Fast high-resolution drawing of algebraic curves and surfaces

Nuwan Herath Mudiyanselage
with Guillaume Moroz and Marc Pouget

ROBEX
ENSTA Bretagne, Lab-STICC

November 10, 2023

Overview

Implicit curve drawing

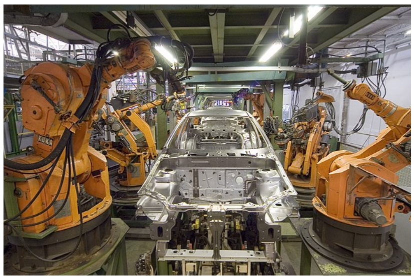
Scientific visualization

Some scientific visualization applications:

- modeling
- medical imaging
- mechanism design

Goal: build an intuition and get an understanding of the data

3D CT reconstruction of distal tibia fracture



Industrial robots from KUKA by Mixabest
(CC BY-SA 3.0)

Implicit curve drawing problem

General problem

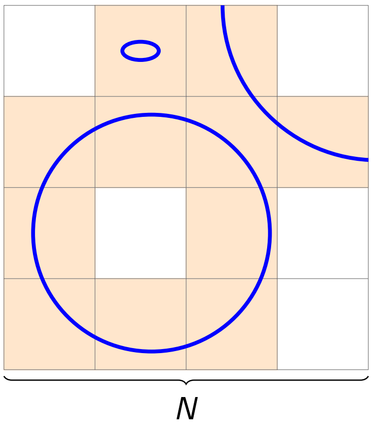
Discrete representation of an implicit curve on a fixed grid

- **Input**:
  - function F
  - resolution N
  - visualization window

  Implicit curve defined as the solution set

$$\{(x, y) \in \mathbb{R}^2 \mid F(x, y) = 0\}$$

- **Output**: drawing (set of pixels)



Implicit curve drawing problem

Our focus

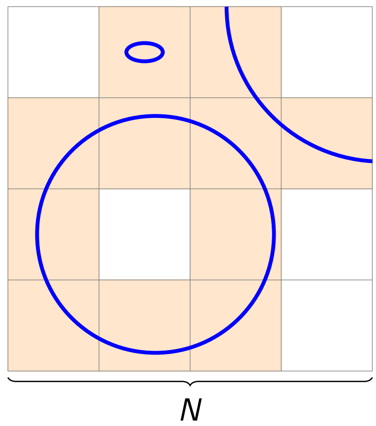
Discrete representation of an **algebraic curve** on a fixed grid

- **Input**:
  - **bivariate polynomial** P of **partial degree** d
  - resolution N
  - **window** $[-1,1] \times [-1,1]$

  Algebraic curve defined as the solution set

$$\{(x,y) \in \mathbb{R}^2 \mid P(x,y) = 0\}$$

- **Output**: drawing (set of pixels)



Goal: fast high-resolution drawing of high degree algebraic curves

- $d \approx 100 \quad \longrightarrow d^2 \approx 10,000$ monomials
- $N \approx 1,000$

Why high degree algebraic curves?

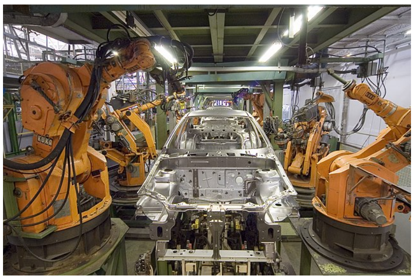
Goal of visualization: build an intuition and get an understanding of the data

In robotics, the configuration space could be of high dimension

$$\mathbb{R}^N \to \mathbb{R}^M$$

Operations on algebraic varieties:

- cut
- projection

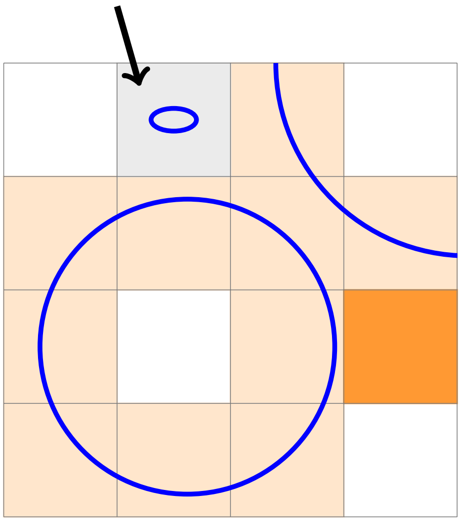


Industrial robots from KUKA by Mixabest

Correctness of the drawing

For numerical reasons, there may be some:

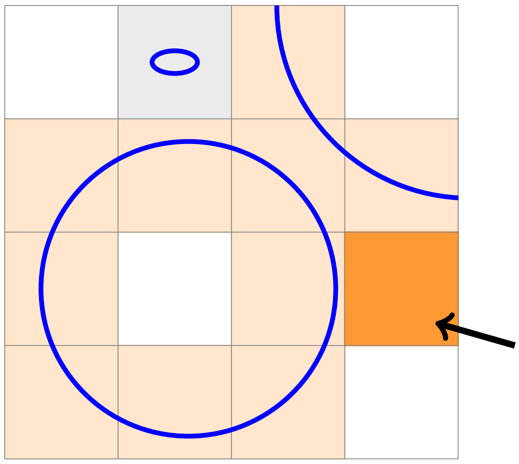
- **False negative** pixels



Correctness of the drawing

For numerical reasons, there may be some:

- **False negative** pixels
- **False positive** pixels

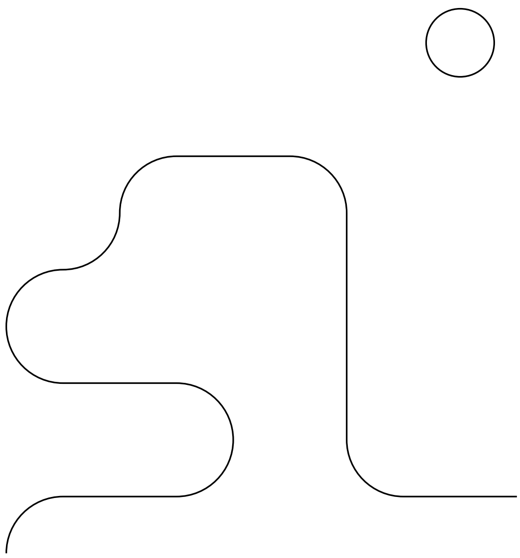


Previous work

Marching squares

The idea

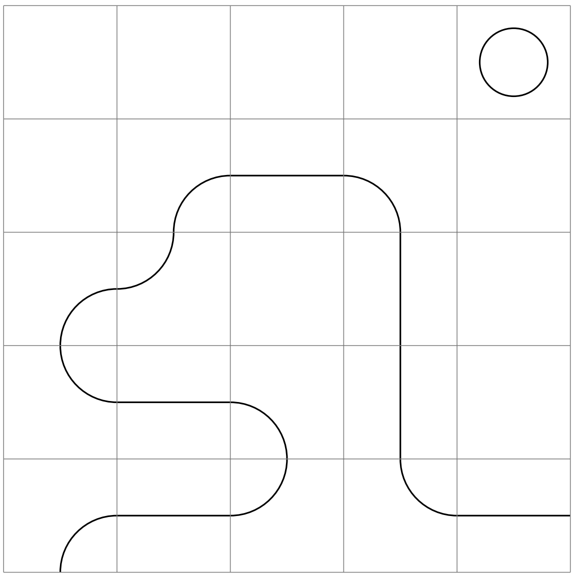
2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]
Implicit curve defined by $P(X, Y) = 0$



Marching squares

The idea

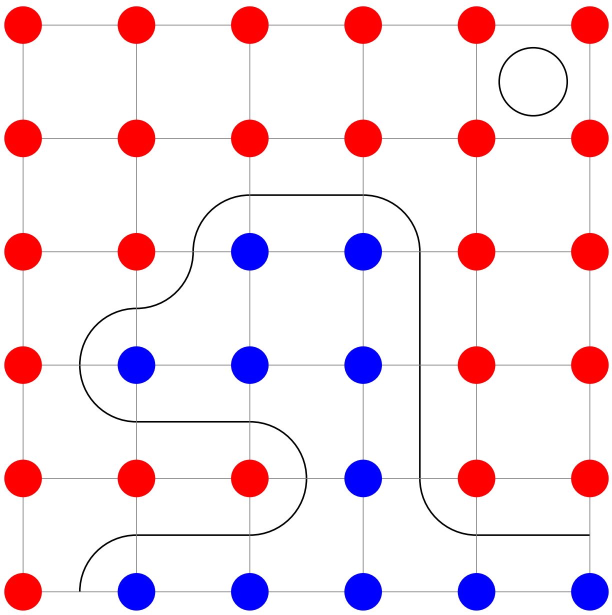
2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]
Implicit curve defined by $P(X, Y) = 0$



Marching squares

The idea

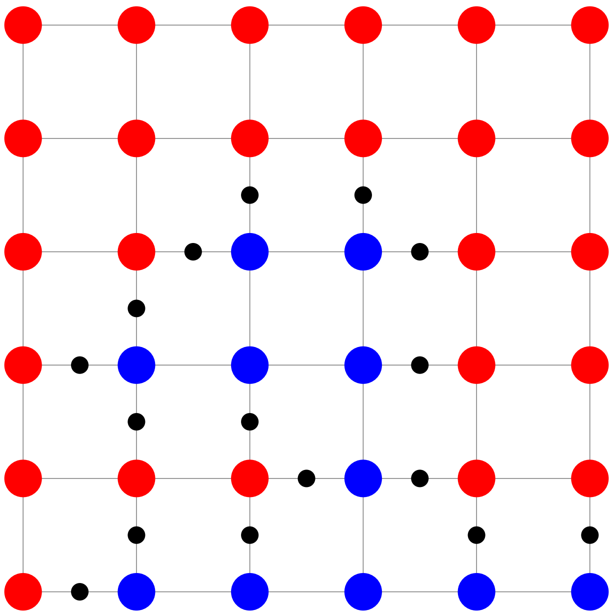
2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]
Implicit curve defined by $P(X, Y) = 0$



Marching squares

The idea

2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]
Implicit curve defined by $P(X, Y) = 0$

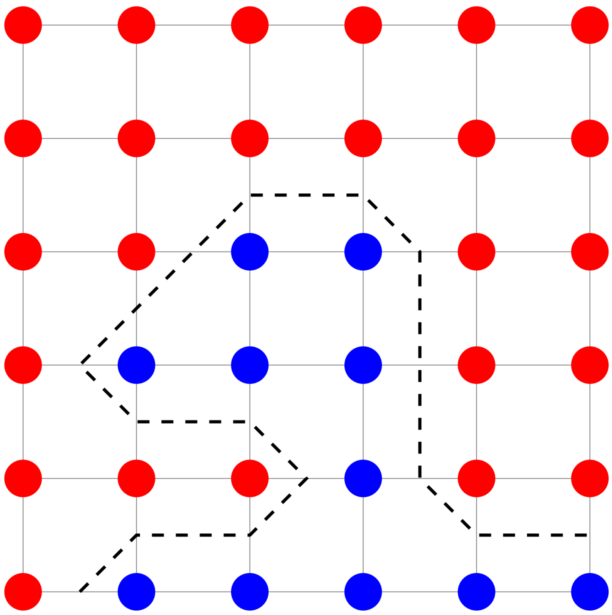


Marching squares

The idea

2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]

