计算机视觉
Computer Vision

-- Image Processing 1

钟 凡
zhongfan@sdu.edu.cn

How to describe a pixel?

- …

How to describe a pixel?

- Spatial coordinate (x, y)
- Pixel color（RGB, YUV, …）

- (x, y, R, G, B)

Basic Image Processing

- Operates on spatial coordinates (x, y)
  - Geometric processing: ……
- Operates on pixel color（RGB, YUV, …）
  - Algebraic processing: ……

代数运算: 像素灰度变换

Gray Level Transformation (灰度变换)

- The most simple image processing task;
- Input an image, output the transformed image;
- Processing pixel-by-pixel, and do transform to the gray level of each pixel:
  - Intensity Adjustment (亮度调整)
  - Contrast Adjustment (对比度调整)
  - …

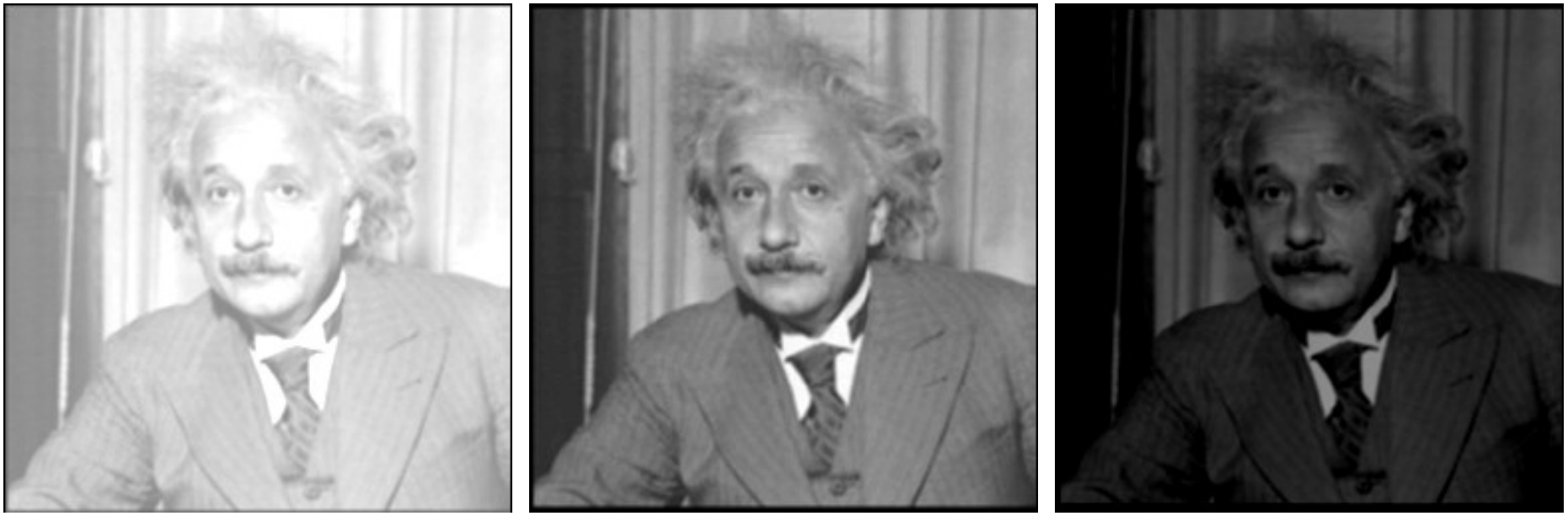
$$L' = T(L) \quad L, L' \in [0, 255]$$

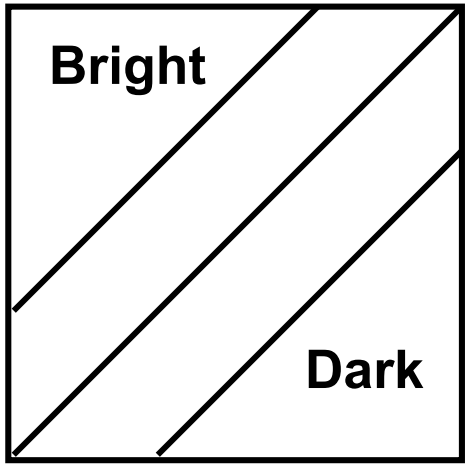
```
void scan_pixels(uchar *data, int width, int height, int step, int
nc)
{
    uchar *row=data;
    for(int yi=0; yi<height; ++yi, row+=step)
    {
        uchar *px=row;
        for(int xi=0; xi<width; ++xi, px+=nc)
        {
            // px now address the pixel  (xi, yi)
        }
    }
}
```

```
void gray_transform(uchar *data, int width, int height, int step,
…)
{
    uchar *row=data;
    for(int yi=0; yi<height; ++yi, row+=step)
    {
        uchar *px=row;
        for(int xi=0; xi<width; ++xi, px++)
        {
            uchar L=*px;
            *px= Transform (L);
        }
    }
}
```

