$ in the first two variables using Taylor approximation. To do so, we use the fast evaluation from Algorithm 10. It returns the polynomials $P(I_{c_i}, I_{c_j}, Z)$, where I_{c_i} is the interval around c_i computed by Subroutine 8. For each of them, Algorithm 10 performs the subdivision process, along a tube as illustrated in Figure 2.1, which corresponds to isolating the zeros for the third variable of P. Notice that the evaluation is not aligned with the voxels of the Chebyshev grid. Nonetheless, we choose to light voxels of that grid. So, when we detect a zero in $(I_{c_i}, I_{c_j}, [c_k, c_{k-1}])$, we light the four voxels $([c_{i+1}, c_i], [c_{j+1}, c_j], [c_k, c_{k-1}])$, $([c_{i+1}, c_i], [c_j, c_{j-1}], [c_k, c_{k-1}])$, $([c_i, c_{i-1}], [c_{j+1}, c_j], [c_k, c_{k-1}])$ and $([c_i, c_{i-1}], [c_j, c_{j-1}], [c_k, c_{k-1}])$.

Lemma 2.0.1. *The number of arithmetic operations of Subroutine 9 (*Float_2DEvaluation_TaylorApprox*) is* $O(md^2N + mN^2 \log_2(N))$.

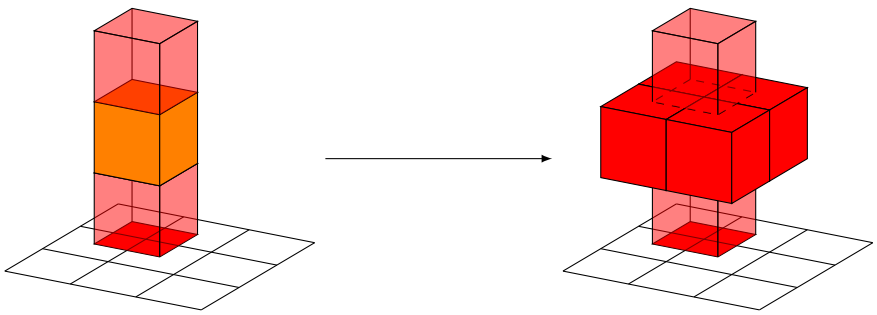


Figure 2.1: Evaluation of $P(I_{c_i}, I_{c_j}, Z)$: detection of a zero in the orange cuboid on the left, lighting of the four voxels it intersects on the right.

Subroutine 9 Bivariate evaluation with Taylor approximation.

Input: A bivariate polynomial $P(X,Y) = \sum_{r,s} a_{r,s} X^r Y^s$ with $\mathbf{a} \in \mathbb{F}^{(d+1)\times(d+1)}$, a Chebyshev grid resolution integer $N > 0$ and the order m of the Taylor approximation

Output: The evaluation on each box $I_{c_i} \times I_{c_j}$

function FLOAT_2DEVALUATION_TAYLORAPPROX($\mathbf{a}$, N, m)
  $\mathbf{b} \in \mathbb{IF}[X]^{N\times(d+1)}$
  for $s \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{s=0}^{d} b_{i,s} Y^t = P(I_{c_i}, Y)$
    $P_s : X \longmapsto \sum_{r=0}^{d} a_{r,s} X^r$
    $T_{0,s}, \ldots, T_{N-1,s} \leftarrow$ FLOAT_TAYLORAPPROXIMATIONS(P_s, N, m)
    $a_{max} \leftarrow \max_{0\le r\le d} |a_{r,s}|$
    for $i \leftarrow 0$ **to** $N-1$ **do**
      $b_{i,s} \leftarrow$ EVALAROUNDCHEBNODE($T_{i,s}$, m, i, a_{max}, N)
    end for
  end for
  $\mathbf{I} \in \mathbb{IF}^{N\times N}$
  for $i \leftarrow 0$ **to** $N-1$ **do** ▷ Processing vertical stripe $X = I_{c_i}$
    $Q_i : Y \longmapsto \sum_{t=0}^{d} b_{i,t} Y^t$
    $U_{i,0}, \ldots, U_{i,N-1} \leftarrow$ FLOAT_TAYLORAPPROXIMATIONS(Q_i, N, m)
    $b_{i,max} \leftarrow \max_{0\le j\le d} |b_{i,j}|$
    for $j \leftarrow 0$ **to** $N-1$ **do** ▷ Evaluation of $P(I_{c_i}, I_{c_j})$
      $I_{i,j} \leftarrow$ EVALAROUNDCHEBNODE($U_{i,j}$, m, j, $b_{i,max}$, N)
    end for
  end for
  return I
end function

Proof. The processing of vertical fibers of Algorithm 6 becomes the processing of vertical stripes in Subroutine 9. However, the operation are unchanged. So, the cost of the second step is still $O(N(md^2 + mN\log_2(N) + d + mN))$. The first step has the same structure, but the loop is not on N stripes corresponding to Chebyshev nodes, but on d partial polynomials. Thus, the cost is $O(d(md^2 + mN\log_2(N) + d + mN))$. So the final complexity is $O(N(md^2 + mN\log_2(N) + d + mN))$ □

Algorithm 10 Fast voxel enclosing with subdivision.

Input: A trivariate polynomial $P(X,Y,Z) = \sum_{r,s,t} a_{r,s,t}X^rY^sZ^t$ with $\mathbf{a} \in \mathbb{F}^{(d+1)\times(d+1)\times(d+1)}$, a Chebyshev grid resolution integer $N > 0$ and the order m of the Taylor approximation
Output: A set of voxels $\mathcal{V}$, enclosing the surface represented by P

1: **procedure** FLOAT_ENCLOSINGVOXELS_TAYLORAPPROX-TAYLORAPPROX-SUBD($\mathbf{a}$, N, m)
2: $\mathbf{b} \in \mathbb{F}^{N\times N\times d}$
3: **for** $t \leftarrow 0$ **to** d **do** ▷ Partial evaluation $\sum_t b_{i,j,t}Z^t = P(I_{c_i}, I_{c_j}, Z)$
4: $b_{i,j,t} \leftarrow$ FLOAT_2DEVALUATION_TAYLORAPPROX($\sum_{r,s,t} a_{r,s,t}X^rY^s$, N, m)
5: **end for**
6: $\mathcal{V} \leftarrow \emptyset$
7: **for** $i \leftarrow 0$ **to** $N-1$ **do**
8: **for** $j \leftarrow 0$ **to** $N-1$ **do** ▷ Isolation for $P(I_{c_i}, I_{c_j}, Z)$
9: $Q : Z \longmapsto \sum_t b_{i,j,t}Z^t$
10: $V \leftarrow$ FLOAT_ISOLATE1D(Q, N, 0, $N-1$)
11: **for all** $I \in V$ **do**
12: Add $([c_{i+1}, c_i], [c_{j+1}, c_j], I)$, $([c_{i+1}, c_i], [c_j, c_{j-1}], I)$, $([c_i, c_{i-1}], [c_{j+1}, c_j], I)$ and $([c_i, c_{i-1}], [c_j, c_{j-1}], I)$ to $\mathcal{V}$
13: **end for**
14: **end for**
15: **end for**
16: **return** $\mathcal{V}$
17: **end procedure**

Theorem 2.0.2. *The number of operations of Algorithm 10 (*FLOAT_ENCLOSINGVOXELS_-TAYLORAPPROX-TAYLORAPPROX-SUBD*) is $O(md^3N + mdN^2\log_2(N) + dN^2T_{max})$, where T_{max} is the size of the largest evaluation tree in the process.*

Proof. The algorithm has two steps, a partial evaluation step and an isolation step. The $(d+1)$ partial polynomials $(\sum_{r,s} a_{r,s,t}X^rY^s)_{t\in[\![0,d]\!]}$ are evaluated using the function 2DTAYLOREVALUATION from Subroutine 10. From Lemma 2.0.3, each call has a complexity of $O(md^2N + mN^2\log_2(N))$, so this partial evaluation cost $O(md^3N + mdN^2\log_2(N))$. Then, for each of the N^2 polynomials of the form $\sum_t b_{i,j,t}Z^t$ the function FLOAT_ISOLATE1D from Algorithm 6 and the corresponding voxels are added to $\mathcal{V}$. For each partial polynomial, this has a complexity of $O(dT)$ from Lemma 2.3.1. So, this step has a complexity of $O(N^2dT_{max})$, where T_{max} is the size of the largest subdivision tree. Therefore, the total complexity is $O(md^3N + mdN^2\log_2(N) + dN^2T_{max})$. □

In the 2D case, Algorithm 7 returns a set of pixels enclosing a curve, then Algorithm 8 colors the pixels in black if we know for sure that they are crossed by the curve or in red if we cannot decide.

