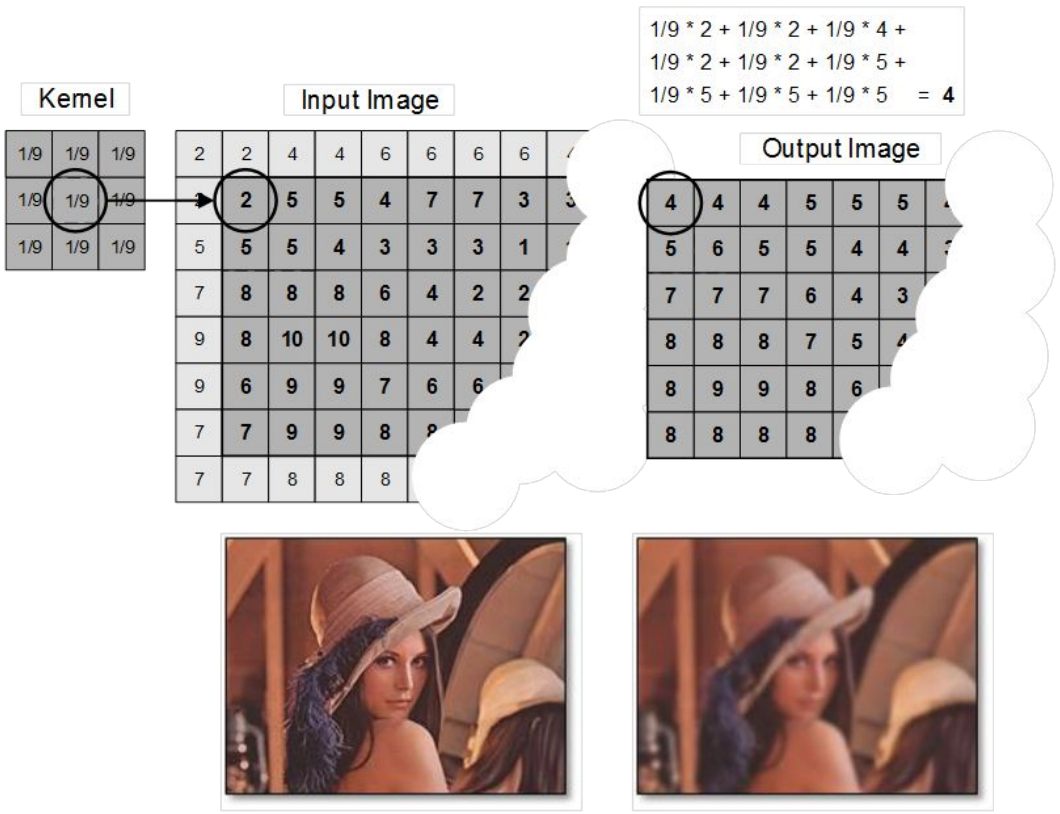
Computer Vision Winter-2017

Tutorial 1

Xichen Zhou

A little recap about filtering

This is filtering as well



A little recap about filtering

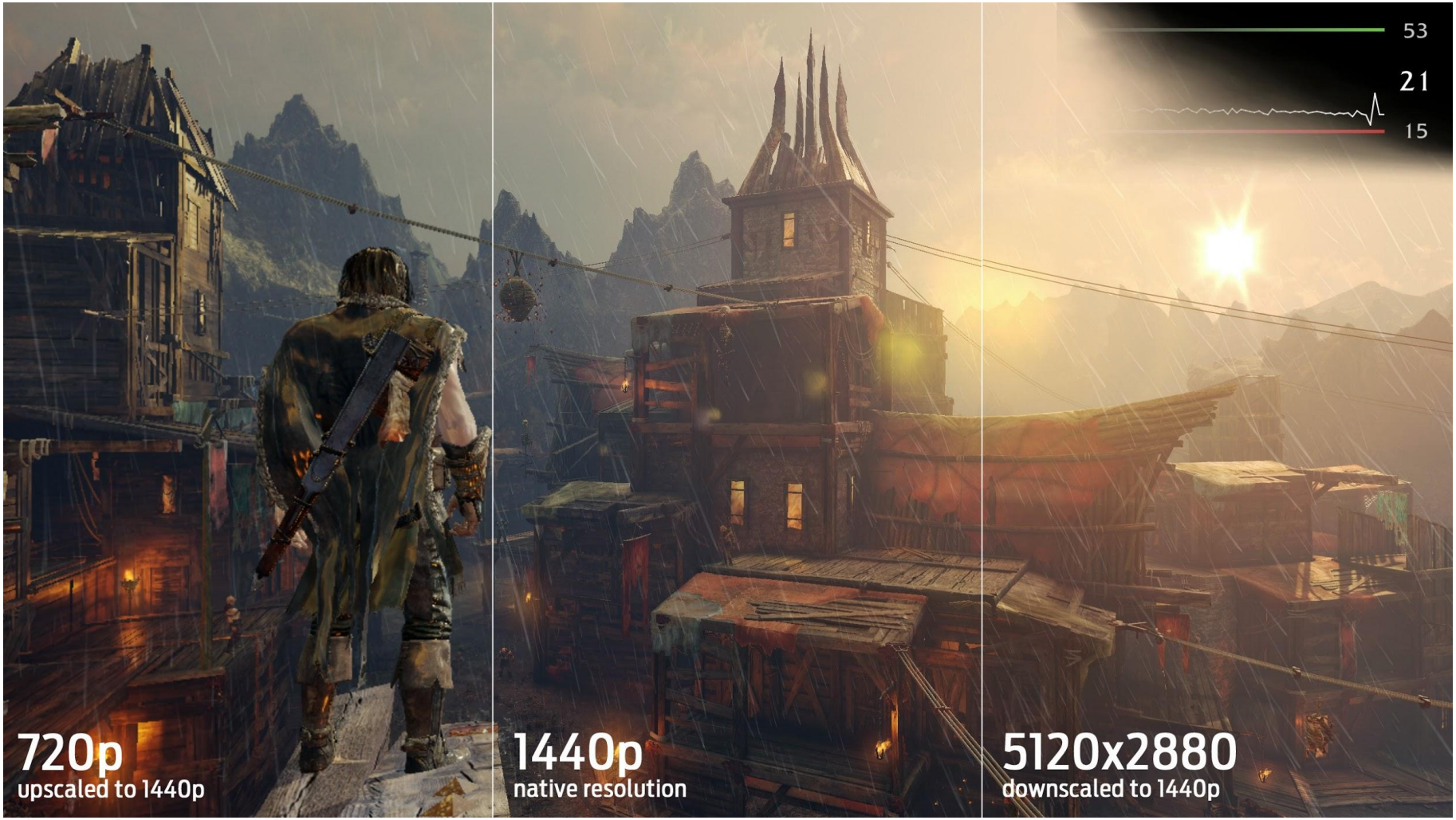
Basically you need follow three steps,

1) a iterating order(any order will do, so don't worry)
2) at each pixel, think about its relationship(mask) with others, the relationship should be the same for every pixel.
3) Convolution

Down-sampling/Up-sampling is filtering as well

In the case of down-sampling/up-sampling, you have **dest-image** and **src-image**
Only difference is:

1) you iterator through **dest-image** coordinates,
2) mask applies on **src-image**



But yes, the resolution is fake

More filtering

1) gaussian kernel:
   a)
2) laplacian kernel:
   a)
3) LoG kernel:
   a)
4) HoG kernel:

More filtering

1) gaussian kernel:
   a) $K(x:x',\sigma) = 1/(sqrt(2\pi)\sigma) * exp((x-x')^2/2\sigma^2)$
2) laplacian kernel:
   a)
3) LoG kernel:
   a)
4) HoG kernel:

More filtering

1) gaussian kernel:
   a) *K(x:x',σ) = 1/(sqrt(2π)σ) * exp((x-x')^2/2σ^2)*
2) laplacian kernel:
   a) $L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
3) LoG kernel:
   a)
4) DoG kernel:

More filtering

1) gaussian kernel:
   a) *K(x:x',σ) = 1/(sqrt(2π)σ) * exp((x-x')^2/2σ^2)*
2) laplacian kernel:
   a) $L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
3) LoG kernel:
   a) equations link
4) DoG kernel:

More filtering

1) gaussian kernel:
   a) *K(x:x',σ) = 1/(sqrt(2π)σ) * exp((x-x')^2/2σ^2)*
2) laplacian kernel:
   a) $L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
3) LoG kernel:
   a) equations link
4) DoG kernel:
   equations link

More filtering

1) gaussian kernel:
   a) *K(x:x',σ) = 1/(sqrt(2π)σ) * exp((x-x')^2/2σ^2)*
2) laplacian kernel:
   a) $L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
3) LoG kernel:
   a) equations link
4) DoG kernel:
   equations link

Which one is the right kernel

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

0	-1	0
-1	4	-1
0	-1	0

$$\begin{bmatrix} 0 & 0 & -1 & -1 & -1 & 0 & 0 \\ 0 & -1 & -3 & -3 & -3 & -1 & 0 \\ -1 & -3 & 0 & 7 & 0 & -3 & -1 \\ -1 & -3 & 7 & 24 & 7 & -3 & -1 \\ -1 & -3 & 0 & 7 & 0 & -3 & -1 \\ 0 & -1 & -3 & -3 & -3 & -1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 & 0 \end{bmatrix}$$

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

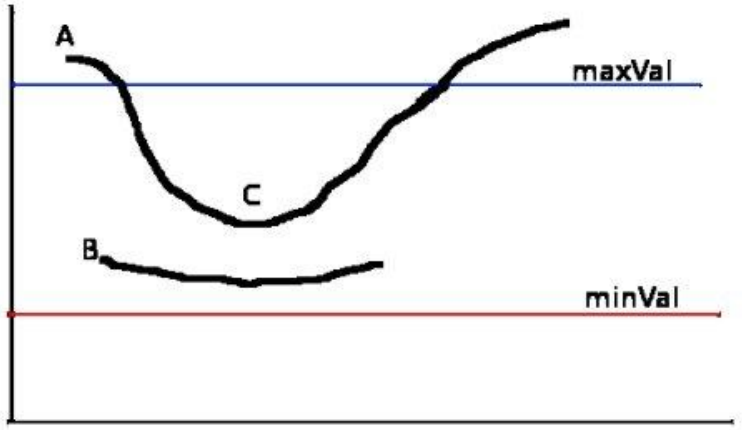
Canny Edge detections

//prototype
C++: void Canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)

canny is a algorithm with 4) steps

1) **Smooth** the image with a gaussian kernel? Why?
2) Apply **sobel filters** on the images to find the gradients.
3) Non-maximum suppression. For what reason?
4) **two** threshold filtering, **upper threshold**, **down-threshold**
   ?) What does the threshold do?

Edge detections



the two threshold

Edges

Let’s code with OpenCV

super-sampling:

filtering:

canny edge detector:

End