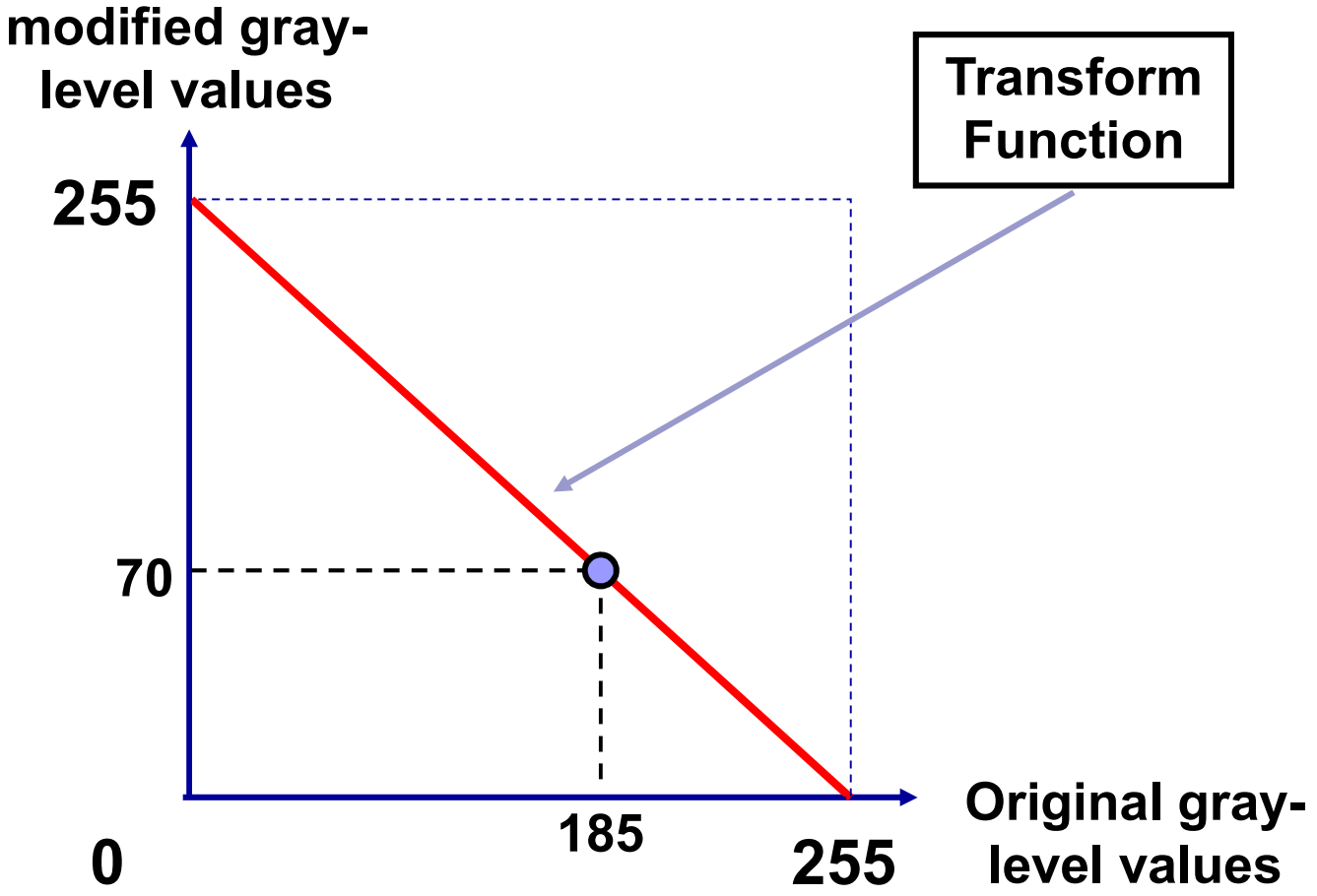
```
void gray_transform(uchar *data, int width, int height, int step, const uchar
T[256] )
{
    uchar *row=data;
    for(int yi=0; yi<height; ++yi, row+=step)
    {
        uchar *px=row;
        for(int xi=0; xi<width; ++xi, px++)
        {
            *px= T[ *px ];
        }
    }
}
```

Intensity





…?

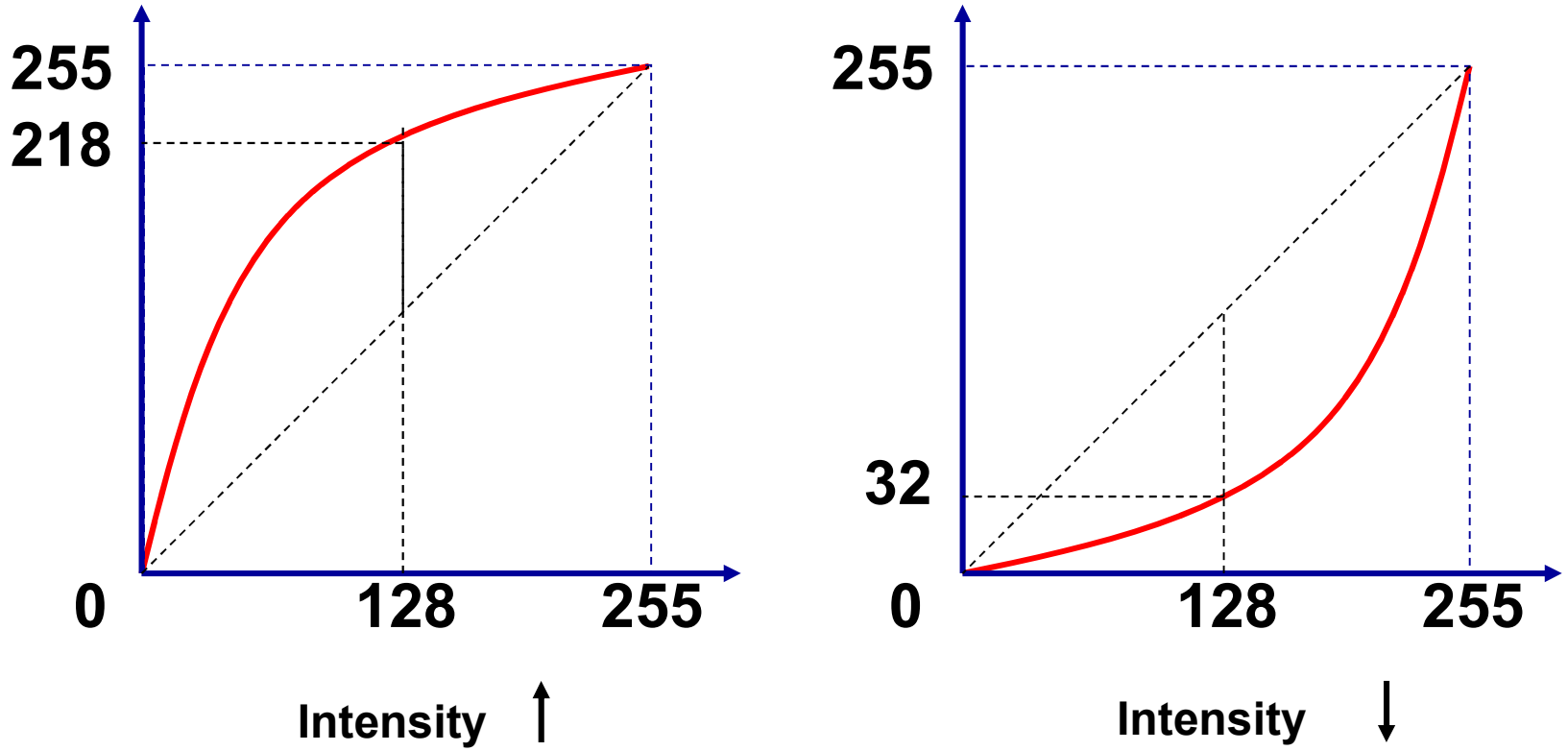

modified gray-level values
Transform Function
255
70
0
185
255
Original gray-level values