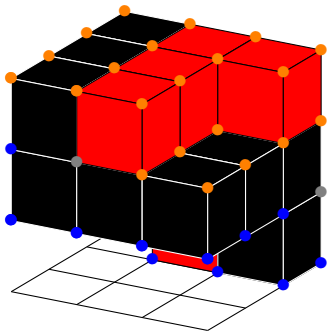


Figure 2.2: A colored voxel drawing. Vertices are colored with respect to the sign of the evaluation at that point: strictly positive (•), strictly negative (•) or the evaluation contains zero (•). A voxel is black when there is at least a vertex of the voxel with a strictly positive evaluation and another with a strictly negative evaluation. A voxel is red when it is part of the enclosure, but is not black. A voxel is white when it is not part of the enclosure.

Likewise, Algorithm 10 returns a set of voxels $\mathcal{V}$ which encloses a surface. Then, Algorithm 11 determines black and red voxels, $\mathcal{V}_b$ and $\mathcal{V}_r$, among these ones. For that purpose, it evaluates partial polynomials on each vertex of the voxel. A voxel is black when there is at least a vertex of the voxel with a strictly positive evaluation and another with a strictly negative evaluation. A voxel is red when it is part of the enclosure, but is not black. A voxel is white when it is not part of the enclosure.

Lemma 2.0.3. *The number of arithmetic operations of Subroutine 10 (*Float_2DEvaluation_PartialEval*) is* $O(d^3N + dN^2\log_2(N))$.

Proof. The main loop of Algorithm 10, from Line 3 to Line 12, contains two inner loops: there is one inner loop for the evaluation in each variable. The first variable is evaluated in Lines 4 to 7. We apply the $\square$FME on a polynomial of degree d for N points, so this operation has a cost $O(d^2 + N\log_2(N))$. Then, $b_{-1,s,t}$ and $b_{N,s,t}$ are computed as sums of $d+1$ elements with a cost $O(d)$. The loop being repeated $d+1$ times, it contributes to a complexity of $O(d(d^2+N\log_2(N))$. In the next inner loop, the operations are similar but the loop is repeated $N+2$ times, therefore it contributes to a complexity of $O(N(d^2 + n\log_2(N))$. The outer loop is repeated $d+1$ times, so the whole loop has a complexity of $O(dN(d^2 + N\log_2(N))$. The rest of the algorithm does no computation, it simply constructs polynomials from the computed $(d_{i,j,t})_{i,j\in[\![0,N-1]\!],t\in[\![0,d]\!]}$. $\square$

Lemma 2.0.4. *The number of arithmetic operations of Algorithm 11 is* $O(d^3N + dN^2\log_2(N) + dS)$*, when it is called on* Float_ColoredVoxels*(*P*,* $\mathcal{V}$*,* N*) with* P *a trivariate polynomaial of partial degree* d *and* $\mathcal{V}$ *a set of voxels of size* S.

Proof. The first step is to compute the partial polynomials using Subroutine 10, whose complexity is given by Lemma 2.0.3. Then for each voxel of $\mathcal{V}$ a test is done using with $P_{i,j}$, $P_{i,j-1}$, $P_{i-1,j}$ and $P_{i-1,j-1}$ of degree d. Eight polynomial evaluations of degree d are performed in Line 6, two for each polynomial. Each one has a complexity of $O(d)$. In Lines 7 and 8, the sign of the eight values are tested twice with a constant cost. The final boolean evaluation has also a constant cost. So, the complexity is $O(d)$. Thus, the cost of the decision step is $O(Sd)$. $\square$

Subroutine 10 Bivariate evaluation at Chebyshev nodes.

Input: A trivariate polynomial $P(X,Y,Z) = \sum_{r,s,t} a_{r,s,t} X^r Y^s Z^t$ with $\mathbf{a} \in \mathbb{F}^{(d+1)^3}$ and a Chebyshev grid resolution integer $N > 0$
Output: The partial evaluations $P(c_i, c_j, Z)$ on each point (c_i, c_j)

function FLOAT_2DEVALUATION_PARTIALEVAL(P, N)
  $\mathbf{b} \in \mathbb{IF}^{N\times(d+1)\times(d+1)}$, $\mathbf{d} \in \mathbb{IF}^{N\times N\times(d+1)}$
  for $t \leftarrow 0$ **to** d **do**
    for $s \leftarrow 0$ **to** d **do** ▷ Evaluation along the first variable
      $P_s : X \longmapsto \sum_{r=0}^{d} a_{r,s,t} X^r$
      $b_{0,s,t}, \ldots, b_{N-1,s,t} \leftarrow$ FLOAT_FME($(a_{0,s,t}, \ldots, a_{d,s,t})$,$N$)
    end for
    for $i \leftarrow 0$ **to** $N-1$ **do** ▷ Evaluation along the second variable
      $Q_i : Y \longmapsto \sum_{t=0}^{d} b_{i,s,t} Y^s$
      $d_{i,0,t}, \ldots, d_{i,N-1,t} \leftarrow$ FLOAT_FME($(b_{i,0,t}, \ldots, b_{i,d,t})$,$N$)
    end for
  end for
  for $i \leftarrow 0$ **to** $N-1$ **do**
    for $j \leftarrow 0$ **to** $N-1$ **do**
      $U_{i,j} : Z \longmapsto \sum_{t=0}^{d} d_{i,j,t} Z^t$
    end for
  end for
  return $(U_{i,j})_{i,j\in[\![0,N-1]\!]}$
end function

Algorithm 11 Fast colored voxel drawing.

Input: A trivariate polynomial $P(X,Y,Z) = \sum_{r,s,t} a_{r,s,t} X^r Y^s Z^t$ with $\mathbf{a} \in \mathbb{F}^{(d+1)\times(d+1)\times(d+1)}$, a set of voxels $\mathcal{V}$ and a Chebyshev grid resolution integer $N > 0$
Output: A set of black voxels $\mathcal{V}_b$ and a set of red voxels $\mathcal{V}_r$

procedure FLOAT_COLOREDVOXELS(P, $\mathcal{V}$, N)
  $P \in \mathbb{IF}[X]^{N\times N}$
  $P_{i,j} \leftarrow$ FLOAT_2DEVALUATION_PARTIALEVAL(P, N)
  $\mathcal{V}_b, \mathcal{V}_r \leftarrow \emptyset, \emptyset$
  for all $[c_i, c_{i-1}] \times [c_j, c_{j-1}] \times [z_1, z_2] \in \mathcal{V}$ **do** ▷ Decide if a voxel is black or red
    $V \leftarrow \cup_{k\in\{i,i-1\}, l\in\{j,j-1\}} \{\square P_{k,l}(z_1), \square P_{k,l}(z_2)\}$
    $b_+ \leftarrow \exists\, v \in V,\ v > 0$
    $b_- \leftarrow \exists\, v \in V,\ v < 0$
    if b_+ and b_- **then**
      Add $[c_i, c_{i-1}] \times [c_j, c_{j-1}] \times [z_1, z_2]$ to $\mathcal{V}_b$
    else
      Add $[c_i, c_{i-1}] \times [c_j, c_{j-1}] \times [z_1, z_2]$ to $\mathcal{V}_r$
    end if
  end for
  return $\mathcal{V}_b, \mathcal{V}_r$
end procedure

In practice, we apply Algorithm 11 to an enclosure of the surface which has already been computed as the output of Algorithm 10. So, using the notations of Theorem 2.0.2 the output $\mathcal{V}$ is of size $S = O(dT_{max})$ (where T_{max} is defined in Note of Section 1.2.3). Thus, the cost of Algorithm 10 dominates the cost of Algorithm 11, hence the latter gives the complexity of the whole process.

