EE Experiments (Computer Vision)
(Lecture Note 12)

Image Inpainting

2019. 11. 25

Prof. Wonjun Kim

School of Electrical and Electronics Engineering

Review of the Last Class

- **Image segmentation**
  - **Concept of the clustering (unsupervised)**
  - **Complete your implementation regarding K-means clustering**
  - **SLIC algorithm**

    - Spatially-limited K-means clustering

    - Apply SLIC segmentation to the face region

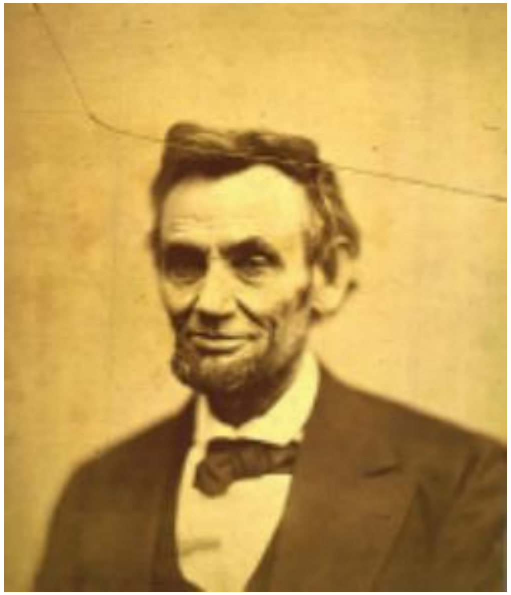
Introduction (1/3)

▪ Image inpainting

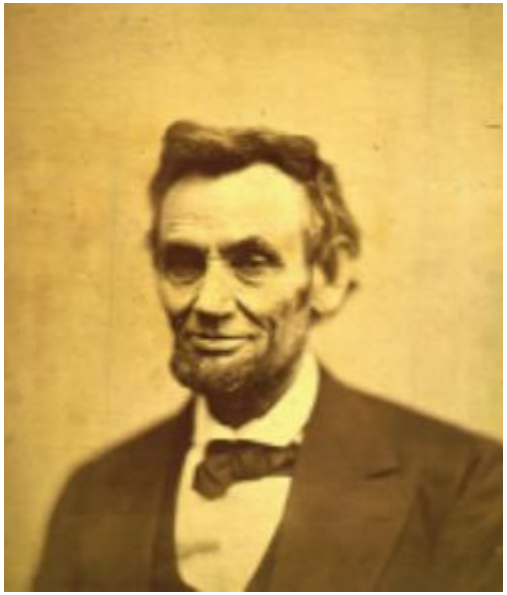
- **Reconstruction problem from small damaged images**
  - Fill out some defected parts in a given image without any additional materials

    ※ It can be regarded as an ill-posed problem

    ※ Applications : removing texts and logos, reconstruction from scan, etc.



Original input



Inpainted result

Introduction (2/3)

▪ Algorithm flow

- **Most inpainting methods follows the two steps :**

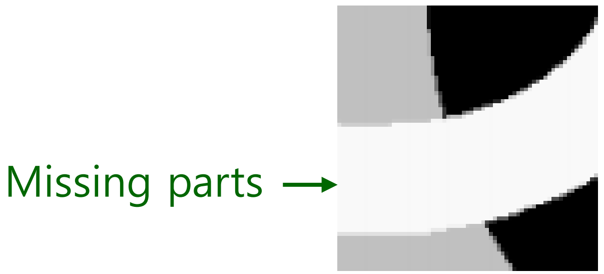
1) Inpainted regions are selected (manually or automatically)

2) Color information is propagated inward from the region boundaries

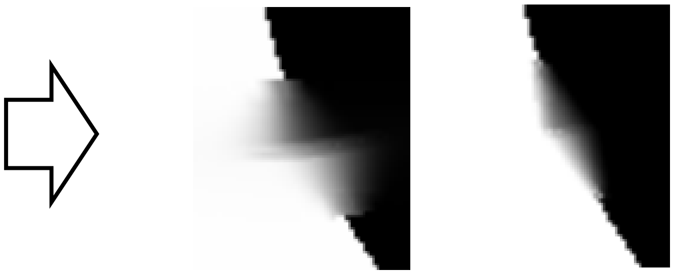
i.e., Known image patches are used to fill missing parts

※ To produce a perceptually plausible reconstruction …

- Continue the ***isophotes*** as smoothly as possible inside the recon. region



Original input



Which one is more desirable ?