CELTICS
5


CELTICS
5

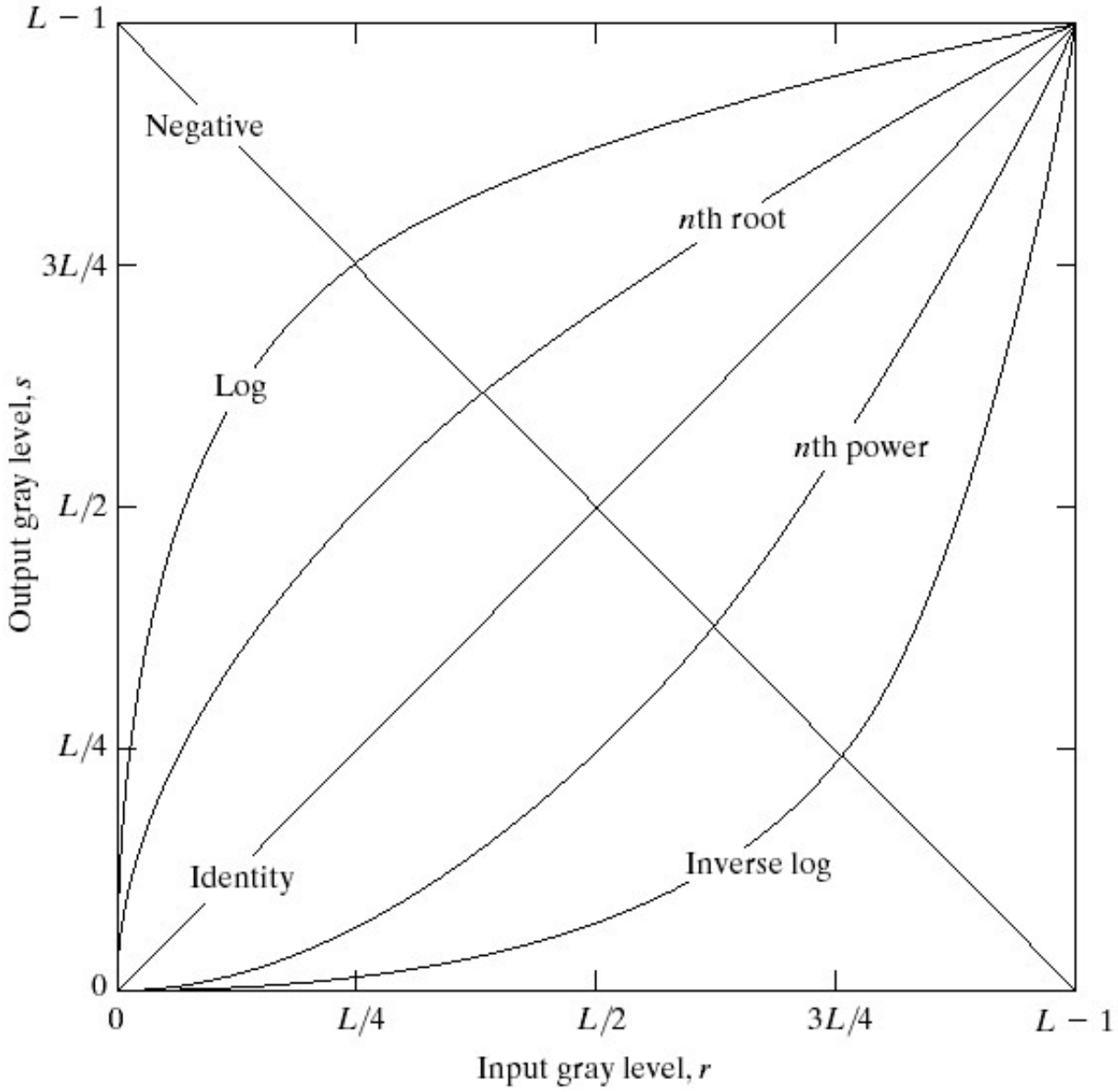
Intensity Adjustment: Other Functions

- Intensity adjustment by other functions



Transformation Functions

- **Log Transformations**
  - $s = c \log(1 + r)$
  - c: constant
- **Power-Law Transformations**
  - $s = cr^{\gamma}$
  - c,γ: positive constants
  - Gamma correction