Corollary 6. *When Algorithm 10 is called on* FLOAT_ENCLOSINGVOXELS_TAYLORAPPROX-TAYLORAPPROX-SUBD*(P, N, m) to get $\mathcal{V}$ and then Algorithm 11 is called on* FLOAT_COLOREDVOXELS*(P, $\mathcal{P}$), the number of arithmetic operations of the whole process is $O(md^3 + mdN\log_2(N) + dNT_{max})$, where P is a trivariate polynomial of partial degree d and where T_{max} is the size of the largest evaluation tree in the process.*

Conclusion

Contributions

We have presented new algorithms for visualizing implicit two- and three-dimensional algebraic varieties, that is plane curves and surfaces, on a fixed resolution grid. We have specifically tackled the visualization of high degree ($d > 20$) varieties at high resolutions ($N > 1,000$). Indeed, when the degree of the polynomial which defines a variety increases, common approaches happen to struggle to output a drawing in a reasonable time, because of the high cost of the evaluation. Therefore, we have introduced a multipoint evaluation method from computer algebra, namely a fast DCT computation. That leads to a grid aligned on Chebyshev nodes. In some algorithms, we have combined it to the classical subdivision approach for root finding.

Let us remind the main algorithms that we have presented and give them abbreviated names.

Algorithm EMS
Float_Enclosing**E**dges_**M**ultiEval-**S**ubd

Algorithm EMT
Float_Enclosing**E**dges_**M**ultiEval-**T**aylorApprox

Algorithm PTS
Float_Enclosing**P**ixels_**T**aylorApprox-**S**ubd

Algorithm EMMS
Float_Enclosing**E**dges_**M**ultiEval-**M**ultiEval-**S**ubd

Algorithm VTTS
Float_Enclosing**V**oxels_**T**aylorApprox-**T**aylorApprox-**S**ubd

There are three ways to classify these algorithms: algorithms for plane curves and algorithms for surfaces, algorithms entirely relying on multipoint evaluation and algorithms relying on multipoint evaluation and subdivision, or algorithms on edges of the underlying grid and algorithms on cells (pixels or voxels) of the underlying grid. That is summarized in Table 1.

We have analysed the theoretical complexity and have experimentally verified that these methods are indeed fast in two dimensions. They even prove to be competitive against the fastest state-of-the-art software, while providing guarantees. They return a non-trivial superset of the edges or the cells intersected by the variety that one wants to draw. We pay attention to the numerical uncertainty with error analysis and interval arithmetic in order to minimize the size of this superset. We show that evaluation around nodes is possible thanks to Taylor approximation with multipoint evaluation. The use of multipoint evaluation followed by a subdivision approach

Table 1: Summary of our main algorithms.

	MultiEval	MultiEval + Subd	
edge	EMT	EMS	2D
	-	EMMS	3D
cell	-	PTS	2D
	-	VTTS	3D

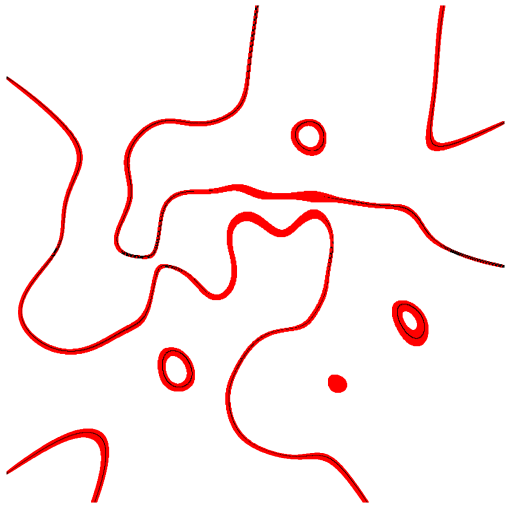


Figure 1: Drawing of *random_30_kss* with Algorithm PTS for $N = 1,024$.

for the last dimension proves to be the most efficient. That is what has been chosen for most of our algorithms.

Algorithms returning cells give an enclosing of the variety, while algorithms returning edges are unable to detect connected components of the variety which lie inside a cell. Nonetheless, from the outputs we are able to identify a subset of the cells intersected by the variety. We light those cells in black, the cells which are in the superset but not in the subset in red, and the rest are white (see Figure 1). Using edge enclosing algorithms for the drawing may be reasonable when one knows that connected components are not smaller than the cell or if one tolerates that kind of error.

Perspectives and open questions

Ideally, one would like a correct drawing, a drawing composed exactly of the pixels intersected by the curve. Algorithms on edges are unable to satisfy such a requirement. However, that could be possible for algorithms on cells if they could compute results with arbitrary precision. Let us introduce the intuitive notion of "thickness" of a variety, that is roughly the number of adjacent pixels with the same color in the normal direction to the curve, locally. It would be natural to ask for a drawing whose thickness for red cells does not exceed its thickness for black cells.

We could give additional information about the drawing by using criteria like those of Snyder or Plantinga and Vegter. For enclosing edge algorithms, one could use root isolation methods other

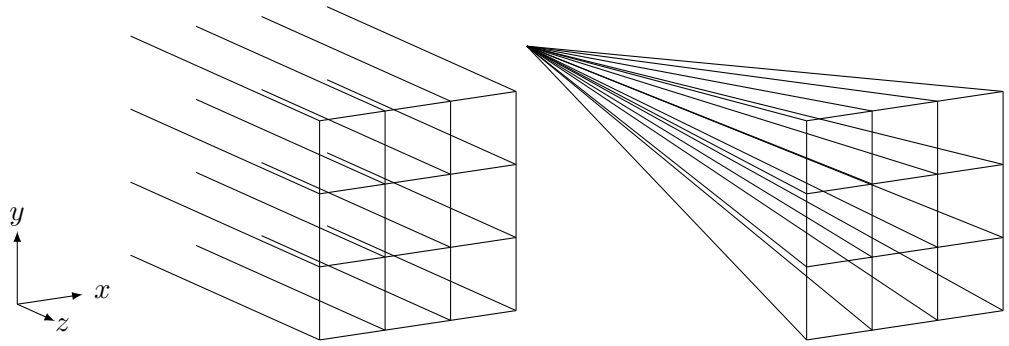


Figure 2: Illustration of orthogonal (left) and perspective (right) projection.

than a subdivision process on a univariate polynomial. Using the polynomial and its derivative, one could know if an edge is intersected exactly once. If that is known early enough in the subdivision process, that could allow the process to converge to the intersected edge even faster.

We have shown that our ideas can be used to improve the speed of the marching cubes algorithm and all the methods based on that seminal algorithm. For that application, we have a trivariate polynomial associated to the Cartesian coordinate system. We partially evaluate the polynomial in the first two variables, then evaluate and subdivide in the last dimension along vertical fibers or tubes. All those values could be projected and used for another task for instance. That would correspond to an orthogonal projection. One could also use this idea for ray tracing. In that case, we would do a perspective projection, also known as central projection. The two cases are illustrated by Figure 2. For a trivariate polynomial $P(X, Y, Z)$, a ray correspond to the univariate polynomial $R_{\alpha,\beta}(T) = P(\alpha T, \beta T, T)$. With a careful reordering of the term, one can get $R_{\alpha,\beta}(T) = Q(\alpha, \beta, T)$ by performing fast multipoint evaluations on the first two variables of Q.

Bibliography

[AR91] Bradley K. Alpert and Vladimir Rokhlin. A fast algorithm for the evaluation of legendre expansions. SIAM Journal on Scientific and Statistical Computing, 12(1):158–179, 1991.

[BES05] Alin Bostan and Éric Schost. Polynomial evaluation and interpolation on special sets of points. Journal of Complexity, 21(4):420–446, 2005. Festschrift for the 70th Birthday of Arnold Schonhage.

[BJM+20] Nicolas Brisebarre, Mioara Joldes, Jean-Michel Muller, Ana-Maria Nanes, and Joris Picot. Error analysis of some operations involved in the cooley-tukey fast fourier transform. ACM Trans. Math. Softw., 46(2):11:1–11:27, 2020.

[BK78] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. J. ACM, 25(4):581–595, oct 1978.

[BKY09] M Burr, F Krahmer, and Chee Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. In Electronic colloquium on computational complexity (ECCC), TR09 (136), 2009.