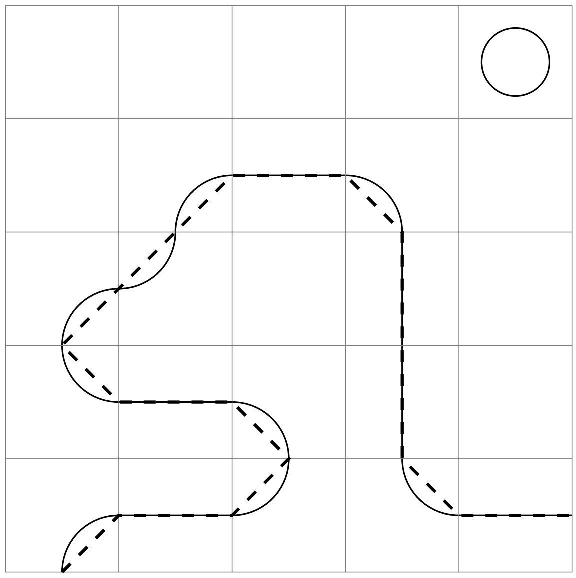
Implicit curve defined by $P(X, Y) = 0$



Marching squares

The idea

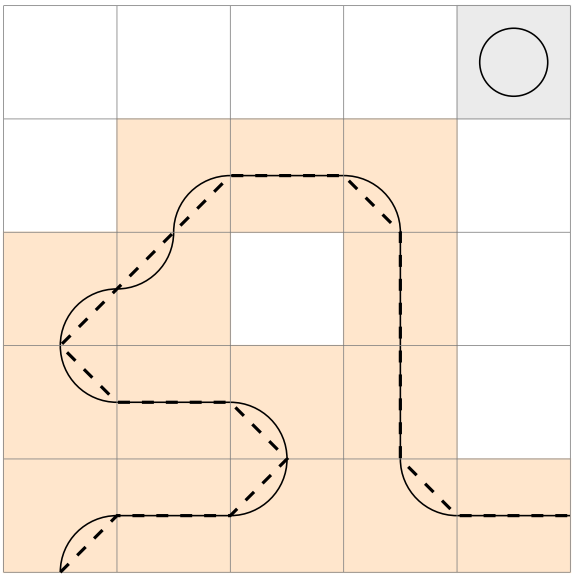
2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]
Implicit curve defined by $P(X, Y) = 0$



Marching squares

The idea

2D variant of the widely used marching cubes algorithm [Lorensen & Cline, 1987]
Implicit curve defined by $P(X, Y) = 0$



Marching squares

Complexity

Complexity (number of elementary operations)
Naive evaluation

$$\theta(d^2 N^2)$$

- d partial degree
- N resolution of the grid

Arithmetic complexity of the marching squares

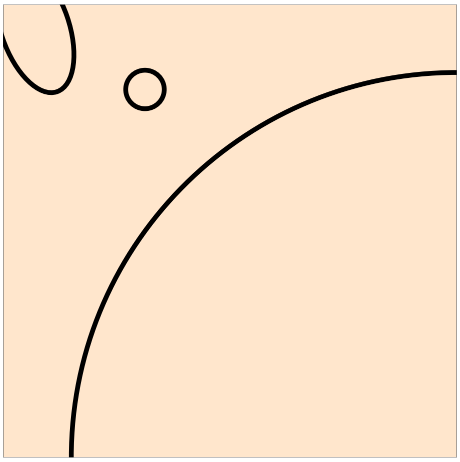
With partial evaluation of $P(x, y)$, assuming $d < N$

$$\theta(dN^2)$$

Slow for high resolutions...
Can we have an algorithm in $O(dN)$?

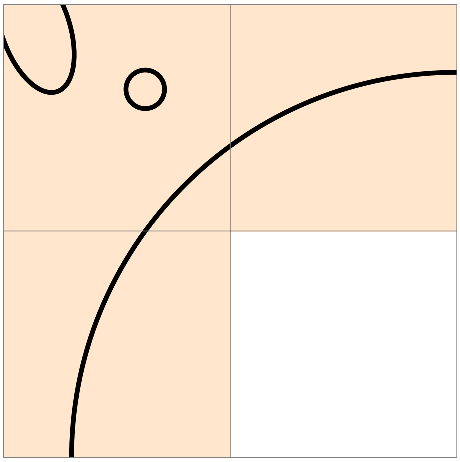
Adaptive subdivision

Local refinements of the grid



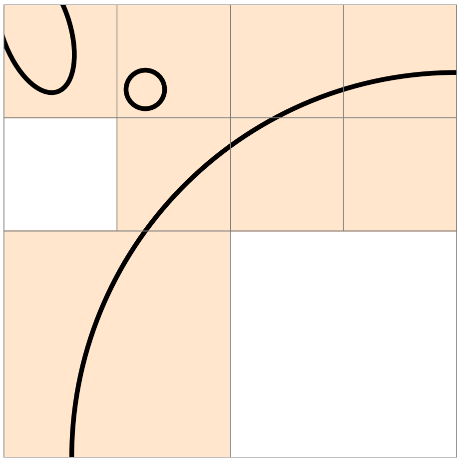
Adaptive subdivision

Local refinements of the grid



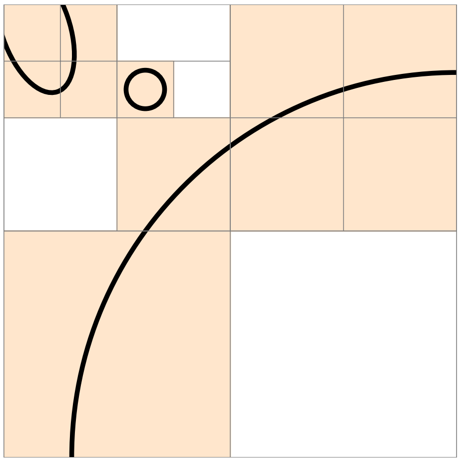
Adaptive subdivision

Local refinements of the grid



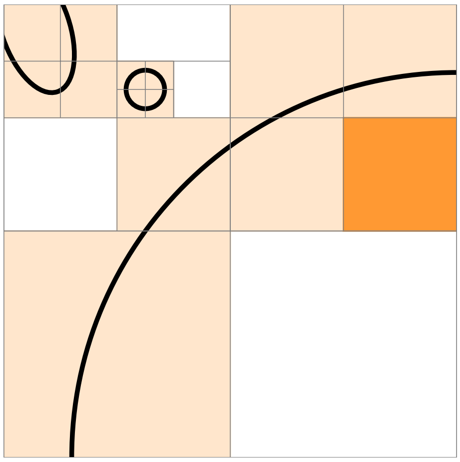
Adaptive subdivision

Local refinements of the grid



Adaptive subdivision

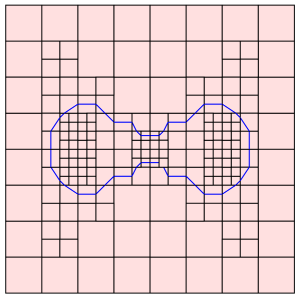
Local refinements of the grid



Methods providing topological correctness

Adaptive 2D subdivision with interval arithmetic

- [Snyder, 1992]
- [Plantinga & Vegter, 2004]
- [Burr et al., 2008]
- [Lin & Yap, 2011]
- . . .



[Lin & Yap, 2011]

Cylindrical algebraic decomposition (CAD)

- [Gonzalez-Vega & Necula, 2002]
- [Eigenwillig et al., 2007]
- [Alberti et al., 2008]
- [Cheng et al., 2009]
- [Kobel & Sagraloff, 2015]
- [Diatta et al., 2018]
- . . .

https://isotop.gamble.loria.fr/

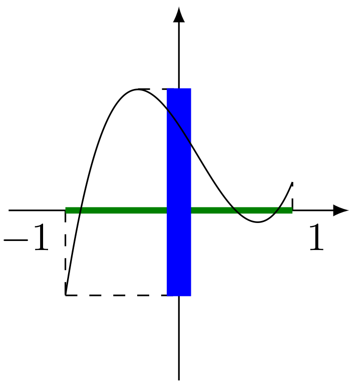
Our approach

Interval arithmetic

Inclusion property

$$P(X) = 2X^3 - X^2 - 1.5X + 0.75$$

How to compute $P(I)$ for $I = [-1, 1]$?



x	-1		$x_1 = \frac{1-\sqrt{10}}{6}$		$x_2 = \frac{1+\sqrt{10}}{6}$		1
$P'(x)$		$+$	0	$-$	0	$+$	
$P(x)$	$P(-1)$	↗	$P(x_1)$	↘	0 $P(x_2)$	↗	$P(1)$

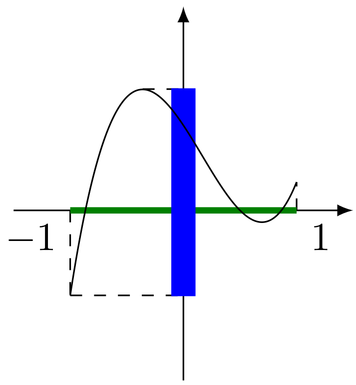
$P(I) = [-0.75, 1.06\ldots]$

Interval arithmetic

Inclusion property

$$P(X) = 2X^3 - X^2 - 1.5X + 0.75$$

How to compute $P(I)$ for $I = [-1, 1]$?



$$\Box P(I) = 2[-1,1]^3 - [-1,1]^2 - 1.5[-1,1] + 0.75$$
$$= [-5.25, 5.25]$$

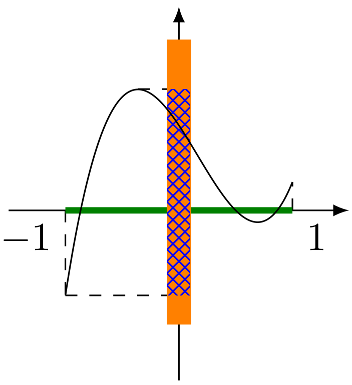
$P(I) = [-0.75, 1.06\ldots]$

Interval arithmetic

Inclusion property

$$P(X) = 2X^3 - X^2 - 1.5X + 0.75$$

How to compute $P(I)$ for $I = [-1, 1]$?