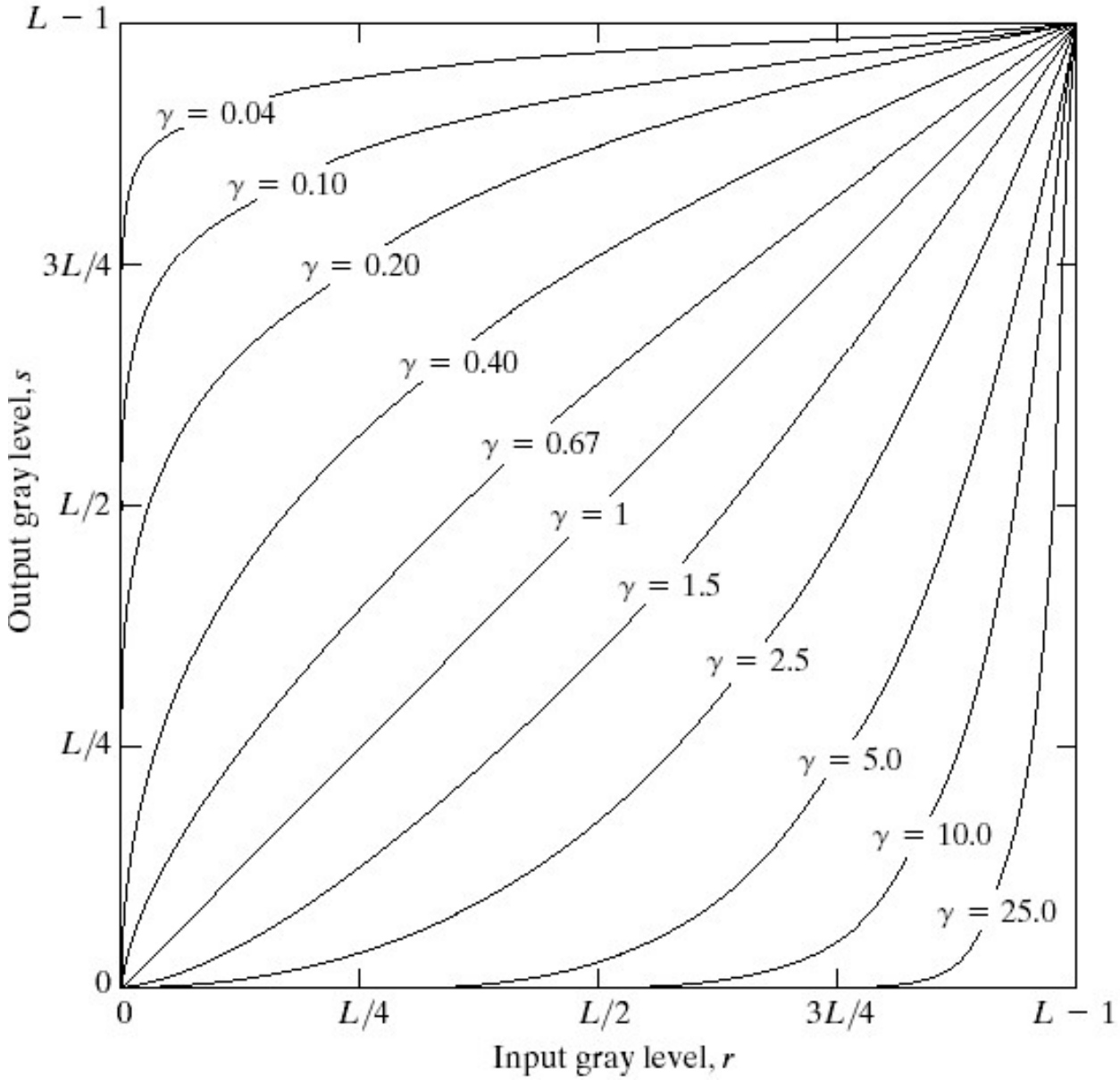


Some basic gray-level transformation functions used for image enhancement.

