# Fast Gaussian Filtering Algorithm Using Splines

Kentaro Imajo (M2, Kyoto University)

@imos
http://imoz.jp/

# Contents

*Chapter 1 of 7*

# Background and Goal

# Background

Many algorithms such as SIFT
need to compute Gaussian filter.

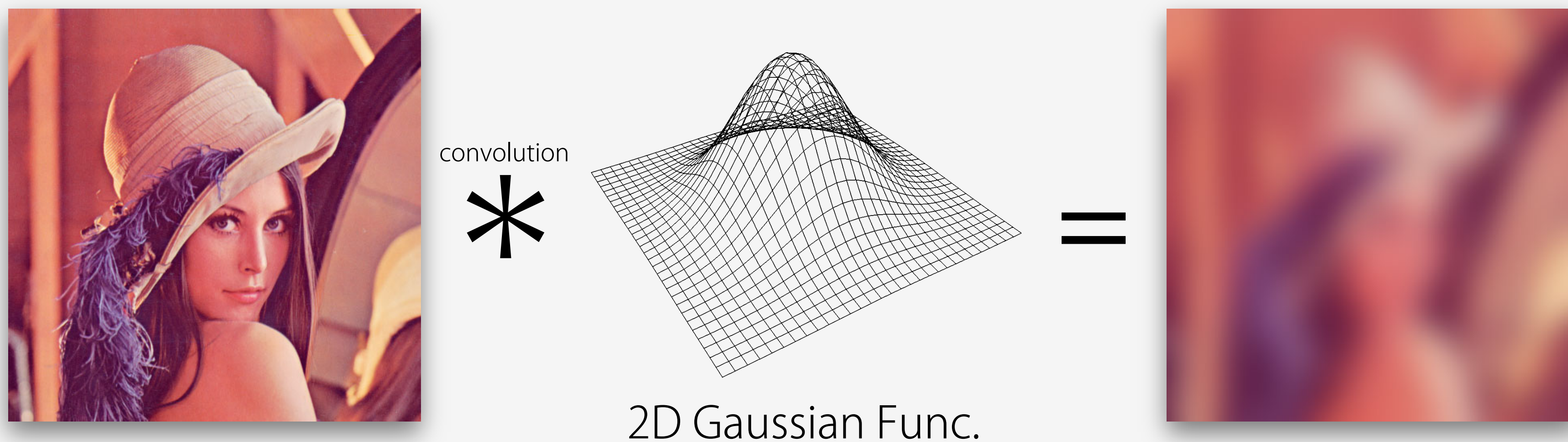



**Figure** Image Blurred by 2D Gaussian Filter

# Gaussian Filter

Gaussian Filter is computed by convolutions with 2D Gaussian function.

2D Gaussian Function:

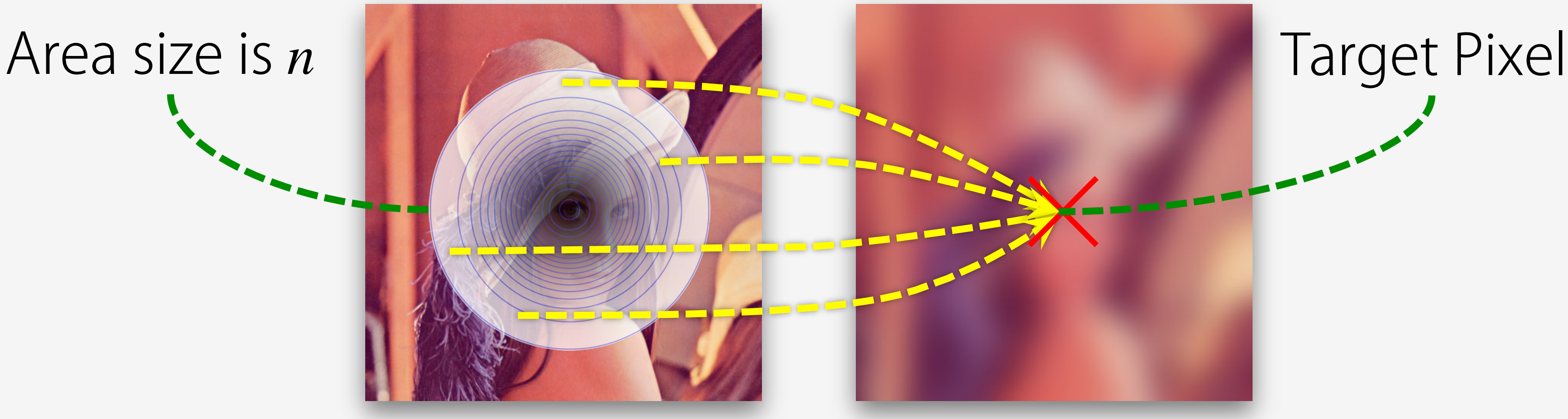
Circle

$$\frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{\boxed{x^2+y^2}}{2\sigma^2}\right)$$

where $\sigma$ is spread.