$P(I) = [-0.75, 1.06\ldots]$

$$\square P(I) = 2[-1,1]^3 - [-1,1]^2 - 1.5[-1,1] + 0.75 = [-5.25, 5.25]$$

With Horner's scheme:

$$\square P(I) = ((2[-1,1] - 1)[-1,1] - 1.5)[-1,1] + 0.75 = [-3.75, 5.25]$$

$$P(I) \subseteq \square P(I)$$

Interval arithmetic

Convergence property

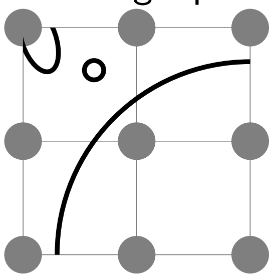
Convergence at a point

With $x \in [a, b]$

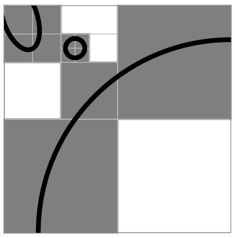
$$\lim_{[a,b] \longrightarrow [x,x]=\{x\}} \square P([a, b]) = P(x)$$

Our approach: guaranteed intersection with the grid

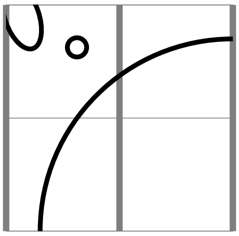
Marching squares



Adaptive subdivision



New approach: evaluation along fibers



⇒ Make it fast and provide some guarantees

An algorithm

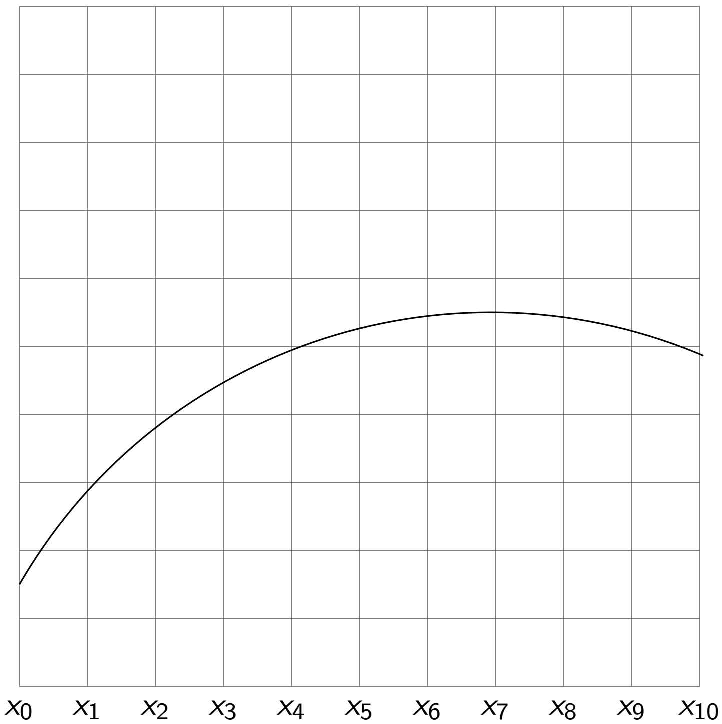
Pixel drawing

- *evaluation in X*
  Chebyshev nodes
  multipoint evaluation with IDCT
  Taylor approximation
- *subdivision in Y*
  naive root finding method

Guarantees
False positive pixels *only*

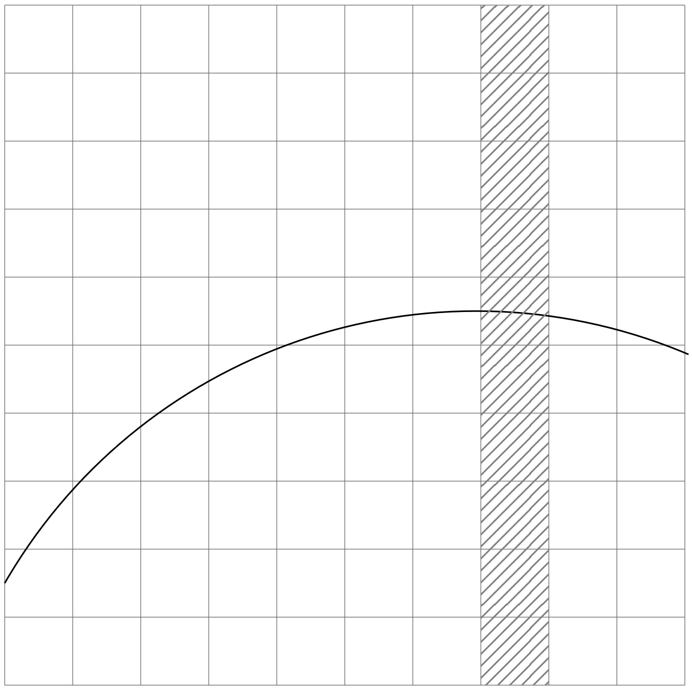
Subdivisions along a stripe

$P([x_k, x_{k+1}], Y) = \sum a_j Y^j$



Subdivisions along a stripe

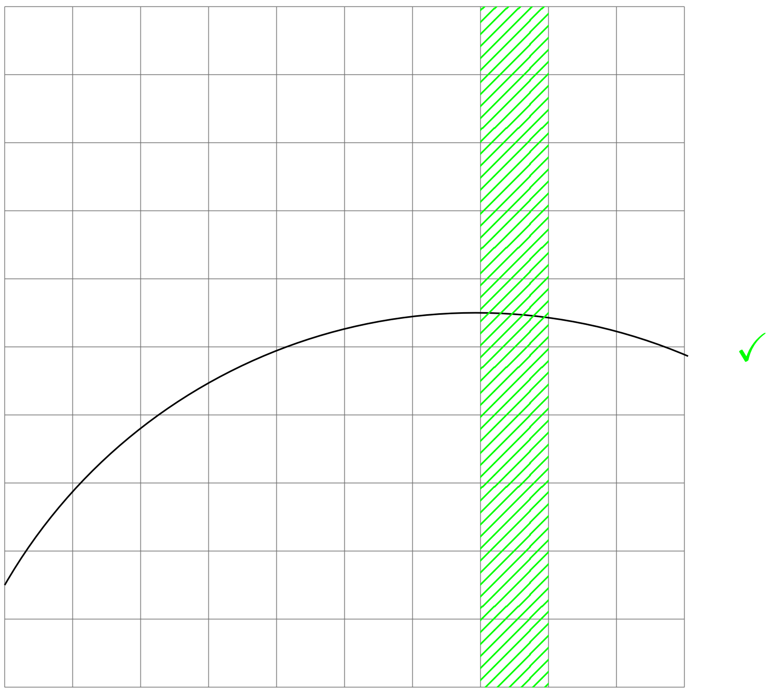
$P([x_k, x_{k+1}], Y) = \sum a_j Y^j$



$P([x_7, x_8], Y)$

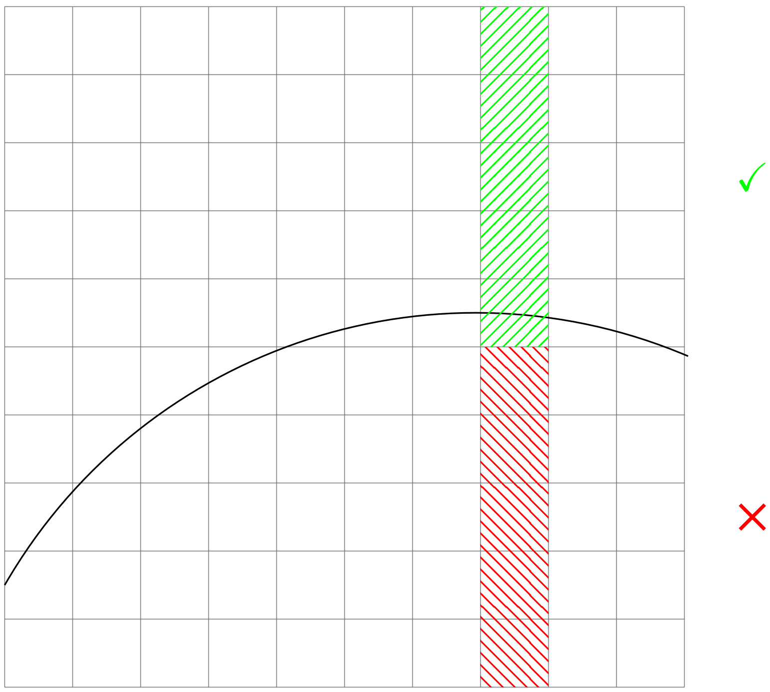
Subdivisions along a stripe

$P([x_k, x_{k+1}], Y) = \sum a_j Y^j$



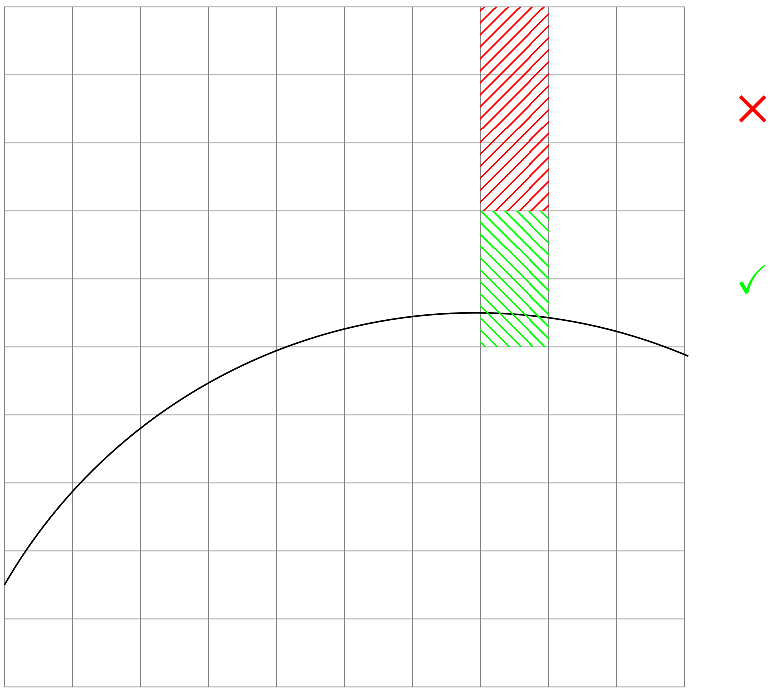
Subdivisions along a stripe

$P([x_k, x_{k+1}], Y) = \sum a_j Y^j$



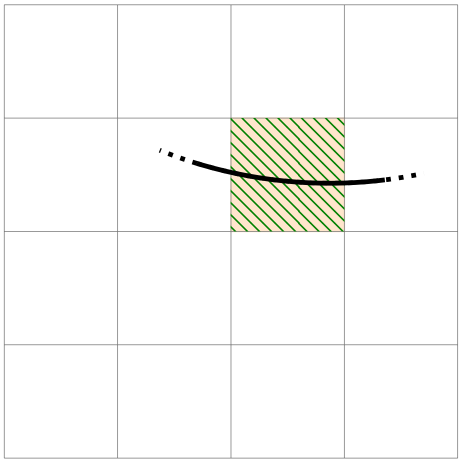
Subdivisions along a stripe

$P([x_k, x_{k+1}], Y) = \sum a_j Y^j$



Pixel drawing
Pixel lighting

- Detect a crossing in pixel of the grid
- Light that pixel

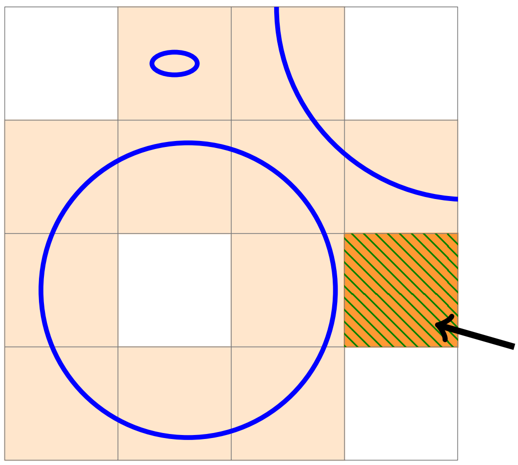


Pixel drawing
False positive and false negative pixels

Some incorrect pixels:

- ~~**False negative** when a connected component lies inside of a pixel~~
- **False positive** when the evaluation on an edge of a pixel is close to zero
  That occurs for a segment S when

$$0 \in \Box P(S) + [-E, E]$$

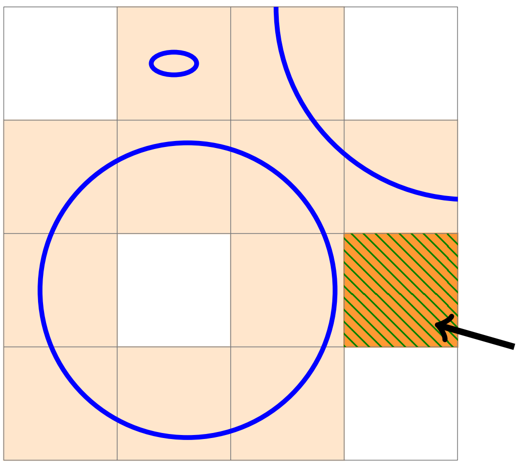


Pixel drawing
False positive and false negative pixels

Some incorrect pixels:

- ~~**False negative** when a connected component lies inside of a pixel~~
- **False positive** when the evaluation on an edge of a pixel is close to zero
  That occurs for a segment S when

$$0 \in \Box P(S) + [-E, E]$$



Certification of segments that are not crossed:

$$0 \notin \Box P(S) + [-E, E]$$
$$\Downarrow$$
$$0 \notin P(S)$$

Fast multipoint evaluation

A prerequisite to fast multipoint evaluation

Chebyshev polynomials

Definition

The Chebyshev polynomials (T_k) verify $\forall k \in \mathbb{N}$, $T_k(\cos\theta) = \cos(k\theta)$

The first three Chebyshev polynomials

$$\cos(0 \cdot \theta) = 1 \qquad T_0 = 1$$
$$\cos(1 \cdot \theta) = \cos(\theta) \qquad T_1 = X$$
$$\cos(2 \cdot \theta) = 2\cos(\theta)^2 - 1 \qquad T_2 = 2X^2 - 1$$

A prerequisite to fast multipoint evaluation

Chebyshev polynomials

Definition

The Chebyshev polynomials (T_k) verify $\forall k \in \mathbb{N}$, $T_k(\cos\theta) = \cos(k\theta)$

Lemma

An arbitrary polynomial p of degree d can be written in terms of the Chebyshev polynomials:

$$p(X) = \sum_{k=0}^{d} \alpha_k T_k(X)$$

A prerequisite to fast multipoint evaluation

Chebyshev polynomials

Definition

The Chebyshev polynomials (T_k) verify $\forall k \in \mathbb{N}$, $T_k(\cos\theta) = \cos(k\theta)$

Lemma

An arbitrary polynomial p of degree d can be written in terms of the Chebyshev polynomials:

$$p(X) = \sum_{k=0}^{d} \alpha_k T_k(X)$$

Lemma

For $N \in \mathbb{N}$, a polynomial p of degree d can be evaluated on the Chebyshev nodes $(c_n)_{0 \le n \le N-1}$ using the IDCT*:*

$$(p(c_n))_{0 \le n \le N-1} = \frac{1}{2}(\alpha_0, \ldots, \alpha_0) + \mathrm{IDCT}((\alpha_k)_{0 \le k \le N-1})$$

A prerequisite to fast multipoint evaluation

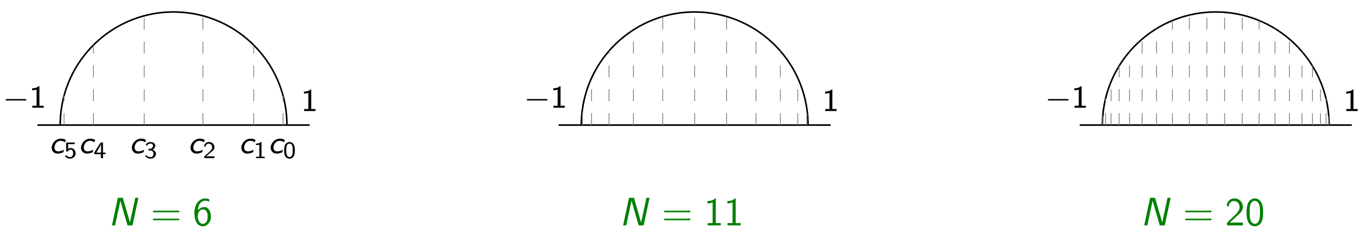
Chebyshev nodes

Definition

For $N \in \mathbb{N}$, the Chebyshev nodes are

$$c_n = \cos\left(\frac{2n+1}{2N}\pi\right), \quad n = 0, \ldots, N-1$$

They are the roots of T_N



Inverse Discrete Cosine Transform

Inverse Discrete Cosine Transform (IDCT): $\alpha_k \rightarrow x_n$

$$x_n = \frac{1}{2}\alpha_0 + \sum_{k=1}^{N-1} \alpha_k \ \cos\left[\frac{\pi k(2n+1)}{2N}\right]$$

IDCT

linear transformation *linear transformation*

$$(\alpha_k) \dashrightarrow (V_k) \xrightarrow{\text{FFT}} (v_k) \dashrightarrow (x_k)$$

$\Rightarrow$ Fast thanks to the Fast Fourier Transform (FFT) algorithm in $O(N \log_2 N)$

[Makhoul, 1980]

Inverse Discrete Cosine Transform

Inverse Discrete Cosine Transform (IDCT): $\alpha_k \to x_n$

$$x_n = \frac{1}{2}\alpha_0 + \sum_{k=1}^{N-1} \alpha_k \ \cos\left[\frac{\pi k(2n+1)}{2N}\right]$$

IDCT

$$(\alpha_k) \overset{\textit{linear transformation}}{\dashrightarrow} (V_k) \xrightarrow{\text{FFT}} (v_k) \overset{\textit{linear transformation}}{\dashrightarrow} (x_k)$$

$\Rightarrow$ Fast thanks to the Fast Fourier Transform (FFT) algorithm in $O(N \log_2 N)$

[Makhoul, 1980]

$$p(c_n) = \sum_{k=0}^{N-1} \alpha_k T_k\left(\cos\left(\frac{2n+1}{2N}\pi\right)\right)$$

Inverse Discrete Cosine Transform

Inverse Discrete Cosine Transform (IDCT): $\alpha_k \to x_n$

$$x_n = \frac{1}{2}\alpha_0 + \sum_{k=1}^{N-1} \alpha_k \ \cos\left[\frac{\pi k(2n+1)}{2N}\right]$$

$$\overset{\text{IDCT}}{\overbrace{(\alpha_k) \xrightarrow{\textit{linear transformation}} (V_k) \xrightarrow{\text{FFT}} (v_k) \xrightarrow{\textit{linear transformation}} (x_k)}}$$

$\Rightarrow$ Fast thanks to the Fast Fourier Transform (FFT) algorithm in $O(N \log_2 N)$

[Makhoul, 1980]

$$p(c_n) = \sum_{k=0}^{N-1} \alpha_k T_k \left(\cos\left(\frac{2n+1}{2N}\pi\right)\right) = \sum_{k=0}^{N-1} \alpha_k \cos\left[\frac{\pi k(2n+1)}{2N}\right]$$

Inverse Discrete Cosine Transform

Inverse Discrete Cosine Transform (IDCT): $\alpha_k \to x_n$

$$x_n = \frac{1}{2}\alpha_0 + \sum_{k=1}^{N-1} \alpha_k \cos\left[\frac{\pi k(2n+1)}{2N}\right]$$

IDCT

$$(\alpha_k) \xrightarrow{\text{linear transformation}} (V_k) \xrightarrow{\text{FFT}} (v_k) \xrightarrow{\text{linear transformation}} (x_k)$$

$\Rightarrow$ Fast thanks to the Fast Fourier Transform (FFT) algorithm in $O(N \log_2 N)$

[Makhoul, 1980]

$$p(c_n) = \frac{1}{2}\alpha_0 + \frac{1}{2}\alpha_0 + \sum_{k=1}^{N-1} \alpha_k \cos\left[\frac{\pi k(2n+1)}{2N}\right]$$

$$(p(c_n))_{0\leq n\leq N-1} = \frac{1}{2}(\alpha_0, \ldots, \alpha_0) + \text{IDCT}((\alpha_k)_{0\leq k\leq N-1})$$

Error of the IDCT

[Makhoul, 1980] and [Brisebarre et al., 2020, Theorem 3.4] yield

Theorem (H., Moroz, Pouget, 2022)

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

$$\|\widehat{x} - x\|_\infty = n\|\alpha\|_\infty O(u).$$

Table: IDCT error bounds for $p = 53$ (double precision)

$N = 2^n$	$1,024$	$2,048$	$4,096$	$8,192$	$16,384$	$32,768$
$\|\widehat{x} - x\|_\infty / \|\alpha\|_\infty$	7.97e-15	8.84e-15	9.72e-15	1.06e-14	1.15e-14	1.23e-14