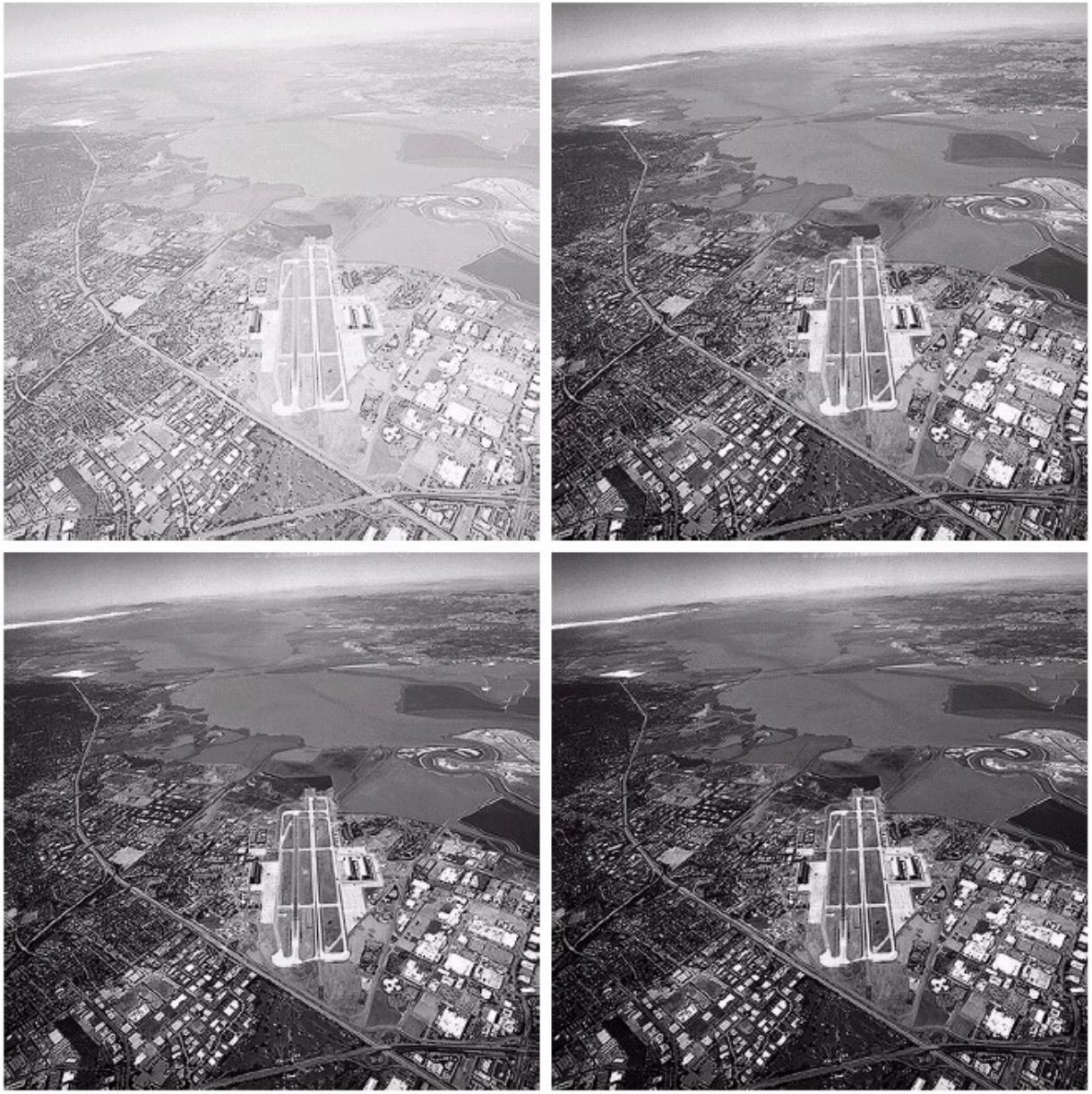
Linear: Negative, Identity

Logarithmic: Log, Inverse Log

Power-Law: *n*th power, *n*th root



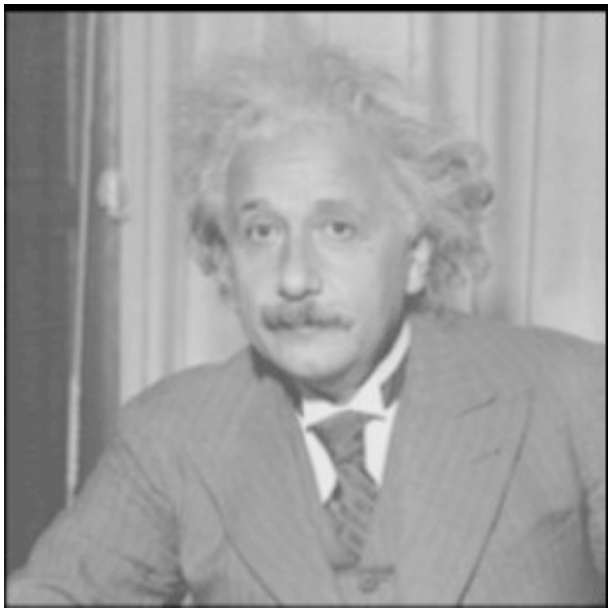
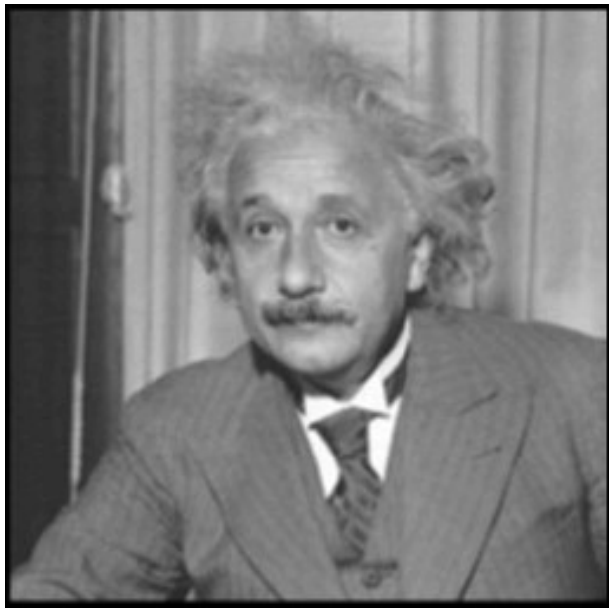
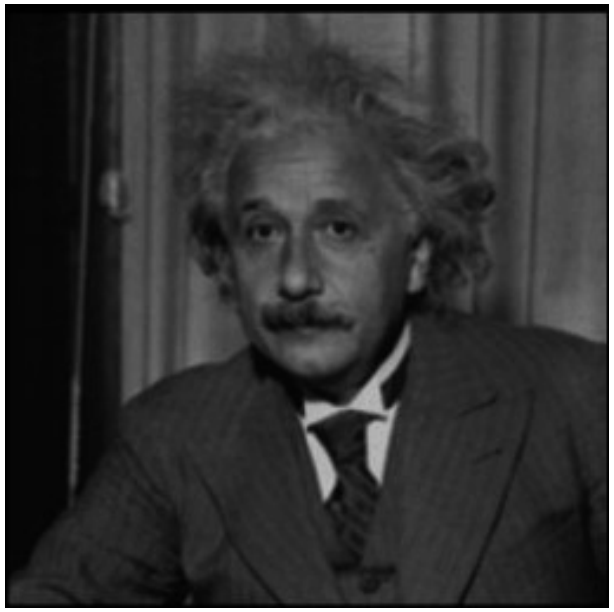
Plot of the equation $s = cr^{\gamma}$ for various of γ (c = 1 in all cases).

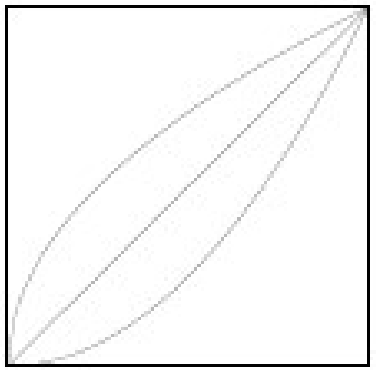


(a) Aerial image. (b) - (d) Results of applying the power-law transformation with $c = 1$ and $\gamma = 3.0$, 4.0, and 5.0, respectively. (Original image for this example courtesy of NASA.)

a	b
c	d

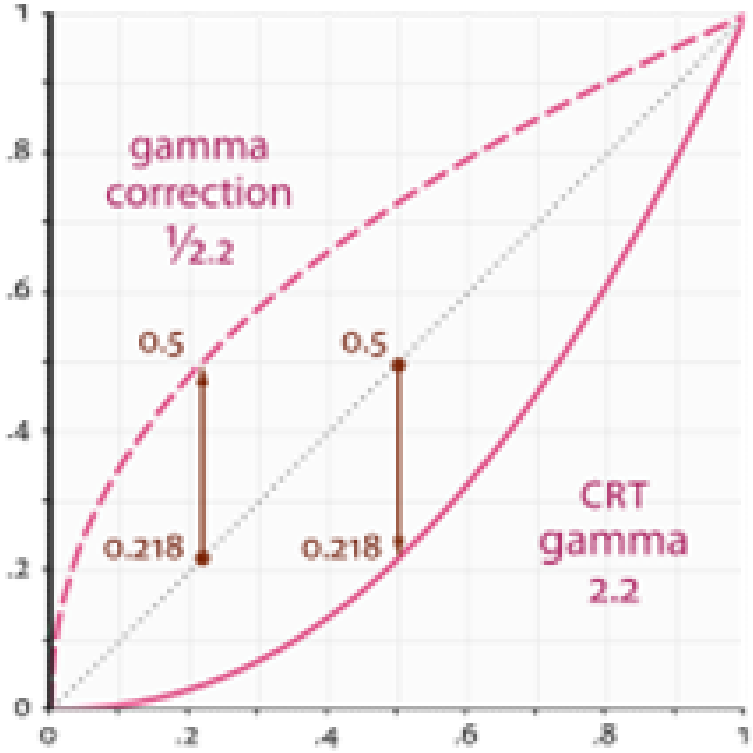
Gamma Transformation



$$s = r^{\gamma}$$

Gamma Correction




γ=1 (original)