[BP04] Roberto Barrio and Juan Manuel Peña. Basis conversions among univariate polynomial representations. Comptes Rendus Mathematique, 339(4):293–298, 2004.

[BSS08] Alin Bostan, Bruno Salvy, and Éric Schost. Power series composition and change of basis. In Proceedings of the Twenty-First International Symposium on Symbolic and Algebraic Computation, ISSAC '08, page 269–276, New York, NY, USA, 2008. Association for Computing Machinery.

[BT06] Jean-Daniel Boissonnat and Monique Teillaud, editors. Effective Computational Geometry for Curves and Surfaces. Springer-Verlag, Mathematics and Visualization, 2006.

[CLO07] D. Cox, J. Little, and D. O'Shea. Ideals, Varieties, and Algorithms. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 3rd edition, 2007.

[CLP+10] Jinsan Cheng, Sylvain Lazard, Luis Peñaranda, Marc Pouget, Fabrice Rouillier, and Elias Tsigaridas. On the topology of real algebraic plane curves. Mathematics in Computer Science, 4(1):113–137, Nov 2010.

[Col75] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In Automata Theory and Formal Languages: 2nd GI Conference Kaiserslautern, May 20–23, 1975, pages 134–183. Springer, 1975.

[DJR97] J.R. Driscoll, D.M. Healy Jr., and D.N. Rockmore. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs, 1997.

[EK95] Alan Edelman and Eric Kostlan. How many zeros of a random polynomial are real? Bulletin of the American Mathematical Society, 32(1):1–37, 1995.

[Gan26] Solomon Gandz. On the origin of the term "root. The American Mathematical Monthly, 33(5):261–265, 1926.

[Gan28] Solomon Gandz. On the origin of the term "root." second article. The American Mathematical Monthly, 35(2):67–75, 1928.

[Ger00] Jürgen Gerhard. Modular algorithms for polynomial basis conversion and greatest factorial factorization, 2000.

[GVJ+09] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill, and C. Galbraith. Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms. Springer London, 2009.

[Hig02] Nicholas J. Higham. Accuracy and Stability of Numerical Algorithms. Society for Industrial and Applied Mathematics, second edition, 2002.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020.

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n\log n)$. Annals of Mathematics, 193(2):563 – 617, 2021.

[IEE19] Ieee standard for floating-point arithmetic. IEEE Std 754-2019 (Revision of IEEE 754-2008), pages 1–84, 2019.

[Imp] ImplicitEquations: Julia package to facilitate graphing of implicit equations and inequalities. `https://github.com/jverzani/ImplicitEquations.jl`.

[Joh13] F. Johansson. Arb: a C library for ball arithmetic. ACM Communications in Computer Algebra, 47(4):166–169, 2013.

[Kei09] Jens Keiner. Computing with expansions in gegenbauer polynomials. SIAM Journal on Scientific Computing, 31(3):2151–2171, 2009.

[KLVS98] Fred T. Krogh, Charles L. Lawson, and W. Van Snyder. MATH77 and mathc90, Release 6.0. JPL Computational Mathematics group, May 1998.

[Kos02] Eric Kostlan. On the expected number of real roots of a system of random polynomial equations. In Foundations of computational mathematics, pages 149–188. World Scientific, 2002.

[KS16] Alexander Kobel and Michael Sagraloff. Fast approximate polynomial multipoint evaluation and applications, 2016. `arXiv:https://arxiv.org/abs/1304.8069v2 [cs.NA]`.

[KZ14] Zakhar Kabluchko and Dmitry Zaporozhets. Asymptotic distribution of complex zeros of random analytic functions. The Annals of Probability, 42(4):1374–1395, 2014.

[Lab09] O. Labs. A list of challenges for real algebraic plane curve visualization software. In Nonlinear Computational Geometry, volume IMA 151, pages 137–164. Springer, 2009.

[Lat12] Jean-Claude Latombe. Robot motion planning, volume 124. Springer Science & Business Media, 2012.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput. Graph., 21:163–169, August 1987.

[LLVT03] Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. Journal of Graphics Tools, 8(2):1–15, 2003.

[Mak80] J. Makhoul. A fast cosine transform in one and two dimensions. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(1):27–34, 1980.

[MH03] J.C. Mason and D.C. Handscomb. Chebyshev Polynomials. CRC Press, 2003.

[MKC09] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. Introduction to interval analysis. Siam, 2009.

[Neu90] Arnold Neumaier. Interval methods for systems of equations. Cambridge University Press, 1990.

[NV22] Oanh Nguyen and Van Vu. Roots of random functions: A framework for local universality. American Journal of Mathematics, 144(1):1–74, 2022.

[Pan98] V.Y. Pan. New fast algorithms for polynomial interpolation and evaluation on the chebyshev node set. Computers & Mathematics with Applications, 35(3):125–129, 1998.

[PM02] Nicholas M Patrikalakis and Takashi Maekawa. Shape interrogation for computer aided design and manufacturing, volume 15. Springer, 2002.

[PS85] Franco P Preparata and Michael I Shamos. Computational geometry: an introduction. Springer Science & Business Media, 1985.

[PST98] Daniel Potts, Gabriele Steifdl, and Manfred Tasche. Fast algorithms for discrete polynomial transforms. Mathematics of Computation, 67, 04 1998.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical Recipes 3rd Edition: The Art of Scientific Computing. Cambridge University Press, USA, 3 edition, 2007.

[PV04] Simon Plantinga and Gert Vegter. Isotopic approximation of implicit curves and surfaces. In Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, SGP '04, pages 245–254, New York, NY, USA, 2004. ACM.

[Roy90] Marie-Françoise Roy. Computation of the topology of a real curve. In Hayat-Legrand Claude and Sergeraert Francis, editors, Algorithmique, topologie et

géométrie algébriques - Sévilla,1987, Toulouse 1988, number 192 in Astérisque. Société mathématique de France, 1990.

[Sch82] Arnold Schönhage. Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients. In Jacques Calmet, editor, Computer Algebra, pages 3–15, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[Sny92] John M. Snyder. Interval analysis for computer graphics. SIGGRAPH Comput. Graph., 26(2):121–130, July 1992.

[SS93] Michael Shub and Steve Smale. Complexity of bezout's theorem ii volumes and probabilities. In Frédéric Eyssette and André Galligo, editors, Computational Algebraic Geometry, pages 267–285, Boston, MA, 1993. Birkhäuser Boston.

[Tup01] Jeff Tupper. Reliable two-dimensional graphing methods for mathematical formulae with two free variables. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, page 77–86, New York, NY, USA, 2001. Association for Computing Machinery.

[vdH02] Joris van der Hoeven. Relax, but don't be too lazy. Journal of Symbolic Computation, 34(6):479–542, 2002.

[vdH08] Joris van der Hoeven. Fast composition of numeric power series. Technical Report 2008-09, Université Paris-Sud, Orsay, France, 2008.

[vdH09] Joris van der Hoeven. Ball arithmetic. 2009.

[vdWSN+14] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. PeerJ, 2:e453, 6 2014.

[vzGG97] Joachim von zur Gathen and Jürgen Gerhard. Fast algorithms for taylor shifts and certain difference equations. In Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, ISSAC '97, page 40–47, New York, NY, USA, 1997. Association for Computing Machinery.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. Modern Computer Algebra. Cambridge University Press, 3 edition, 2013.

[YX98] Yong-Ming Li and Xiao-Ying Zhang. Basis conversion among bézier, tchebyshev and legendre. Computer Aided Geometric Design, 15(6):637–642, 1998.

[YY09] Chee K. Yap and Jihun Yu. Foundations of exact rounding. In Sandip Das and Ryuhei Uehara, editors, WALCOM: Algorithms and Computation, pages 15–31, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

Résumé en français

1 Contexte

Les représentations visuelles peuvent servir différents objectifs. Elles peuvent être artistiques, fonctionnelles ou même les deux. Nous nous intéressons à l'aspect fonctionnel. Plus précisément, nous nous concentrons sur la visualisation scientifique, c'est-à-dire l'illustration graphique de données afin de les présenter de manière condensée ou même d'en obtenir un aperçu. Ces techniques sont utiles aussi bien pour les scientifiques que pour les ingénieurs. Les applications des méthodes fournies par la visualisation scientifique sont nombreuses. Par exemple, la conception assistée par ordinateur est utilisée dans l'animation et les jeux vidéo, mais aussi dans la modélisation pour les simulations, la conception de mécanismes et la théorie du contrôle.