Introduction (3/3)

- **Two main strategies**
  - **Partial differential equation (PDE) based approaches**

    - Color propagation inside the missing region

    while preserving the isophotes' direction (but not practical)

    ➔ Diffusion for solving PDE, however, yielding the blurring effect !
  - **Convolution-based approaches**

    - Repeatedly convolving a simple 3x3 filter over missing regions

    using the known image patches (i.e., other image regions)

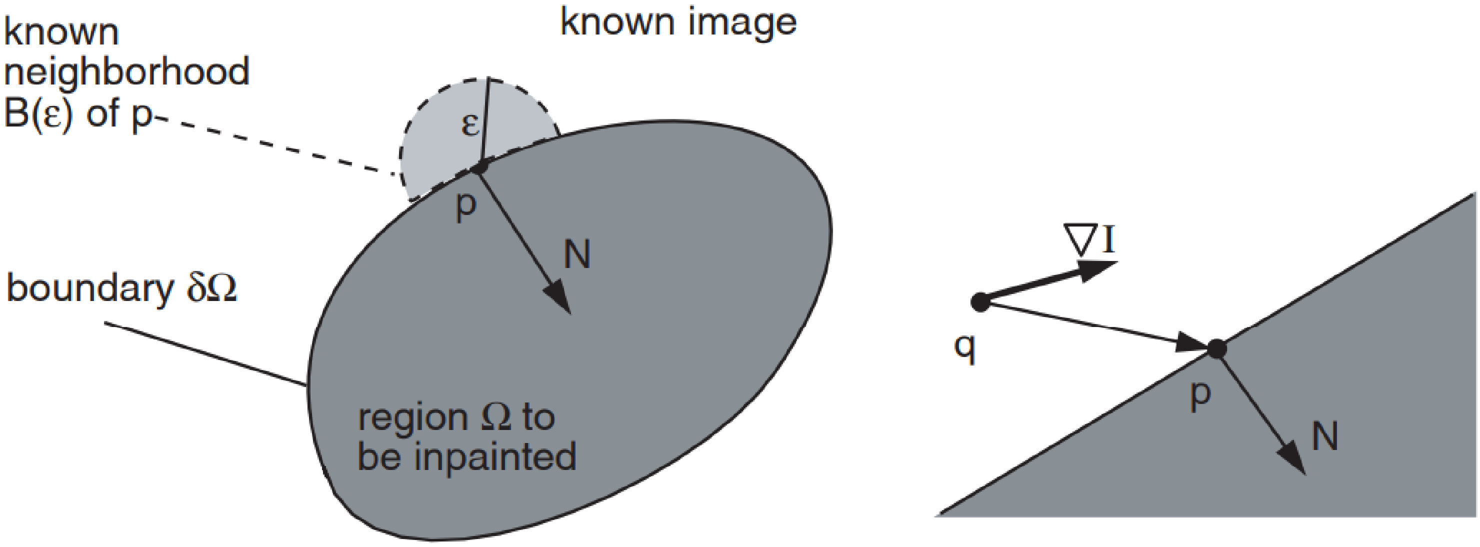
    ➔ No provisions for preserving the isophotes' directions !

※ In this class, a simple and fast method, so-called FMM, is introduced

Fast Marching Method (1/5)

Mathematical formulation [1]

- **Propagation with known image patches**
  - We will inpaint the point ***p*** on the boundary $\partial\Omega$



[Inpainting principle]

1. A. Telea, "An image inpainting technique based on the fast matching method," Journal of Graphics Tools, vol. 9, no. 1, pp. 25-36, 2004.

Fast Marching Method (2/5)

▪ Mathematical formulation – cont’d

- **We consider the first order approximation of p using Taylor’s series**

1) The pixel value of p can be represented as follows :

$$I = (p) = I(q+\Delta x) = I(q) + \nabla I(q)\Delta x + \frac{1}{2!}\Delta x^{\mathrm{T}}\nabla^{2} I(q)\Delta x + \cdots$$
$$\approx I(q) + \nabla I(q)\Delta x$$

where △x denotes p-q (difference of pixel positions)

2) We inpaint p as a function of all points q in p’s neighbor region :

$$I(p) = \frac{\sum_{q \in B_{\varepsilon}(p)} w(p,q)[I(q) + \nabla I(q)(p-q)]}{\sum_{q \in B_{\varepsilon}(p)} w(p,q)}$$

where the weighting factor is $w(p,q) = dir(p,q)dst(p,q)lev(p,q)$

Fast Marching Method (3/5)