Algorithms

General idea: edge enclosure

Illustration

$$P(X, Y) = \sum \left(\sum a_{i,j} X^i \right) Y^j = \sum p_j(X) Y^j$$

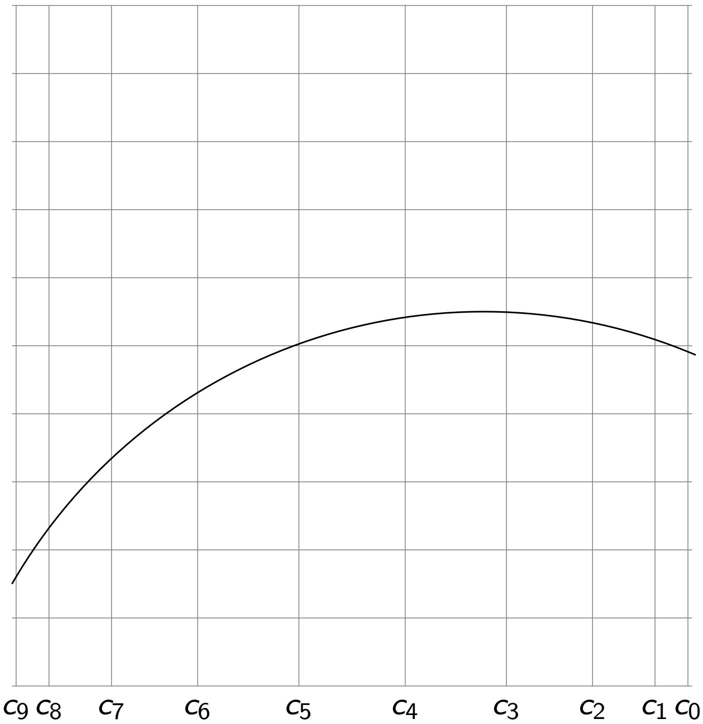
$$p_j(X) = \sum a_{i,j} X^i = \sum \alpha_{i,j} T_i(X)$$

$$(p_j(c_n))_{0 \leq n \leq N-1} = \frac{1}{2}(\alpha_{0,j}, \ldots, \alpha_{0,j}) + \mathrm{IDCT}((\alpha_{k,j})_{0 \leq k \leq N-1})$$

General idea: edge enclosure

Illustration

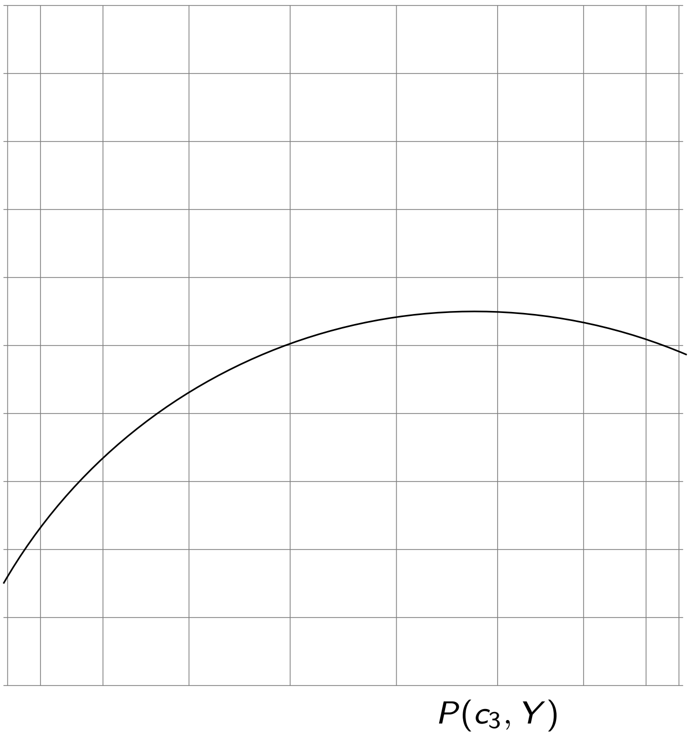
$P(c_n, Y) = \sum p_j(c_n) Y^j$



General idea: edge enclosure

Illustration

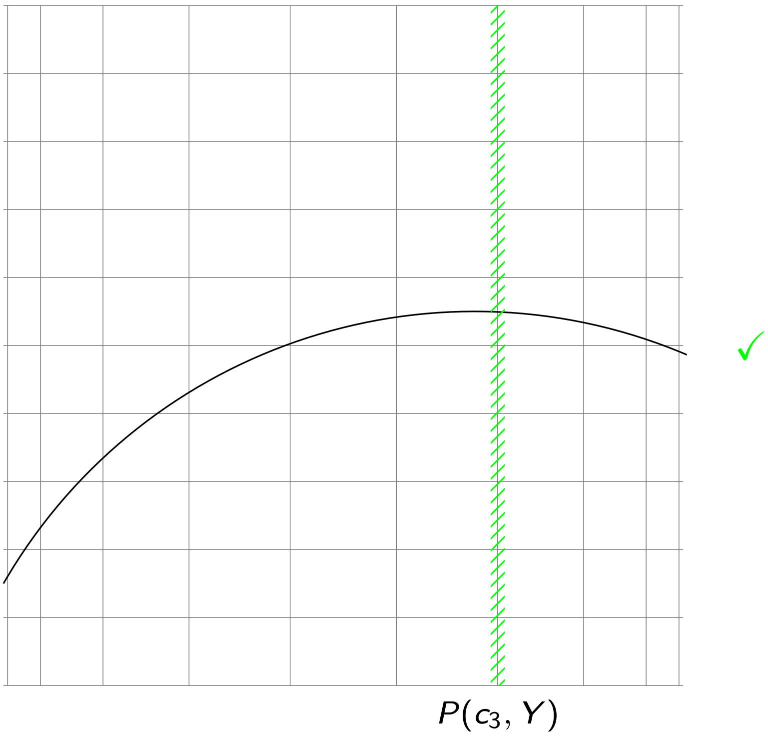
$P(c_3, Y) = \sum p_j(c_3) Y^j$



General idea: edge enclosure

Illustration

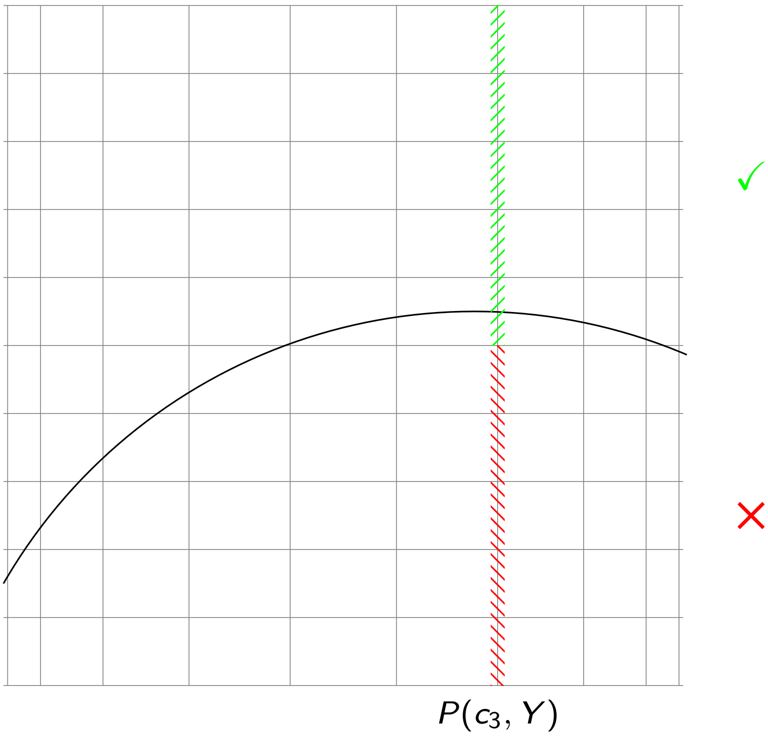
$P(c_3, Y) = \sum p_j(c_3)Y^j$



General idea: edge enclosure

Illustration

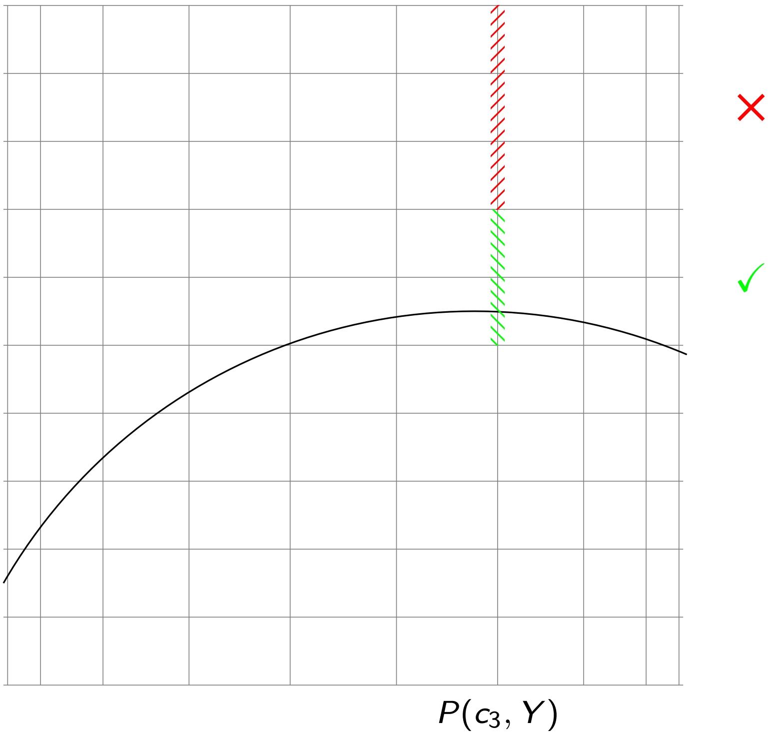
$P(c_3, Y) = \sum p_j(c_3) Y^j$



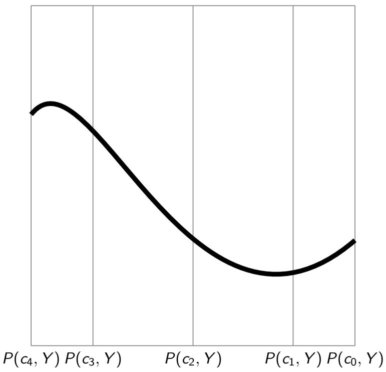
General idea: edge enclosure

Illustration

$P(c_3, Y) = \sum p_j(c_3) Y^j$



An edge enclosing algorithm

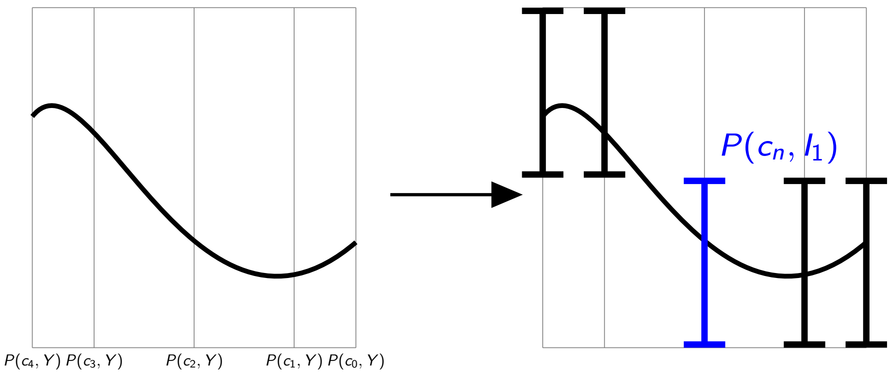


IDCT multipoint evaluation in X
at $c_0, c_1 \ldots$

subdivision in Y

IDCT multipoint evaluation of the partial polynomials of $P(X, Y) = \sum p_j(X) Y^j$