La visualisation scientifique concerne des données traitées par des ordinateurs et interprétées par des personnes. Les données elles-mêmes peuvent être multidimensionnelles. Les représentations produites quant à elles sont constituées d'objets géométriques au plus tridimensionnels, car c'est ce que nous, humains, sommes capables d'appréhender. Néanmoins, la visualisation de données permet de véhiculer davantage de dimensions, par le biais des couleurs par exemple. Dans ce contexte, nous proposons des méthodes de visualisation de courbes et de surfaces planes.

Les cartes topographiques sont des exemples de représentations bidimensionnelles dans lesquelles la couleur véhicule une troisième dimension. Par exemple, sur la Figure 1 une échelle relie des couleurs différentes à des niveaux d'élévation différents. La limite entre deux couleurs successives est un ensemble de lignes dont l'altitude est identique en chaque point. Elles sont appelées *courbes de niveau* ou *isolignes* et définies comme l'ensemble des solutions d'une équation de la forme $f(x, y) = h$ où x et y sont les coordonnées dans le plan, la fonction f renvoie l'élévation en un point donné, c'est-à-dire la hauteur au-dessus ou en-dessous d'un point de référence fixe, et h est une hauteur fixe. La généralisation des courbes de niveau à un nombre quelconque de variables est appelée *lignes de niveau*. L'équation précédente peut être réécrite comme $F(x, y) = f(x, y) - h = 0$ où F est une fonction de deux variables. Les courbes définies par une équation de la forme $F(x, y) = 0$ sont appelées *courbes implicites*. L'adjectif « implicite » signifie qu'il ne s'agit ni d'une équation en termes de x, ni en termes de y. Lorsque F est une fonction polynomiale, la

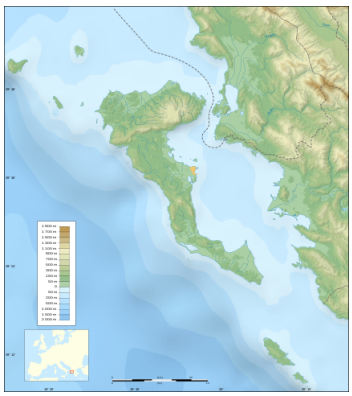


Figure 1: Carte topographique de la région de l'île de Corfu en Grèce par Eric Gaba (CC BY-SA 3.0).[8]

[8]Source: `https://commons.wikimedia.org/wiki/File:Corfu_topographic_map-blank.svg`.

courbe est appelée *courbe algébrique*. Plus généralement, les solutions des systèmes d'équations polynomiales sont appelées *variétés algébriques*.

Les logiciels de conception assistée par ordinateur manipulent des objets représentés par des courbes et des surfaces paramétriques, ainsi que des courbes et des surfaces implicites. Ces dernières sont pratiques pour la visualisation de champs scalaires (altitude, densité...). C'est notamment l'approche adoptée pour la reconstruction de surface en imagerie médicale.

Une autre façon de visualiser une surface implicite est d'utiliser le lancer de rayons. La fonction qui définit la surface implicite est alors une fonction de distance signée. Le lancer de rayons construit une vue des objets, qui peut se réaliser en temps réel grâce aux progrès matériels récents. Ainsi, en changeant le point de vue on peut se déplacer autour des objets comme dans les autres méthodes. En outre, le lancer de rayons modélise le transport de la lumière et peut donc intrinsèquement simuler des effets tels que la réflexion, la réfraction ou les ombres.

2 État de l'art

Nous ne prétendons pas être exhaustifs dans la description des approches présentées ici : les algorithmes que nous présentons sont des algorithmes de maillage. Dans le cadre bidimensionnel, un maillage de courbe est un ensemble de lignes polygonales qui l'approximent. Dans le cadre tridimensionnel, il s'agit d'un ensemble de surfaces polygonales approximant une surface. Les principales idées proviennent soit de l'algorithme *marching cubes*, soit des méthodes de continuation de courbe [GVJ+09, Partie III]. En bref, ces dernières localisent un point sur la courbe et tentent de trouver le reste de la courbe par continuité. Seule la première famille est présentée dans les sections suivantes.

Nous présentons d'abord l'algorithme *marching cubes* dans la Section 2.1. Ensuite, nous présentons des méthodes qui préservent la topologie des courbes dans les Sections 2.2 et 2.3. Succinctement, cela signifie que ces dernières méthodes préservent les boucles et les auto-intersections des courbes.

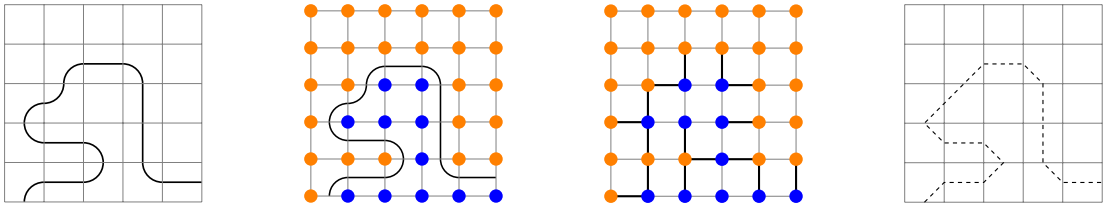
2.1 Marching cubes

L'application originale de l'algorithme *marching cubes* est la construction d'une visualisation de données provenant de la tomographie ou de l'imagerie par résonance magnétique [LC87]. Dans le contexte médical, il est raisonnable de travailler avec des fonctions continues. Regardons d'abord la version bidimensionnelle de l'algorithme qui est plus simple : *marching squares*.

Marching squares

Nous aimerions visualiser la courbe $\mathcal{C}$ définie comme l'ensemble des points (x, y) qui vérifient $f(x, y) = 0$. Au lieu de cela, nous échantillonnons f sur le domaine qui nous intéresse et essayons de visualiser $\mathcal{C}'$ définie comme l'ensemble des points qui vérifient $\phi(x, y)$ où ϕ est un interpolateur de f. Nous construisons en fait une sortie linéaire par morceaux, qui est dite topologiquement cohérente. En d'autres termes, la courbe renvoyée ne présente aucune discontinuité.

Décrivons le principe de l'algorithme. Marching squares reçoit une fonction qui définit la courbe implicite que nous voulons représenter et la fenêtre dans laquelle elle doit être visualisée (voir Figure 2a). La fenêtre est donnée par une grille. La fonction est évaluée à chaque sommet et le signe de l'évaluation est stocké (voir Figure 2b). La fonction est supposée continue. Par



(a) Courbe cible. (b) Évaluation. (c) Arêtes intersectées. (d) Approximation.

Figure 2: Illustration des étapes de l'algorithme marching squares; les points orange (●) représentent des évaluations positives, les points bleus (●) représentent des évaluations négatives.

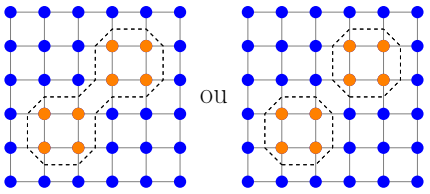


Figure 3: Exemple de la conséquence de l'ambiguïté des points-selles.

conséquent, si deux sommets adjacents ont des signes différents, cela signifie que la fonction s'annule le long de l'arête qui les relie. Donc, la courbe cible traverse cette arête. Pour chaque carré de la grille, il est ainsi possible d'identifier les arêtes traversées par la courbe cible (voir Figure 2c). Chaque arête d'un carré est soit traversée, soit non traversée, ce qui donne $2^4 = 16$ combinaisons différentes. Elles peuvent être réduites à quatre cas, à rotation et changement de signe près. L'algorithme fournit une table de correspondance, qui permet de dessiner à l'intérieur du carré une approximation linéaire de la courbe entre les arêtes traversées pour chaque cas. Par construction, l'algorithme renvoie une représentation linéaire par morceaux de la courbe (voir Figure 2d).

Toute la procédure est locale à l'intérieur de chaque carré. Cependant, elle permet d'obtenir une approximation globale. Cela signifie que la procédure peut être parallélisée.