# Goal

Propose an algorithm to compute one Gaussian-filtered pixel in constant time to area size $n$ ($\propto \sigma^2$).




Naïve method requires $O(n)$ time.

# Preceding Methods

- Naïve method takes $O(n)$ time.
- FFT method is fast for all pixels, but it cannot compute pixels apart.
- Down-sampling method has large errors.

⇓

Within 3% error in constant time

*Chapter 2 of 7*

# Approximation of Gaussian Function

# Spline

Approximate Gaussian function with a spline, which is written as:

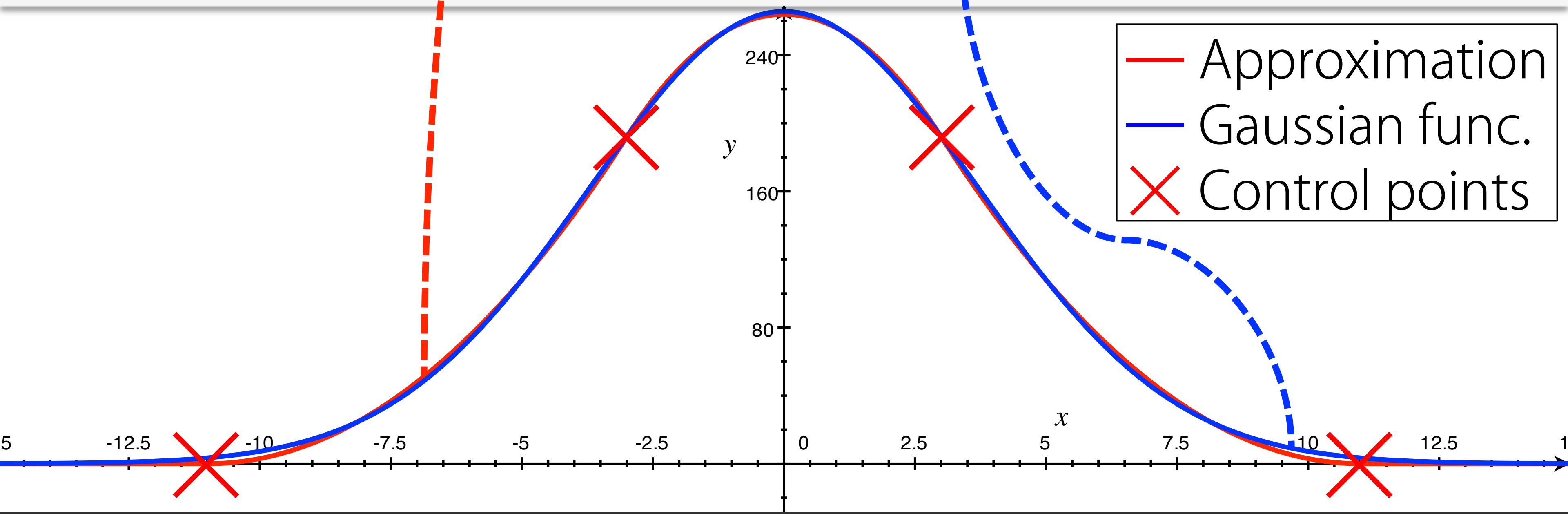
$$\tilde{\psi}(x) = \sum_i a_i \left(x - b_i\right)_+^n$$

where $a, b, n$ are parameters, and $(\cdot)_+$ is $\max(\cdot, 0)$.

* Coordinates $b$ are control points.

# Approximation

$$\begin{aligned}\tilde{\psi}(x) &= 3(x+11)_+^2 - 11(x+3)_+^2 \\ &\quad +11(x-3)_+^2 - 3(x-11)_+^2, \\ &\simeq 2.657 \times 10^2 \exp\left(-\frac{x^2}{5.2720^2}\right).\end{aligned}$$



# Evaluation

Approximation error is about 2%.

(2D error is about 3%.)