An edge enclosing algorithm

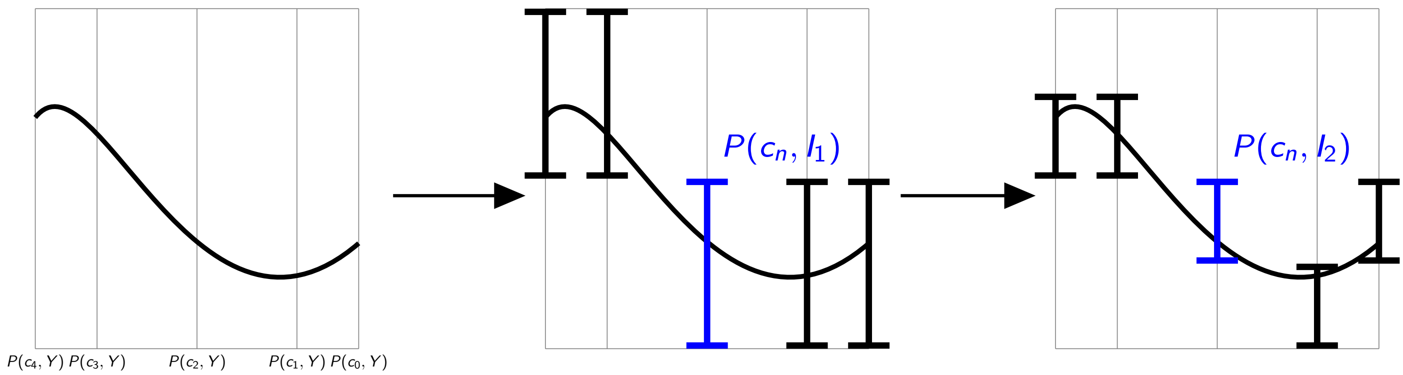


IDCT multipoint evaluation in X
at $c_0, c_1 \dots$

subdivision in Y

IDCT multipoint evaluation of the partial polynomials of $P(X, Y) = \sum p_j(X) Y^j$

An edge enclosing algorithm



IDCT multipoint evaluation in X
at $c_0, c_1 \ldots$

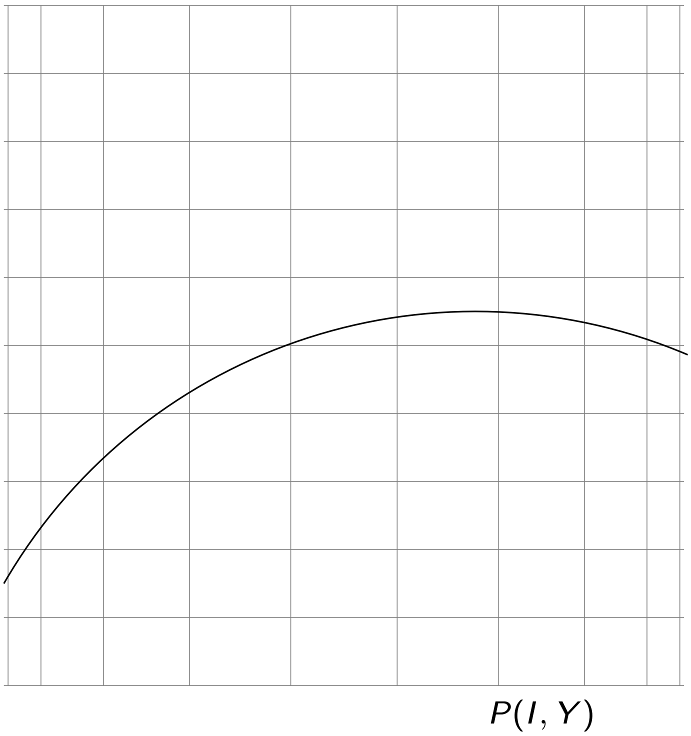
subdivision in Y

IDCT multipoint evaluation of the partial polynomials of $P(X, Y) = \sum p_j(X) Y^j$