Marching cubes

Naturellement, les idées présentées à propos de marching squares s'étendent au cas tridimensionnel. L'algorithme de marching cubes reçoit une fonction qui définit une surface implicite cible. La fenêtre et la grille sont tridimensionnelles. La fonction est évaluée à chaque sommet. Pour chaque cube, il existe $2^8 = 256$ combinaisons différentes. Elles peuvent être réduites à 15 cas, à rotation, réflexion et changement de signe près.

2.2 Méthodes avec grille adaptative

La grille fixe de marching squares, respectivement de marching cubes, n'est pas suffisante pour capturer tous les détails de la courbe, respectivement de la surface. Il peut y avoir des caractéristiques plus petites que la résolution. C'est pourquoi certaines méthodes utilisent un maillage adaptatif, qui est localement affiné afin de capturer les détails de la courbe, comme l'illustre la Figure 4. Dans cette section, nous présentons les versions bidimensionnelles des algorithmes, car les algorithmes tridimensionnels en découlent directement.

(a) Une grille de 2×2.

(b) La même grille avec des raffinements locaux.

Figure 4: Grille avant et après raffinement.

(a) Identification des liaisons.

(b) Approximation linéaire par morceaux.

Figure 5: Construction de l'approximation pour une courbe globalement paramétrable dans une cellule donnée.

Algorithme de Snyder

Snyder introduit l'utilisation de l'arithmétique des intervalles dans le domaine de la visualisation [Sny92]. Il propose un algorithme d'approximation des courbes implicites. Son algorithme utilise un critère de paramétrisation globale qui permet de subdiviser les cellules jusqu'à ce qu'aucun détail de la courbe ne puisse être perdu, afin d'obtenir une exactitude topologique. En s'appuyant sur ce critère, l'algorithme traite les courbes implicites multivariées.

Une courbe $\mathcal{C} = \{(x, y) \in [a, b] \times [c, d] \,|\, f(x, y) = 0\}$ est *globalement paramétrable* en x, si pour chaque valeur x l'équation $f(x, y) = 0$ a au plus une solution y dans la cellule $[a, b] \times [c, d]$.

La grille est affinée en utilisant l'arithmétique des intervalles jusqu'à ce que la courbe soit globalement paramétrable à l'intérieur de chaque cellule de la grille, autrement dit elle peut être écrite comme la solution d'une équation de la forme $y = f(x)$ ou $x = g(y)$. Sans perte de généralité, examinons une courbe globalement paramétrable en x dans une cellule (voir Figure 5). Tout d'abord, on localise les intersections (I_i) de la courbe avec les bords de la cellule. En déterminant le signe le long d'une droite séparant deux intersections consécutives, on peut savoir si elles doivent être reliées ou non. Cette étape est également réalisée avec l'arithmétique des intervalles. Ainsi, s'il n'est pas possible de décider s'il y a ou non un changement de signe, la cellule est subdivisée et le processus est répété.

Algorithme de Plantinga et Vegter

Plantinga et Vegter utilisent un critère sur le gradient pour déterminer si une cellule est suffisamment raffinée pour produire un maillage correct avec l'information du signe de l'évaluation

(a) Approximation isotope.

(b) Violation de la petite variation normale.

Figure 6: Illustration du critère de la petite variation normale, de [BT06, Section 5.2.4].

aux sommets [PV04]. Ils utilisent la condition de petite variation normale :

$$\langle \nabla f(p_1), \nabla f(p_2) \rangle \geq 0, \forall p_1, p_2 \in [a, b] \times [c, d].$$

Cette condition contraint l'angle entre les pairs de vecteurs obtenus par évaluation du gradient de f dans le carré $[a, b] \times [c, d]$. Cela se traduit par une condition sur les vecteurs normaux de la courbe. Comme l'illustre la Figure 6, elle garantit que la courbe ne va pas très loin hors de la cellule. Cela permet au segment approximant d'être isotope et géométriquement proche. Tous les calculs s'appuient sur l'arithmétique des intervalles pour assurer la correction.

Grâce à la condition de petite variation normale, l'algorithme de Plantinga et Vegter n'a pas besoin d'une correction topologique dans chaque cellule pour fournir une correction topologique globale, contrairement à l'algorithme de Snyder.

2.3 Méthodes algébriques

Certains algorithmes produisent une approximation topologiquement correcte des courbes en utilisant des méthodes algébriques. Ils exigent seulement que la courbe soit algébrique, sans supposer qu'elle soit lisse. Ils peuvent donc traiter des courbes singulières. Ces algorithmes décomposent le plan en bandes verticales qui contiennent des portions disjointes de la courbe et en lignes verticales qui contiennent un nombre fini de points de la courbe. Cette méthode utilise la décomposition algébrique cylindrique (on utilise l'acronyme anglais CAD), introduite à l'origine par Collins [Col75]. Les bandes verticales contiennent les parties régulières de la courbe et les lignes verticales contiennent les points critiques, c'est-à-dire que pour une courbe définie par f ces points (x, y) vérifient $f(x, y) = 0$ et $\frac{\partial f}{\partial y}(x, y) = 0$.

Example 2.2.1. Pour la courbe définie par $y^2 - (x^3 + x^2) = 0$ dans la Figure 7a, les droites verticales sont definies par $x = -1$ et $x = 0$ et les bandes sont les domaines ouverts $]-\infty, -1[\times \mathbb{R}$, $]-1, 0[\times \mathbb{R}$ et $]0, +\infty[\times \mathbb{R}$.

À partir de cette décomposition, on peut construire une représentation linéaire par morceaux topologiquement équivalente (voir Figure 7b).

Avec la CAD, on retrouve la nature lisse par morceaux de toute courbe algébrique. Ainsi, toute méthode pertinente peut être utilisée pour obtenir de meilleures approximations géométriques de les portions lisses. Le défi numérique consiste à relier les domaines lisses aux singularités.

(a) c est un point critique et s est un point singulier. (b) Ligne polygonale topologiquement équivalente.

Figure 7: Décomposition de la courbe définie par $y^2 - (x^3 + x^2) = 0$.

3 Motivations

3.1 De la recherche de racines au dessin de surfaces

Représenter des courbes et des surfaces algébriques implique de pouvoir trouver les solutions des équations polynomiales qui les définissent. Examinons donc brièvement ce lien entre algèbre et géométrie, ainsi que quelques outils utiles.

L'étude des fonctions polynomiales à une variable a conduit au développement de méthodes pour trouver leurs racines. L'idée de « racine » peut être étendue aux zéros des fonctions à plusieurs variables. Les systèmes d'équations polynomiales à plusieurs variables sont au centre de la géométrie algébrique. Par exemple, les zéros d'une fonction polynomiale à deux variables P se trouvent sur une courbe plane $\mathcal{C} = \{(x, y) \in \mathbb{R}^2 \,|\, P(x, y) = 0\}$ et les zéros d'une fonction polynomiale trivariée Q se trouvent sur une surface $\mathcal{S} = \{(x, y, z) \in \mathbb{R}^3 \,|\, Q(x, y, z) = 0\}$.

Nous nous intéressons aux racines réelles des polynômes. La méthode standard pour trouver les racines consiste à les isoler (calculer des intervalles disjoints qui contiennent exactement une racine chacun, mais dont l'union les contient toutes) et ensuite améliorer la précision des résultats. Certains outils comme le théorème de Sturm, la règle des signes de Descartes, le théorème de Budan et le théorème de Vincent utilisent la variation de signe d'une séquence donnée afin d'isoler les racines. L'arithmétique des intervalles, qui a été développée à l'origine pour fournir des garanties sur les résultats des calculs, peut également être utilisée dans des techniques fiables d'isolation et d'approximation des racines.

3.2 Courbes et surfaces de haut degré

La visualisation scientifique permet à ses utilisateurs de développer leur intuition et de comprendre leurs données et ce qu'elles modélisent. En théorie du contrôle, la conception assistée par ordinateur est un outil permettant d'optimiser et de régler les paramètres d'un système.