▪ Detailed procedure

- **Iteratively compute I(p) using the equation in the previous slide**

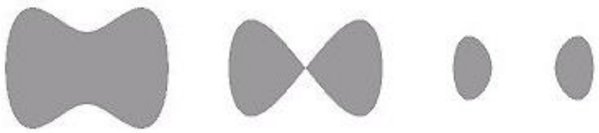
  - That is, propagates $\partial\Omega$ into Ω by advancing pixels in order of their distances

➡ Fast marching method solving the Eikonal equation :

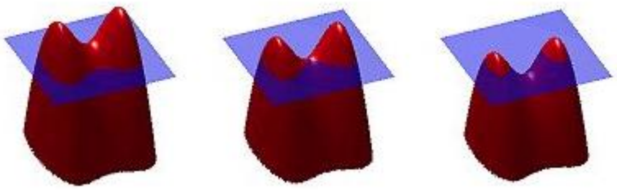
$$|\nabla T| = 1 \ \text{ on } \ \Omega \quad \text{with} \quad T = 0 \ \text{ on } \ \partial\Omega$$

※ What is the basic solution for T in the image processing ?

※ FMM is originally used in the level set computation



: Results in images



: Results in level sets

Fast Marching Method (4/5)

Algorithm details – cont'd

Weight computation

- A product of three factors : $w(p,q) = dir(p,q)dst(p,q)lev(p,q)$

where $dir(p,q) = \frac{p-q}{\| p-q \|} \cdot N(p)$ (inner product)

$$dst(p,q) = \frac{d_0^2}{\| p-q \|^2}$$

$$lev(p,q) = \frac{T_0}{1+|T(p)-T(q)|}$$

* Similar to the direction of normal at p

* Close to the point p in a geometric manner

* Close to the contour through the point p

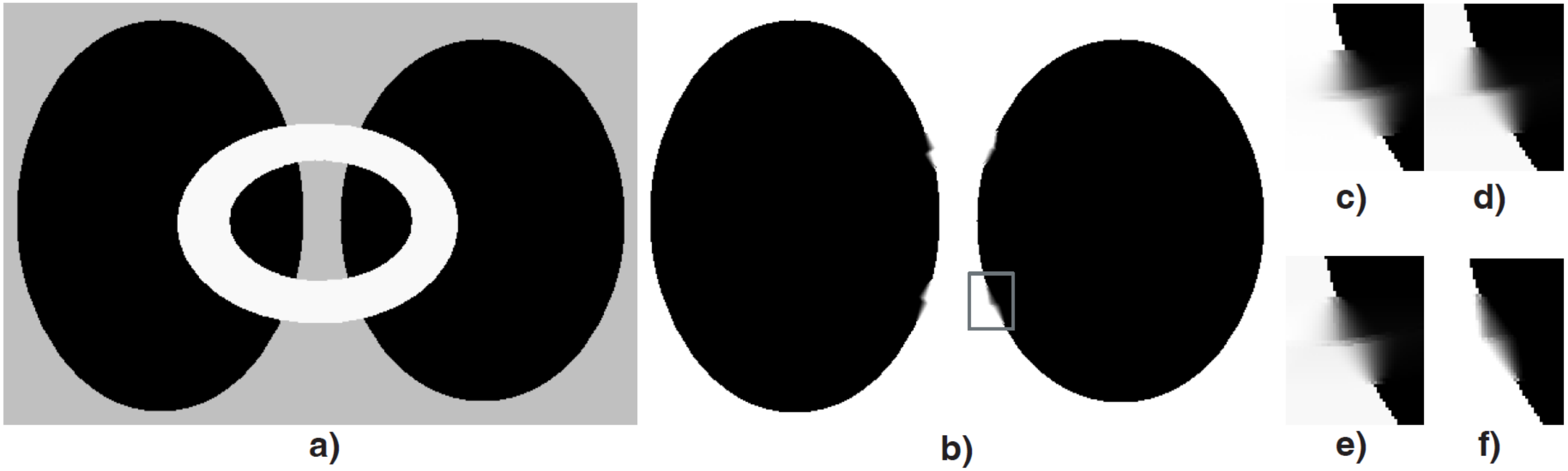
Weight increases !