General idea: pixel enclosure

Illustration

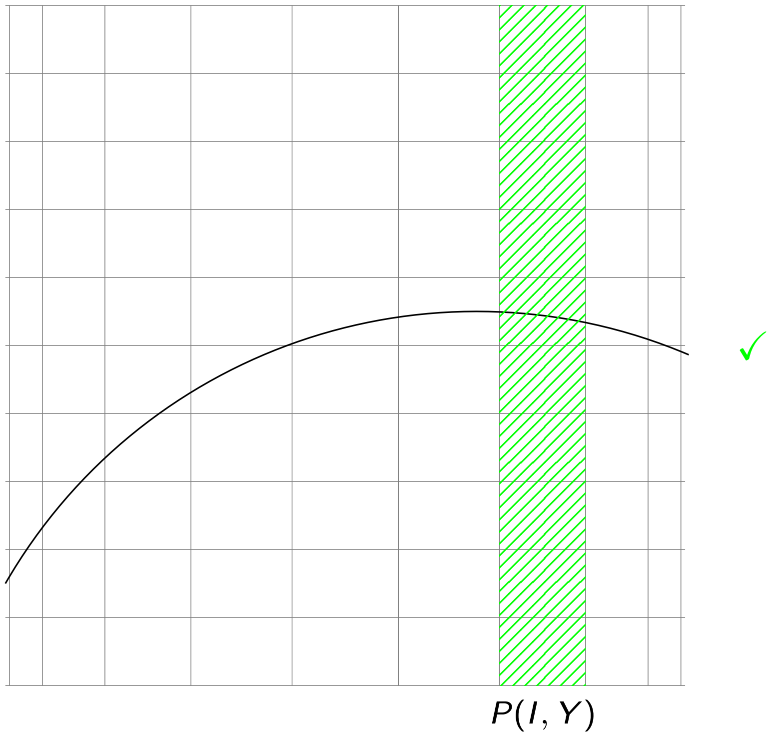
$P(I, Y) = \sum p_j(I) Y^j$



General idea: pixel enclosure

Illustration

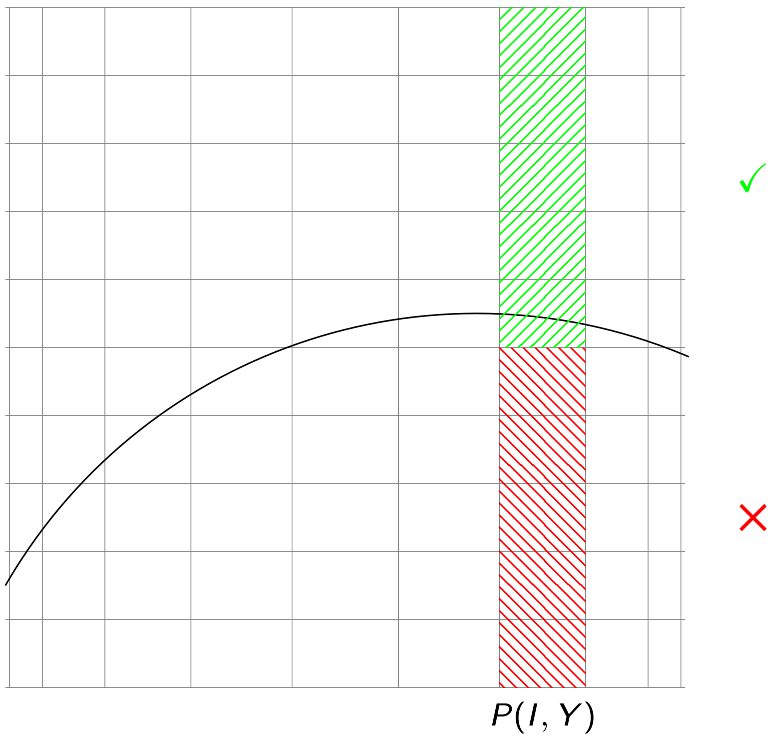
$P(I, Y) = \sum p_j(I) Y^j$



General idea: pixel enclosure

Illustration

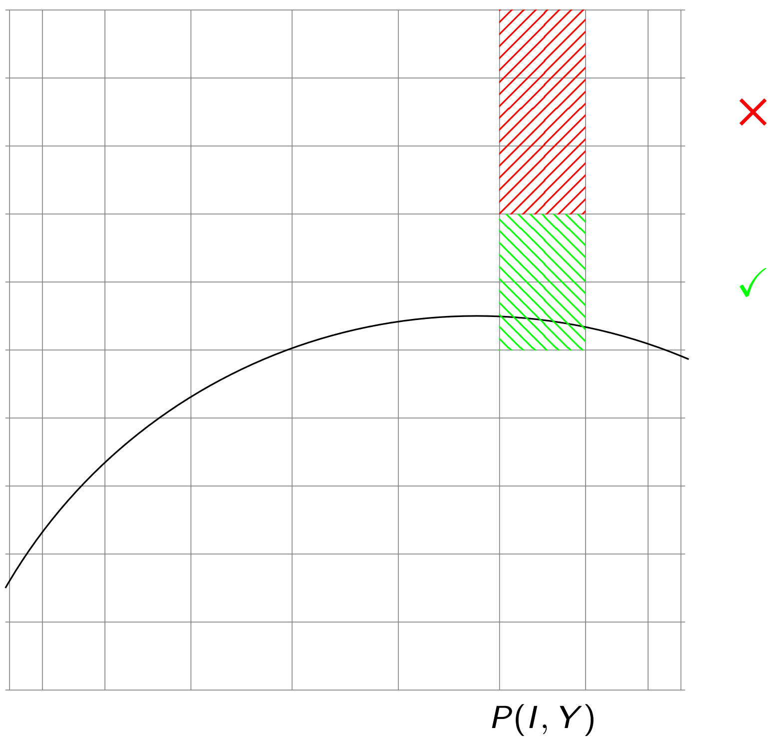
$P(I, Y) = \sum p_j(I) Y^j$



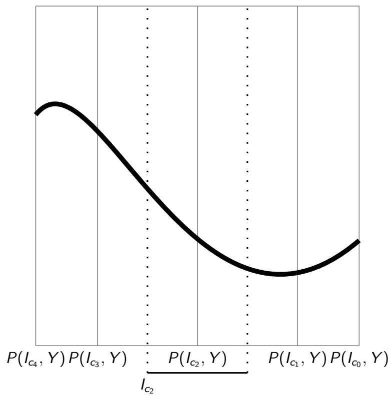
General idea: pixel enclosure

Illustration

$P(I, Y) = \sum p_j(I) Y^j$



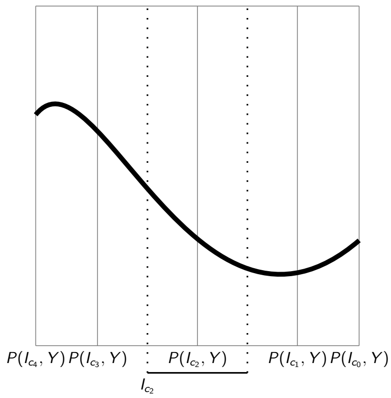
A pixel enclosing algorithm



IDCT multipoint evaluation in X
around $c_0, c_1 \ldots$

subdivision in Y

A pixel enclosing algorithm



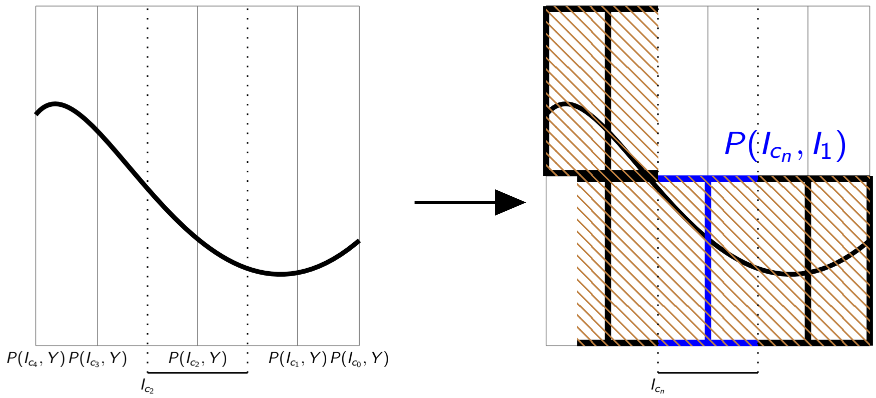
IDCT multipoint evaluation +
Taylor approximation in X

subdivision in Y

Taylor expansion of the partial polynomials of $P(X, Y) = \sum p_j(X) Y^j$

$$\left| p(c_n + r) - \left(p(c_n) + rp'(c_n) + \cdots + \frac{r^m}{m!} p^{(m)}(c_n) \right) \right| \leq \max_{I_{c_n}} \left| p^{(m+1)} \right| \frac{|r|^{(m+1)}}{(m+1)!}$$

A pixel enclosing algorithm



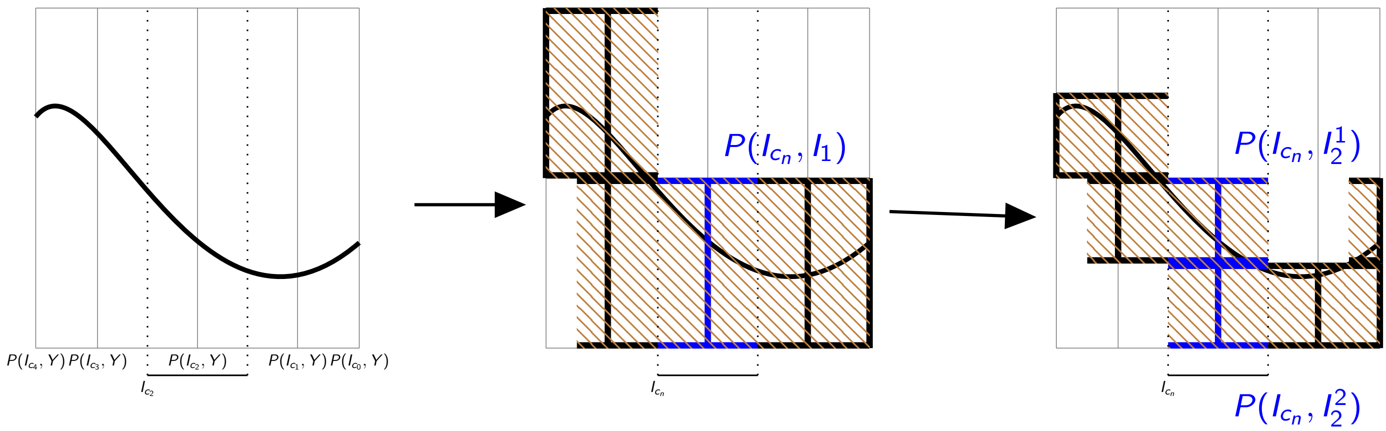
IDCT multipoint evaluation +
Taylor approximation in X

subdivision in Y

Taylor expansion of the partial polynomials of $P(X,Y) = \sum p_j(X) Y^j$

$$\left| p(c_n + r) - \left(p(c_n) + r p'(c_n) + \cdots + \frac{r^m}{m!} p^{(m)}(c_n) \right) \right| \leq \max_{I_{c_n}} \left| p^{(m+1)} \right| \frac{|r|^{(m+1)}}{(m+1)!}$$

A pixel enclosing algorithm



IDCT multipoint evaluation +
Taylor approximation in X

subdivision in Y

Taylor expansion of the partial polynomials of $P(X,Y) = \sum p_j(X)Y^j$

$$\left| p(c_n + r) - \left(p(c_n) + rp'(c_n) + \cdots + \frac{r^m}{m!} p^{(m)}(c_n) \right) \right| \leq \max_{I_{c_n}} \left| p^{(m+1)} \right| \frac{|r|^{(m+1)}}{(m+1)!}$$

Complexities

Arithmetic complexities

multipoint evaluation and subdivision	$O(\ d^3 + \ dN\log_2(N) + dNT)$
multipoint Taylor approximation and subdivision	$O(\ md^3 + mdN\log_2(N) + dNT)$

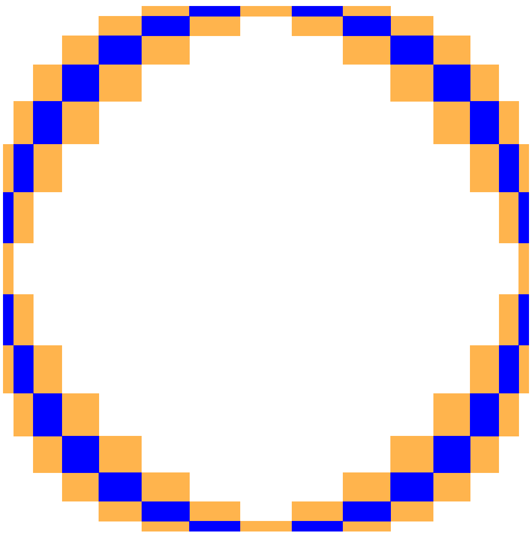
d partial degree
N resolution
T maximum number of nodes of the subdivision trees over all vertical fibers / stripes

With a constant number of branches in the window, we expect $T = O(\log_2(N))$

Experiments

Pixel classification

- crossed: blue
- not crossed: white
- undecided: yellow

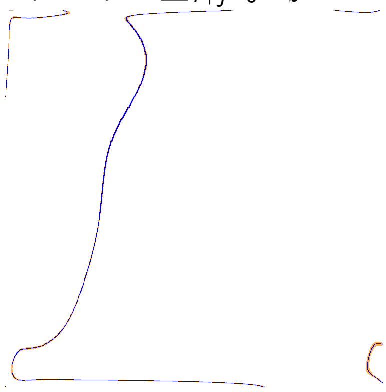


Drawing for two families of polynomials

Experiments on smooth curves $\longrightarrow$ random polynomials
$\xi_{i,j}$: random coefficients in $[-100, 100]$