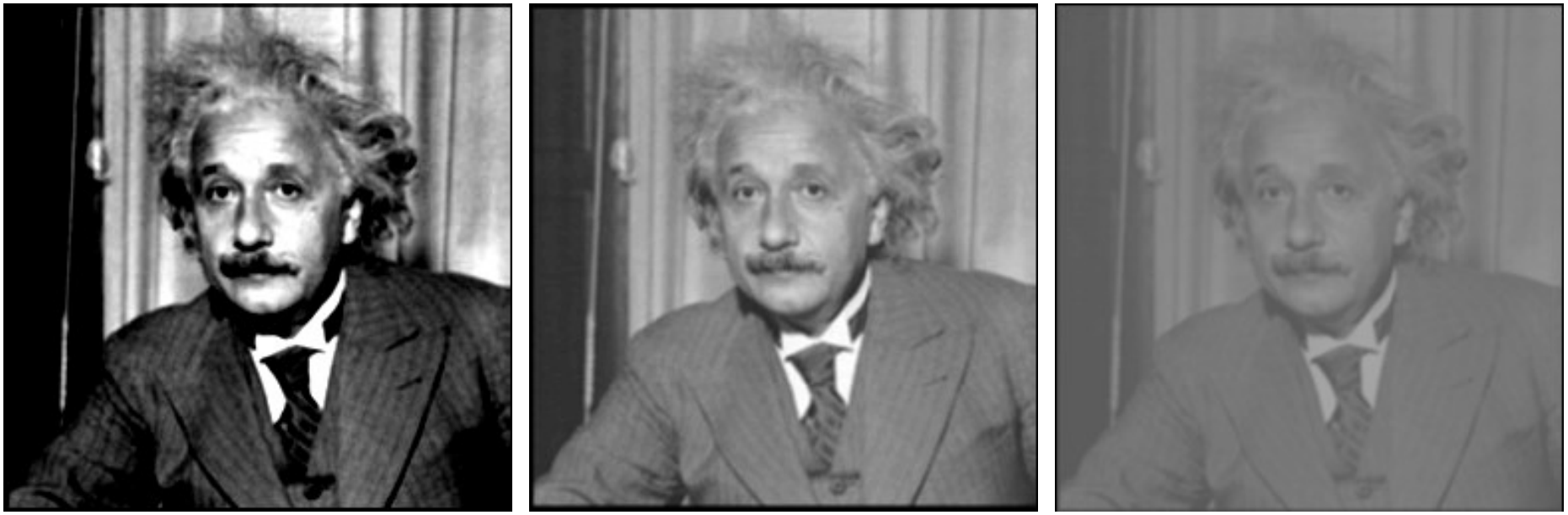

γ=2

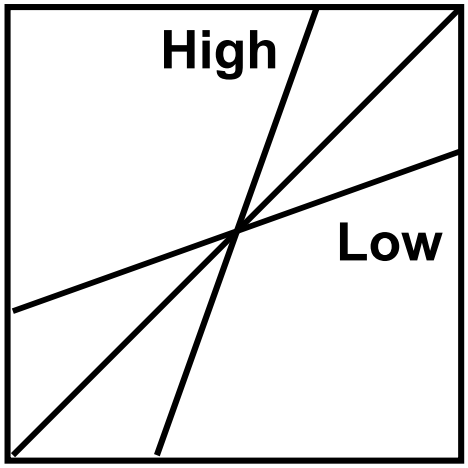

γ=1/2


γ=1/3

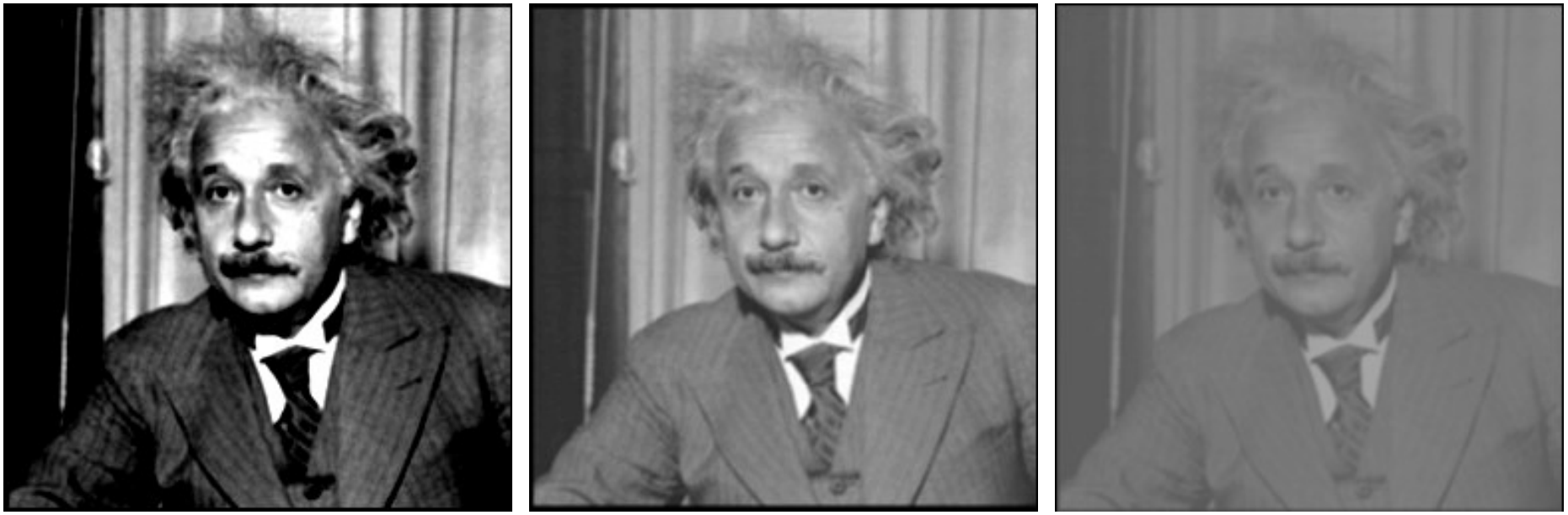

γ=1/4

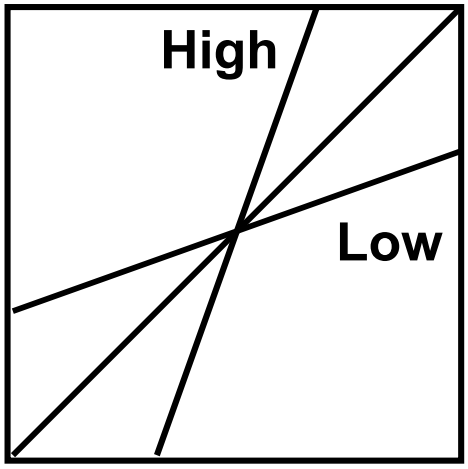
Contrast Adjustment





Contrast Adjustment

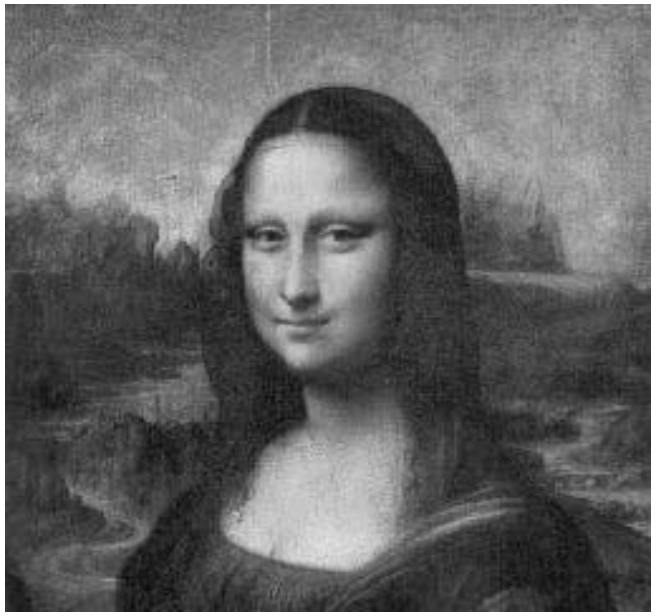