**Higher order improves more:**

Approximation error of

4th order approximation is about 0.3%.

$$
\begin{aligned}
\tilde{\psi}(x) \quad = \quad & 70(x+22)_+^4 - 624(x+11)_+^4 + 1331(x+4)_+^4 \\
& -1331(x-4)_+^4 + 624(x-11)_+^4 - 70(x-22)_+^4 .
\end{aligned}
$$

*Chapter 3 of 7*

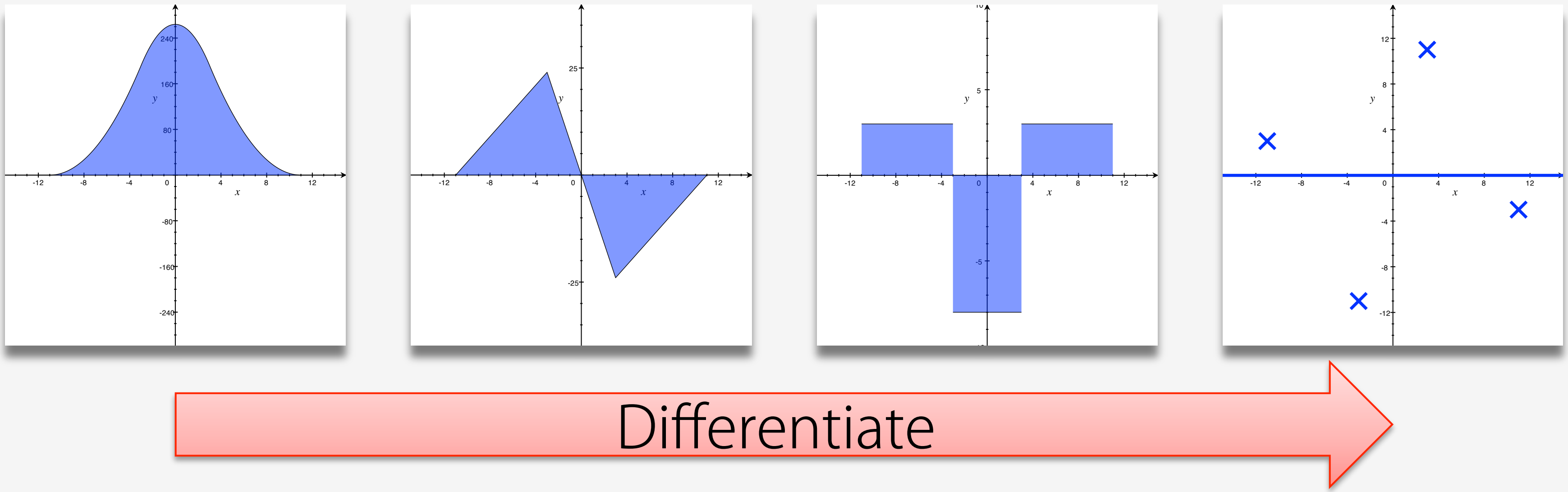
# 1D Convolution

# Convolution

Convolution of a signal and a spline.

**Key Idea:**

Splines get discrete by differentiation.



# Transformation

Transform convolution into summation.

$$
\begin{aligned}
(\tilde{\psi} * I)(x) &= \sum_{\Delta x \in \mathbb{Z}} \tilde{\psi}(\Delta x) I(x - \Delta x) \\
&= \sum_{\Delta x \in \mathbb{Z}} \left( \sum_{i=0}^{m} a_i (\Delta x - b_i)_+^n \right) I(x - \Delta x) \\
&= \underbrace{\boxed{\sum_{i=0}^{m} a_i J(x - b_i)}}_{\text{Several calculations}}, \quad J(x) = \underbrace{\boxed{\sum_{\Delta x \in \mathbb{Z}} \Delta x_+^n I(x - \Delta x)}}_{\text{Pre-computed}}
\end{aligned}
$$

*Chapter 4 of 7*

# 2D Convolution

# 2D Convolution

2D Gaussian func. can be decomposed:

$$\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) = \exp\left(-\frac{x^2}{2\sigma^2}\right)\exp\left(-\frac{y^2}{2\sigma^2}\right)$$

Approximation can be written as:

$$(\tilde{\psi} * I)(x,y) = \sum_{i=1}^{m} a_i \sum_{j=1}^{m} a_j J(x-b_i, y-b_i)$$

where $J(x,y) = \sum_{(\Delta x, \Delta y)\in\mathbb{Z}_+^2} \Delta x^n \Delta y^n I(x-\Delta x, y-\Delta y)$