Kac polynomial

$$P(X,Y) = \sum_{i+j=0}^{d} \xi_{i,j} X^i Y^j$$



Kostlan-Shub-Smale (KSS) polynomial

$$P(X,Y) = \sum_{i+j=0}^{d} \sqrt{\frac{d!}{i!j!(d-i-j)!}} \xi_{i,j} X^i Y^j$$

Drawing for two families of polynomials

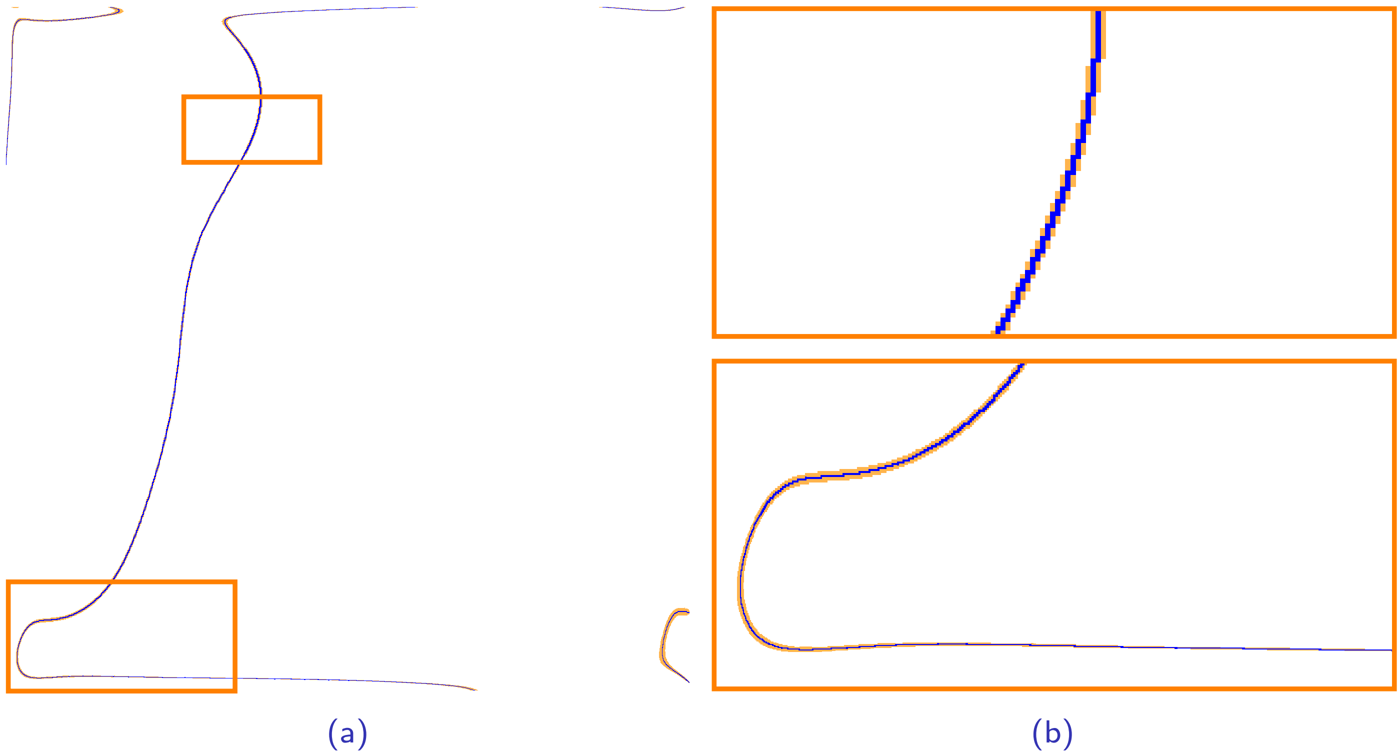
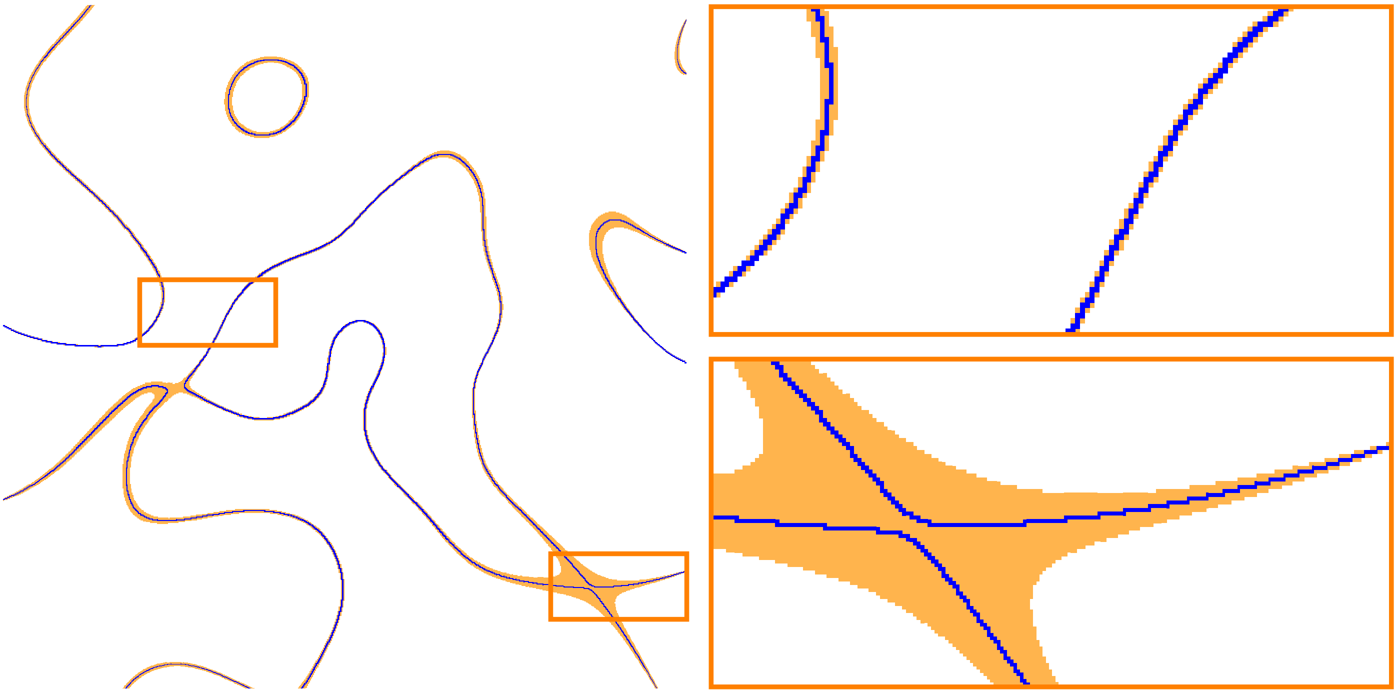


Figure: Kac polynomial of degree $d = 110$ at a resolution $N = 1,024$, $\frac{b}{b+y} = 24\%$

Drawing for two families of polynomials



(a)

(b)

Figure: KSS polynomial of degree $d = 40$ at a resolution $N = 1,024$, $\frac{b}{b+y} = 19\%$

Comparison to state-of-the-art software

Our methods

- edge drawing → curve enclosing edges — false positive and false negative
- pixel drawing → curve enclosing pixels — false positive

Some similar methods

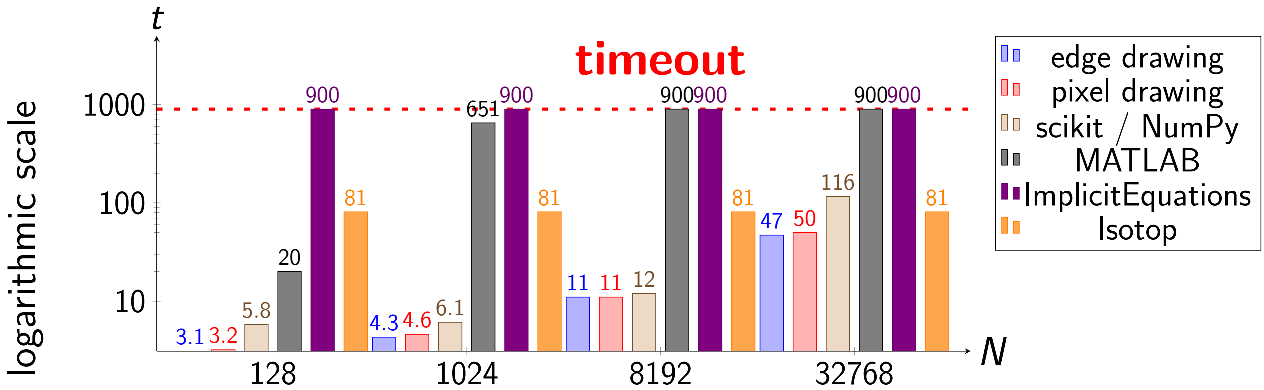
- scikit / NumPy → marching squares — false negative
- MATLAB → could not find the method used — false negative?
- ImplicitEquations → 2D adaptive subdivision — false positive

A topologically correct method

- Isotop → cylindrical algebraic decomposition

Timing

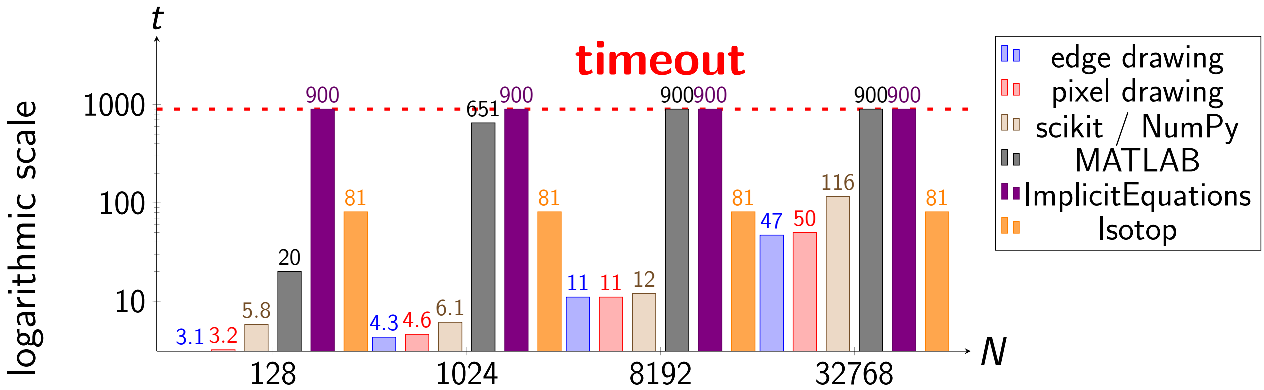
Comparison for a polynomial



Computation times for a **Kac** polynomial of degree 40 (in seconds)

Timing

Comparison for a polynomial



Computation times for a **Kac** polynomial of degree 40 (in seconds)

scikit: $O(dN^2)$

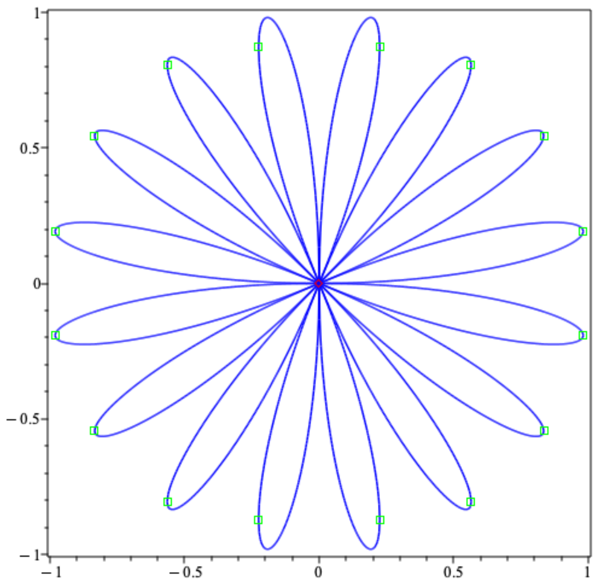
no guarantee
slow when d and N are large

Our methods: $O(dNT)$
as expected $T = O(\log_2(N))$

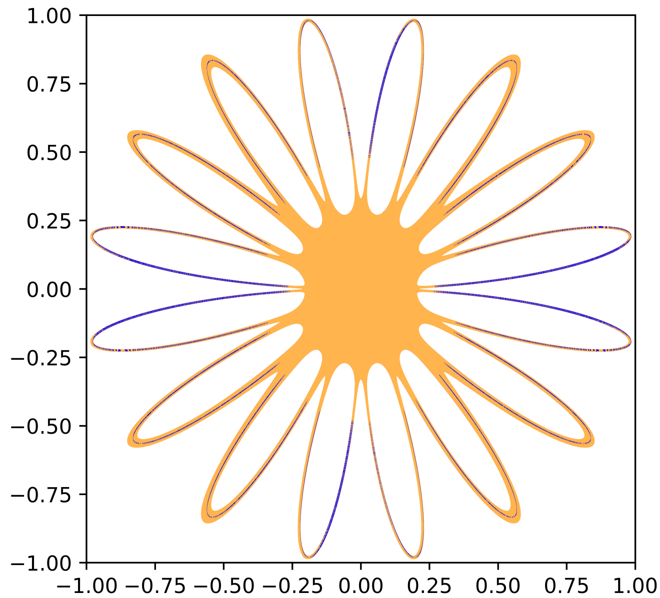
guarantees
fast when d and N are large

Output for a singular curve

Curve: $\mathrm{dfold}_{8,1}$ from Challenge 14 of Oliver Labs[13][37] ($d = 18$)



Isotop



Pixel drawing

Conclusion

Contributions

- Two algorithms
  - enclosure of the edges
  - enclosure of the pixels
- Fast implicit curve and surface algorithms for high resolutions: faster than marching squares and marching cubes
- Better guarantees on the drawing than marching squares
- Ability to handle high degrees ($d > 20$) and high resolutions ($N > 1,000$)