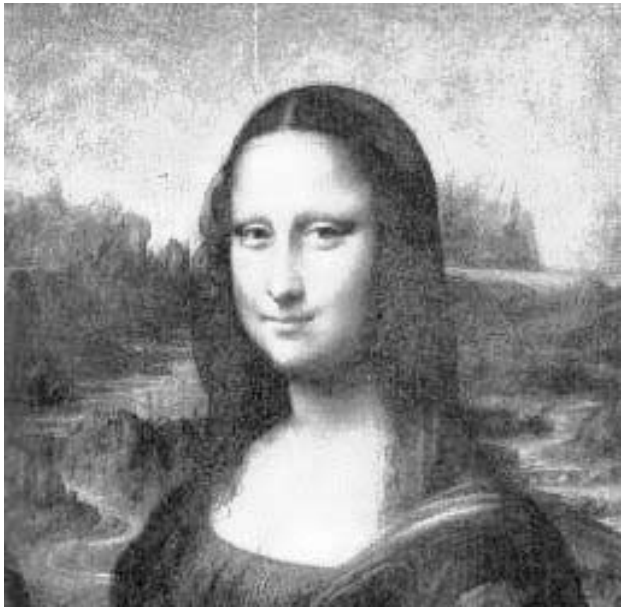
亮度调整：整体变亮或变暗

对比度调整：亮的更亮，暗的更暗

Contrast Adjustment



Original Mona Lisa Image

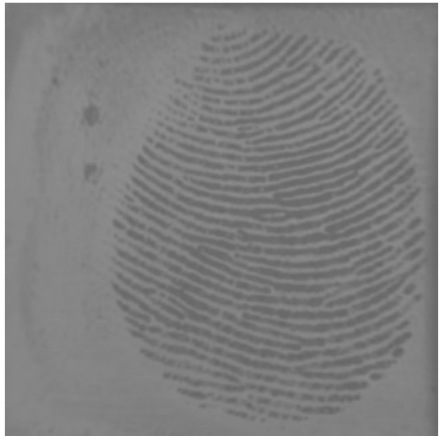


Contrast Stretched Image

Contrast Adjustment

Original Image

Contrast ↑



Contrast ↓

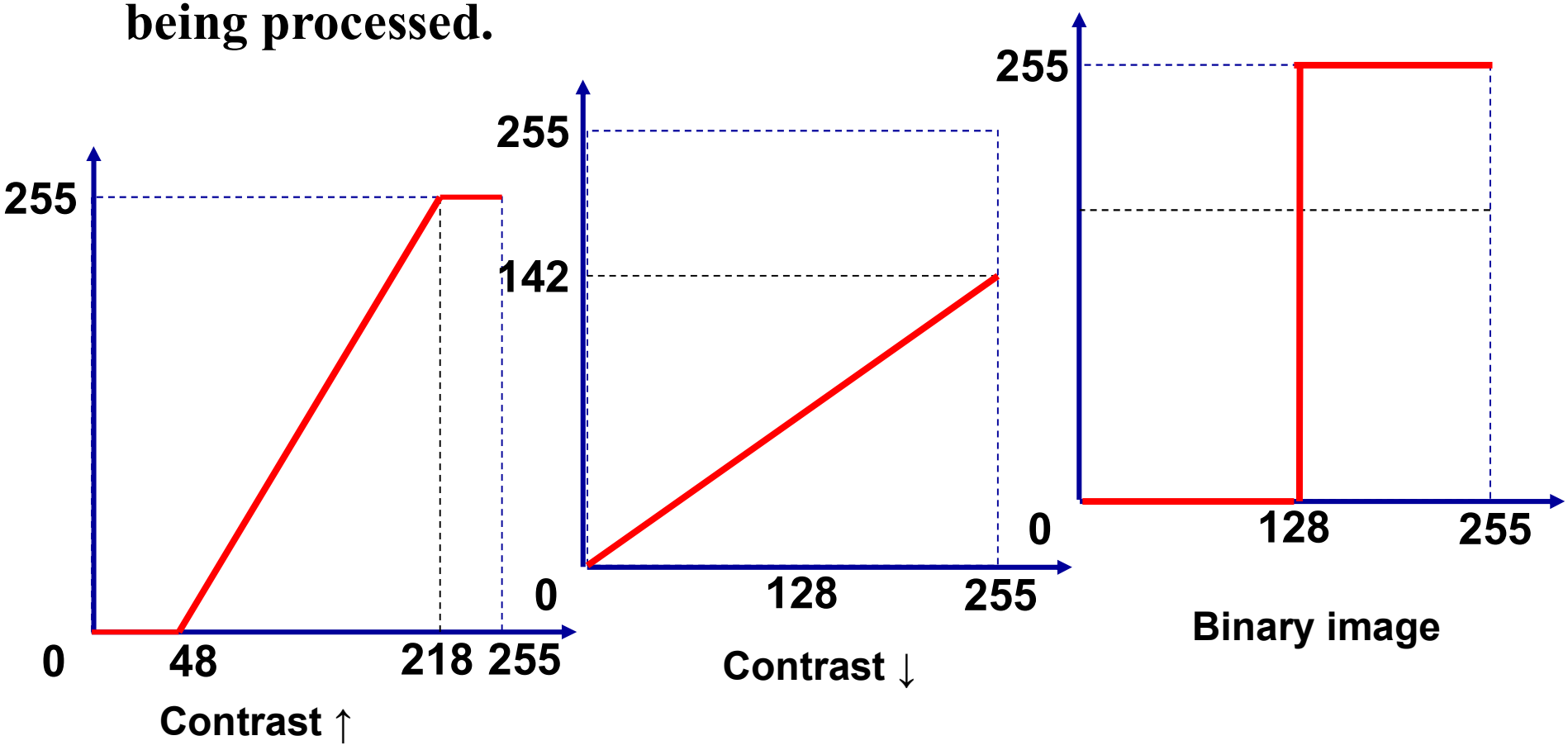
Original Image