20 multiplications and 16 additions

# Evaluation

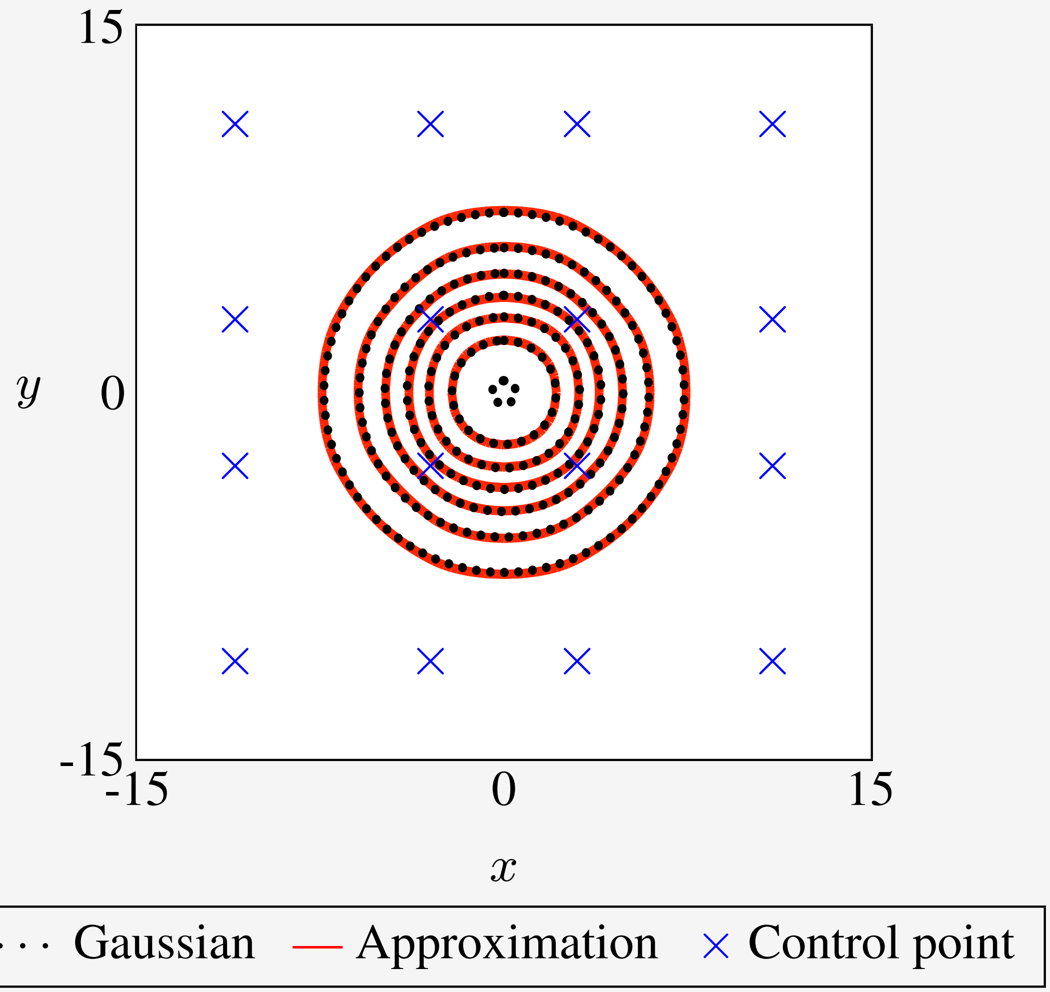



**Figure** Elevation and control points

*Chapter 5 of 7*

# Outline of Algorithm

# Outline of Algorithm

1. Pre-compute $J$ once an image in linear time to the area of the image.

$$J(x, y) = \sum_{(\Delta x, \Delta y) \in \mathbb{Z}_+^2} \Delta x^n \Delta y^n I(x - \Delta x, y - \Delta y)$$