Prenons un exemple en robotique, afin de voir que l'on peut avoir à traiter des courbes algébriques de degrés élevés [CLO07, Chapter 6]. Les robots sont caractérisés par les « positions » qu'ils peuvent atteindre. La robotique doit répondre à deux questions principales : « Quelles positions un robot peut-il atteindre? » et « Comment un robot peut-il atteindre une position donnée? » [Lat12]. Ces « positions » dépendent de la structure du robot lui-même. Comme la paramétrisation du problème peut introduire des fonctions polynomiales multivariées, des variétés algébriques peuvent intervenir. On peut avoir besoin de calculer des intersections d'hypersurfaces ou des projections de courbes [CLO07; PM02, Chapitre 5]. Ce type d'opérations peut produire des variétés algébriques de haut degré. Elles pourraient dépasser un degré de 20.

4 Contributions

4.1 Formulation du problème

Nous abordons le problème de la visualisation des courbes et surfaces algébriques implicites, c'est-à-dire des courbes définies par une équation polynomiale bivariée

$$P(x, y) = 0$$

et les surfaces définies par une équation polynomiale trivariée

$$Q(x, y, z) = 0.$$

Notre objectif est de calculer efficacement une représentation discrète sur une grille de résolution fixe et de traiter spécifiquement le cas des courbes ou des surfaces de haut degré à haute résolution. Dans la plupart des approches de pointe, les auteurs analysent et optimisent leurs algorithmes en supposant que le coût d'évaluation de la fonction P ou Q est négligeable. Une telle hypothèse devient invalide lorsque le degré est élevé. Par exemple, un polynôme bivarié de degré total 20 comporte 231 monômes. En particulier, dans notre cas, nous considérons que les degrés sont au moins 20. Par souci de simplicité, nous traitons des grilles carrées $N \times N$ et nous considérons que la résolution N est élevée lorsqu'elle est supérieure à 1 000.

4.2 Vue d'ensemble

Évaluation rapide aux sommets avec des polynômes de Tchebychev

La principale idée pour calculer plus efficacement les courbes et surfaces algébriques implicites est d'utiliser des méthodes d'évaluation multipoints rapides issues du calcul formel afin d'évaluer les polynômes qui définissent ces courbes et surfaces. En effet, un polynôme de degré d peut être évalué en une valeur en complexité arithmétique linéaire en utilisant la méthode de Horner. Ainsi, l'évaluation naïve en d valeurs est quadratique en d. Elle peut en fait être moins coûteuse en utilisant une approche de type « diviser pour régner ». L'algorithme d'évaluation multipoint rapide améliore la complexité de ces évaluations à d en la rendant quasi-linéaire en d (c'est-à-dire $O(d \operatorname{polylog}(d))$) [vzGG13, Chapitre 10].

Si nous limitons notre évaluation multipoint à un ensemble de nœuds de Tchebychev (voir la Section 2.2 de Prerequisites), nous pouvons alors utiliser la transformée en cosinus discrète (on utilise l'acronyme anglais DCT), de la même manière que la transformée de Fourier discrète (on utilise l'acronyme anglais DFT) peut être utilisée pour l'évaluation multipoint sur les racines de l'unité dans le domaine complexe [vzGG13, Section 8.2]. Cette évaluation multipoint hérite de la stabilité numérique de la DCT et nous étendons l'analyse d'erreur déjà connue pour la DFT [BJM+20]. Ainsi, nous convertissons le polynôme de la base monomiale à la base de Tchebychev et appliquons la DCT sur les coefficients du polynôme résultant. Nous obtenons ainsi une évaluation rapide du polynôme original aux nœuds de Tchebychev.

Évaluation sur les arêtes avec de l'arithmétique des intervalles ou de l'approximation de Taylor

Afin de savoir si la courbe plane ou la surface traverse un bord d'un pixel, respectivement un bord d'un voxel, nous adaptons la méthode d'évaluation rapide pour pouvoir évaluer un polynôme univarié P sur les bords. Nous garantissons que nous ne manquons aucun point d'intersection

de la courbe avec la grille sous-jacente et que nous excluons des pixels ou des voxels lorsque l'évaluation de P sur leurs bords est éloignée de zéro. Ces garanties sont obtenues en utilisant une combinaison d'arithmétique des intervalles, d'une analyse minutieuse de l'erreur numérique de l'algorithme de la DCT et de l'approximation de Taylor.

Même si cette analyse de l'algorithme de la DCT nous permet de borner les valeurs du polynôme P aux nœuds de Tchebychev (c_n), elle ne permet pas de borner ses valeurs sur de petits intervalles englobant les arêtes autour des nœuds de Tchebychev. Nous résolvons ce problème en utilisant l'approximation de Taylor en c_n :

$$P(x) = \underbrace{P(c_n) + \cdots + \frac{P^{(k)}(c_n)}{k!}(x - c_n)^k}_{P_{c_n,k}(x)} + R_{c_n,k}(x).$$

Les approximations de Taylor d'ordre k sont des polynômes d'approximation de faible degré $P_{c_n,k}$ qui peuvent être évalués autour de chaque nœud c_n avec l'arithmétique des intervalles. Les coefficients des approximations de Taylor à tous les nœuds de Tchebychev peuvent être calculés efficacement en utilisant la DCT k fois, une fois pour chaque dérivée successive de P. De plus, nous contrôlons l'erreur en bornant le reste $R_{c_n,k}$. En combinant l'évaluation des approximations de Taylor de faible degré à l'évaluation rapide de leurs coefficients à l'aide de la DCT, nous obtenons des évaluations rapides sur les bords.

Dessin de courbe implicite et garanties

Nous présentons trois algorithmes qui utilisent l'évaluation multipoint rapide basée sur la DCT pour tracer des courbes implicites définies par des polynômes bivariés $P(X, Y)$. Étant donné une grille, ils détectent soit si la courbe intersecte les lignes de la grille, soit si la courbe intersecte un pixel de la grille. Les deux premiers algorithmes détectent les intersections avec les arêtes et le troisième détecte les intersections avec les pixels.

Tout d'abord, nous concevons un algorithme qui évalue le polynôme d'entrée P en utilisant la DCT à la fois en x et en y (voir Idée 3). Alors, la grille doit être alignée sur les nœuds de Tchebychev (c_n) dans les deux directions. Bien que cette grille soit non-uniforme, la distance entre deux nœuds consécutifs d'une telle grille de taille $N \times N$ est toujours inférieure à π/N, alors que la distance serait de $1/(N-1)$ pour une grille uniforme. La grille est donc suffisamment dense pour un tracé. L'algorithme évalue la première variable en utilisant la DCT, puis utilise l'approximation de Taylor avec la DCT pour faire les évaluations le long des fibres verticales. C'est ce qu'illustre la moitié supérieure de la Figure 8. Ainsi, en écrivant $P(X, Y) = \sum_{s=0}^{d} p_s(X)Y^s$, nous pouvons utiliser la DCT sur chaque p_i, ce qui conduit à N polynômes univariés $q_n(Y) = P(c_n, Y)$ de degré d, qui peuvent également être évalués à l'aide de la DCT une fois de plus. Cela peut être fait en un nombre quasi-quadratique d'opérations arithmétiques en N si $N \gg d$.

Deuxièmement, nous concevons un algorithme qui incorpore l'évaluation rapide avec le DCT à une méthode de subdivision de l'arithmétique des intervalles (voir Idées 3 et 2). Les domaines sont évalués avec l'arithmétique des intervalles ; s'ils contiennent un zéro, ils sont subdivisés et évalués récursivement jusqu'à une résolution donnée afin de localiser les zéros. Nous tirons profit de la DCT pour évaluer rapidement la première variable, puis nous utilisons la subdivision pour localiser les intersections de la courbe avec les fibres verticales de la grille. C'est ce qu'illustre la moitié inférieure de la Figure 8. Cela nous oblige à utiliser une grille non uniforme alignée sur les nœuds de Tchebychev (c_n) le long de l'axe x. En écrivant $P(X, Y) = \sum_{s=0}^{d} p_s(X)Y^j$,