Fast Marching Method (5/5)

▪ Algorithm details – cont'd

• Effect of each weight component

- A product of three factors : $w(p,q) = dir(p,q)dst(p,q)lev(p,q)$



a) Thick region to inpaint   b) Inpainting result of a)

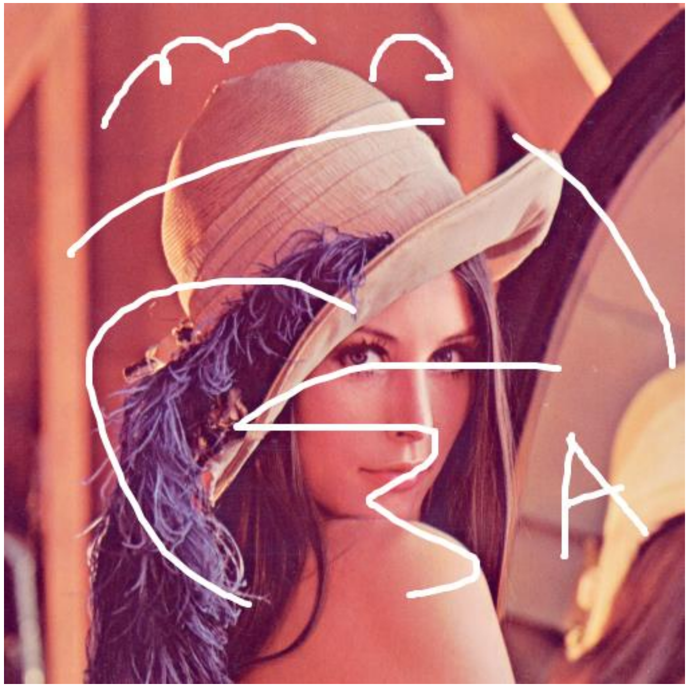
Effect of weighting functions :

c) direction   d) direction and geometric distance

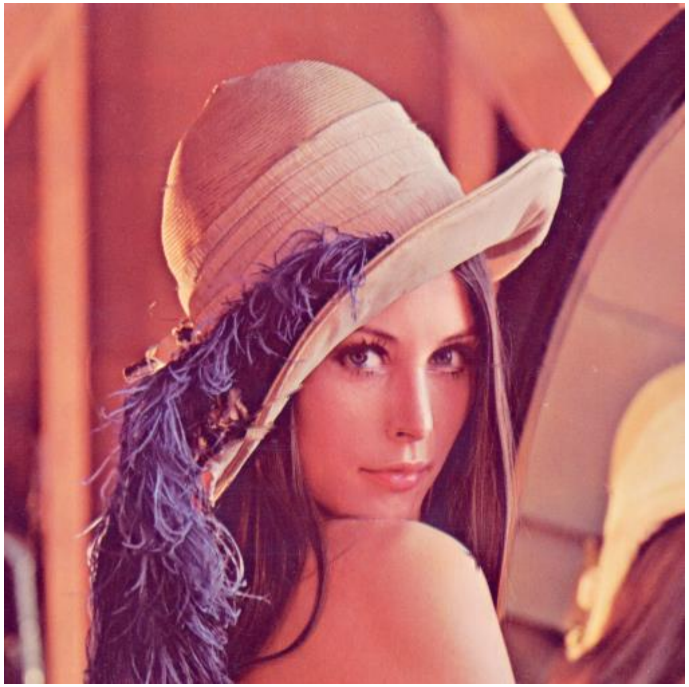
e) Direction and level set distance   d) direction, geometric, and level set distance

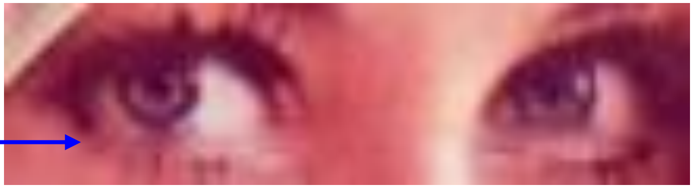
Examples of Inpainting (1/2)

- **Examples of segmentation results**
  - **Weakness : blurring effect when thickness is more than 10~15 pixels**