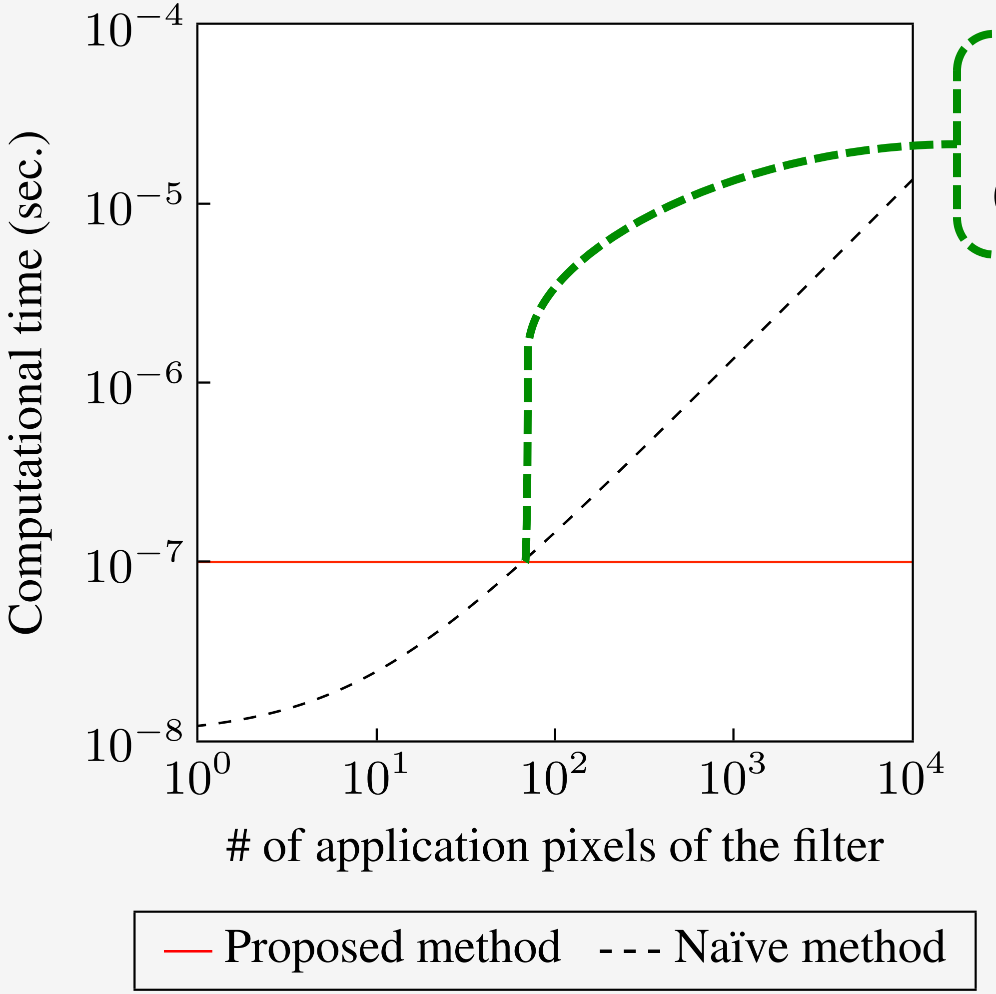
2. Compute a Gaussian filtered value in constant time (tens operations) for any combinations $a, b$.

$$(\tilde{\psi} * I)(x, y) = \sum_{i=1}^{m} a_i \sum_{j=1}^{m} a_j J(x - b_i, y - b_i)$$

*Chapter 6 of 7*

# Experiments

# Experiments




For 70+ pixels
our algorithm is faster

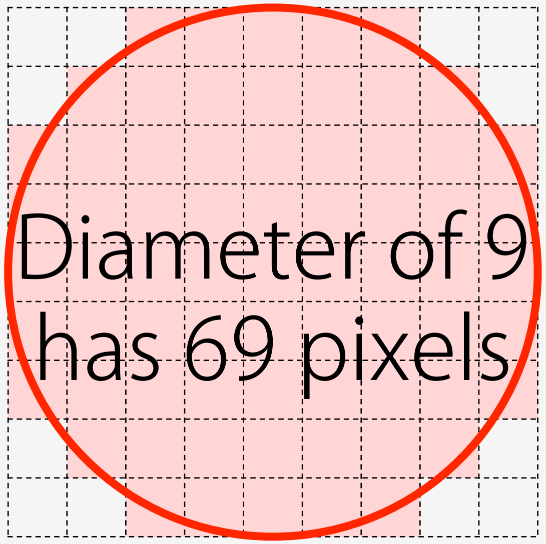



**Figure** Computational time for one pixel on average

*Final Chapter*

# Conclusions

# Conclusions

With pre-computing once an image,
the proposed algorithm computes
any size of the Gaussian filter

- in constant time to size,
- faster than naïve for 70+ pixels,
- within 3% error.