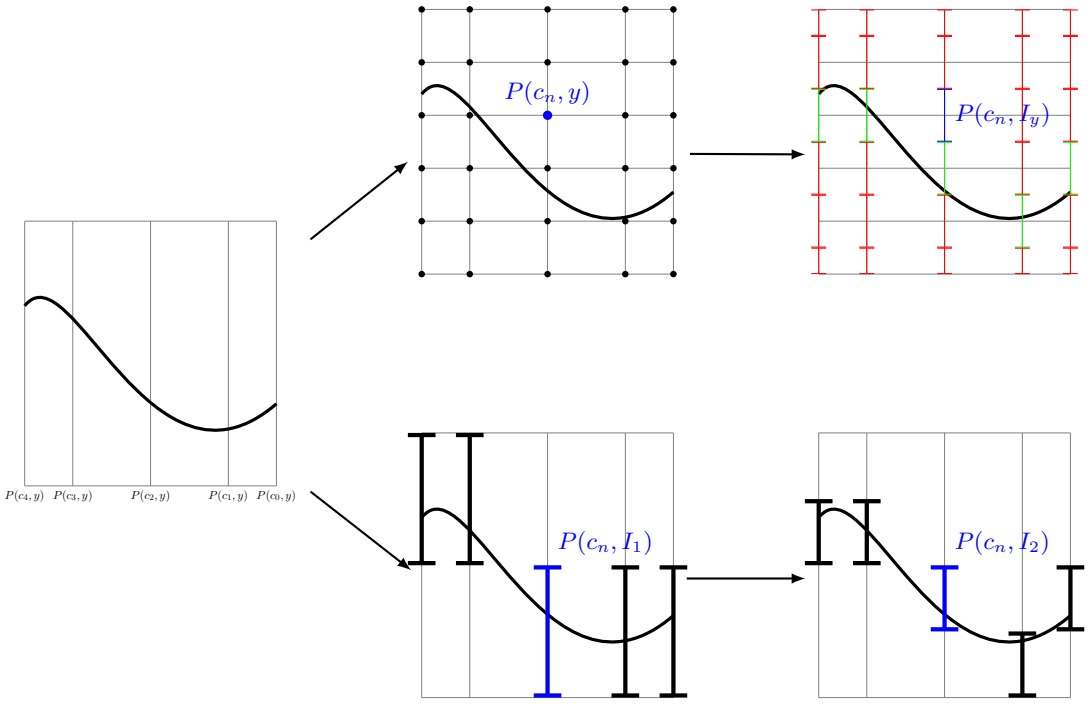


Figure 8: Diagramme des principales étapes des algorithmes englobant les arêtes. D'abord, le polynôme est partiellement évalué en la première variable aux nœuds de Tchebychev. Ensuite, soit les polynômes partiels sont évalués autour des nœuds de Tchebychev le long de l'axe vertical (haut), soit les polynômes partiels sont utilisés pour isoler la courbe par subdivision (bas).

nous pouvons d'abord utiliser la DCT sur chaque p_s, ce qui conduit à N polynômes univariés $q_n(Y) = P(c_n, Y)$ de degré d, puis nous résolvons chaque polynôme $q_n(Y) = P(c_n, Y)$ de degré d avec une approche par subdivision. Si $N \gg d$, cela conduit à une complexité de $\widetilde{O}(dNT)$, où $T \leq N$ est le nombre maximum de nœuds dans les arbres de subdivision considérés. Nous montrons que cette approche fonctionne bien expérimentalement.

Pour ces deux algorithmes, la sortie est un ensemble d'arêtes verticales. Inverser le rôle de X et Y et appeler les algorithmes une seconde fois produit l'ensemble des arêtes horizontales. Obtenir un dessin revient à éclairer un pixel si l'un de ses bords se trouve dans les arêtes de la sortie.

Troisièmement, nous intégrons l'utilisation de l'approximation de Taylor au deuxième algorithme dans l'évaluation de la première variable (voir Idée 4). Cela est illustré par la Figure 9. Cela nous permet de détecter la présence de la courbe à l'intérieur des pixels et ne nécessite qu'un seul appel à l'algorithme. Une fois de plus, nous écrivons $P(X, Y) = \sum_{s=0}^{d} p_s(X)Y^s$. Pour l'évaluation partielle, nous pouvons utiliser les approximations de Taylor aux nœuds de Tchebychev avec la DCT sur chaque p_s, ce qui conduit à N polynômes univariés $q_n(Y) = P(I_{c_n}, Y)$ de degré d, où I_{c_n} est un intervalle autour de c_j. Nous résolvons ensuite chaque polynôme $q_n(Y)$ avec l'approche de subdivision basée sur l'arithmétique des intervalles le long de bandes verticales.

Nous avons implémenté nos approches en Python. Nous vérifions que les temps d'exécution correspondent à l'analyse de la complexité. Nous comparons également nos implémentations avec l'état de l'art, et montrons que pour les hautes résolutions, l'approche basée sur une stratégie mixte utilisant les évaluations multipoints rapides et la subdivision s'avère être la plus rapide.

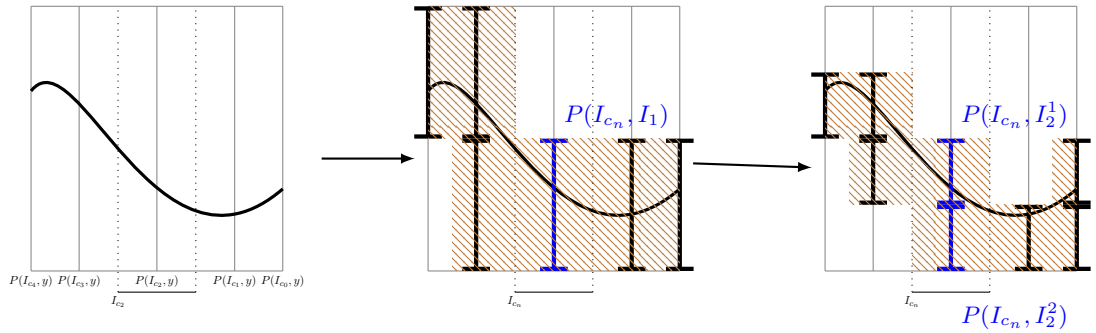


Figure 9: Diagramme des principales étapes de l'algorithme englobant les pixels. D'abord, le polynôme est partiellement évalué en la première variable autour des nœuds de Tchebychev. Ensuite, les polynômes partiels sont utilisés pour isoler la courbe par subdivision (bas).

Nous sommes également en mesure de fournir des garanties pour la courbe de sortie et, au mieux, de la certifier. En effet, certains algorithmes présentés ici font un compromis entre certification et vitesse, mais peuvent encore fournir des garanties partielles sur la sortie.

Dessin de surfaces implicites et garanties

Nous montrons comment les idées utilisées pour les courbes planes peuvent être étendues au cas tridimensionnel. Nous présentons deux algorithmes pour le dessin de surfaces implicites définies par un polynôme trivarié $P(X, Y, Z)$. Nous utilisons des techniques d'évaluation rapide pour les deux premières directions et réservons la subdivision pour la dernière.

Le premier algorithme applique simplement la DCT successivement dans les deux premières variables pour obtenir une évaluation rapide des polynômes partiels $q_{n,m}(Z) = P(c_n, c_m, Z)$ et effectue ensuite une subdivision le long de l'axe z d'une grille de Tchebychev tridimensionnelle. L'algorithme doit être appelé trois fois pour obtenir les arêtes dans chaque direction, en intervertissant les rôles de X, Y et Z. Comme pour l'algorithme bidimensionnel, on obtient un dessin en éclairant un voxel si l'une de ses arêtes se trouve dans la sortie. Ces voxels sont également les seuls que l'algorithme de marching cubes doit examiner. Ce faisant, nous améliorons l'étape la plus coûteuse de cet algorithme. Non seulement nous évaluons le polynôme rapidement avec la DCT, mais nous réduisons également le nombre de voxels à visiter pour construire le maillage approximant. L'algorithme de marching cubes est souvent la première étape de la visualisation, même pour les méthodes avec des grilles raffinées. Par conséquent, ces contributions sont également pertinentes pour ces méthodes.

Le second algorithme utilise l'approximation de Taylor avec la DCT pour l'évaluation des deux premières variables sur des intervalles, obtenant les polynômes partiels $q_{n,m}(Z) = P(I_{c_n}, I_{c_m}, Z)$. Il résout ensuite chaque polynôme $q_{n,m}(Y)$ avec l'approche de subdivision basée sur l'arithmétique des intervalles. Cela nous permet de détecter la présence de la surface à l'intérieur des voxels en un seul appel à l'algorithme.