Original input



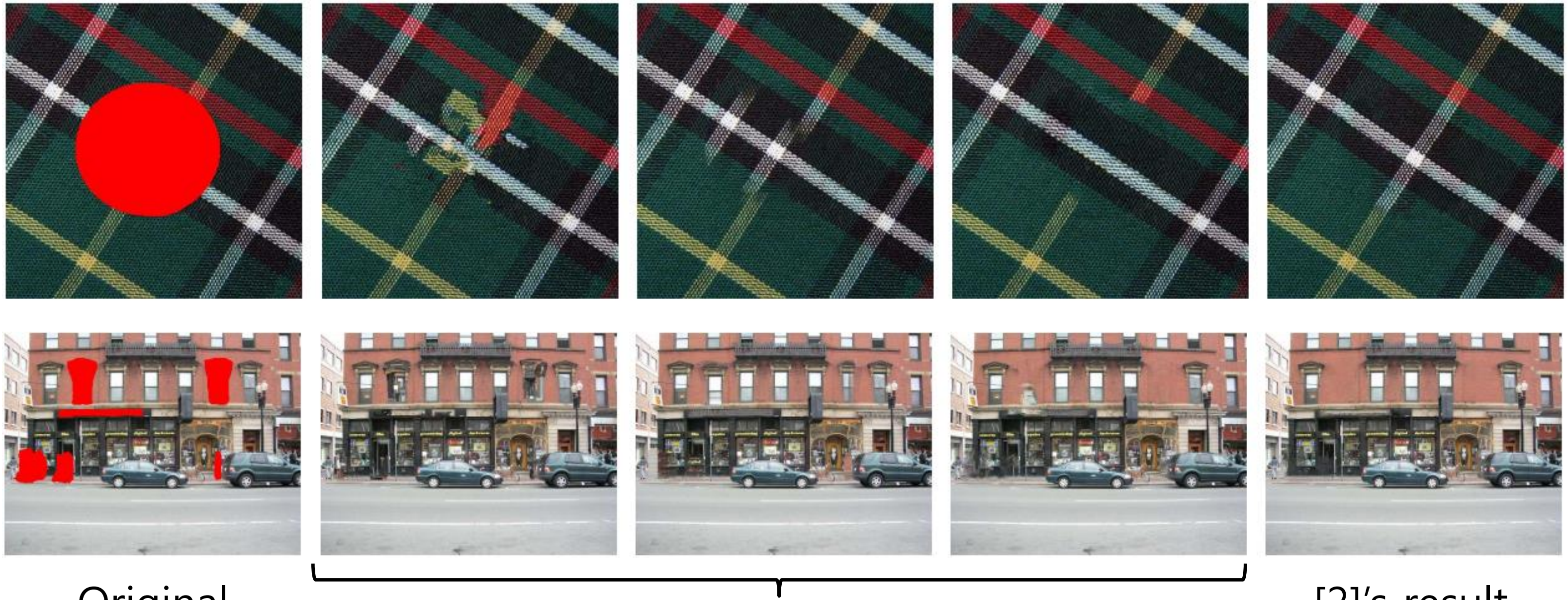


Blurring artifact

Examples of Inpainting (2/2)

- State-of-the-art results

  - Patch refinement in MRF framework [2]



Original input

Previous approaches

[2]'s result

2. M. Ghorai , S. Mandal, and B. Chanda, “A group-based image inpainting using patch refinement in MRF framework,” IEEE Transactions on Image Processing, vol. 27, no. 2, pp. 556-567, Feb. 2018.

Implementation Task (1/2)

▪ Make a your code for inpainting

- **Key header file : #include <opencv2/photo.hpp>**

void inpaint(InputArray src, InputArray inpaintMask, OutputArray dst, double inpaintRadius, int flags)

Parameters:
src : Input 8-bit 1-channel or 3-channel image.
inpaintMask : Inpainting mask, 8-bit 1-channel image.
(Non-zero pixels indicate the area that needs to be inpainted)
dst : Output image with the same size and type as src .
inpaintRadius : Radius of a circular neighborhood of each point inpainted.
flags : Inpainting method that could be one of the following:

INPAINT_NS Navier-Stokes based method.
INPAINT_TELEA Method by Alexandru Telea [Telea04].

Make your main function !

Implementation Task (2/2)

- **Inpaint the overlay text in a given scene**
  - **Try to make your own algorithm to remove the overlay text**

    ※ You can use color, corner information (and more…)

Summary

▪ Image inpainting

- **It is very useful for various applications**

  - Reconstruction from old and demaged images
  - Remove texts and logs in a given scene, etc.

- **Fast marching method based inpainiting**

  - PDE-based solution

- **Try to complete implementation**