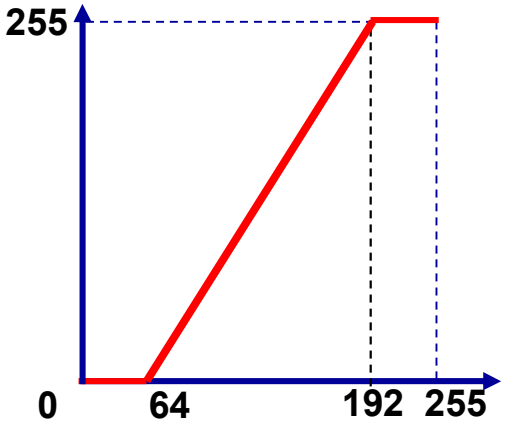
Contrast ↑

Contrast ↓

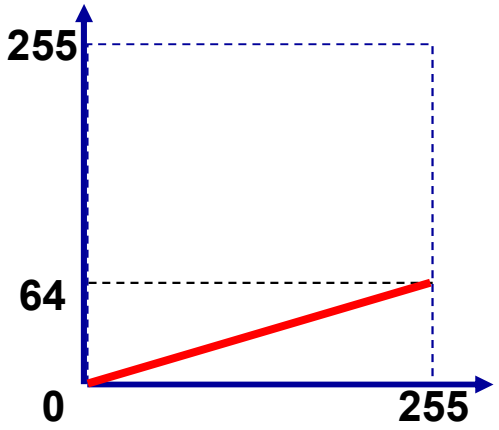
Contrast Adjustment

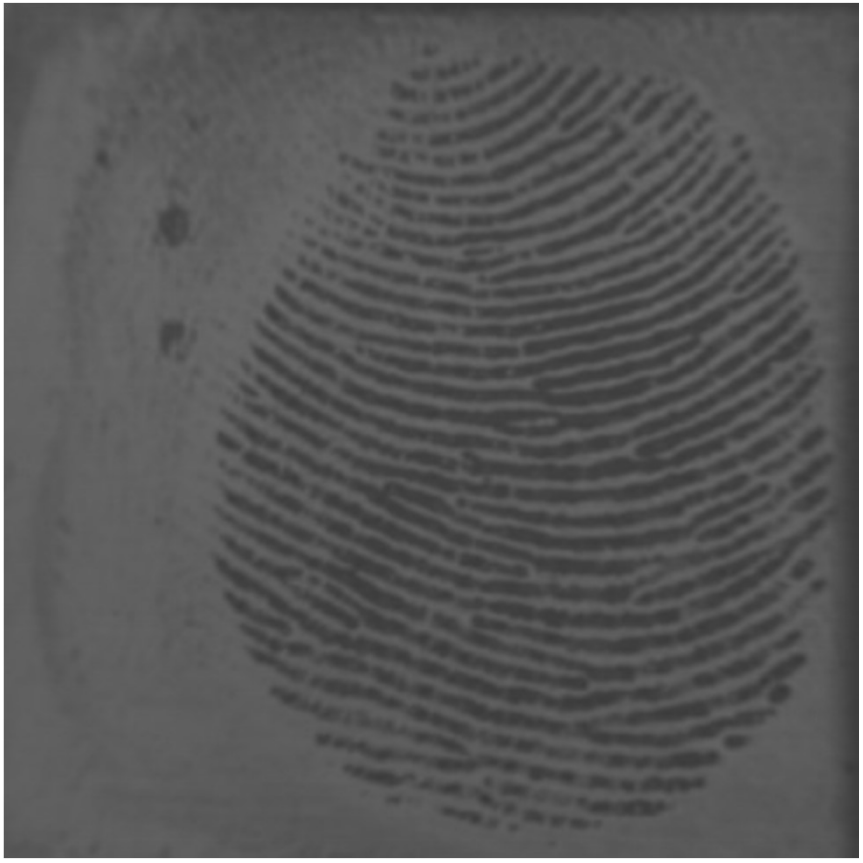
- To increase the dynamic range of the gray levels in the image being processed.



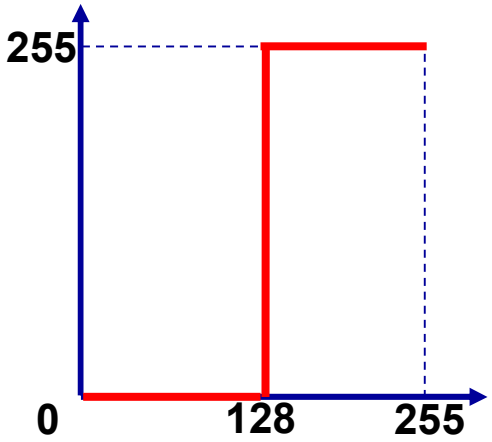


Contrast ↑

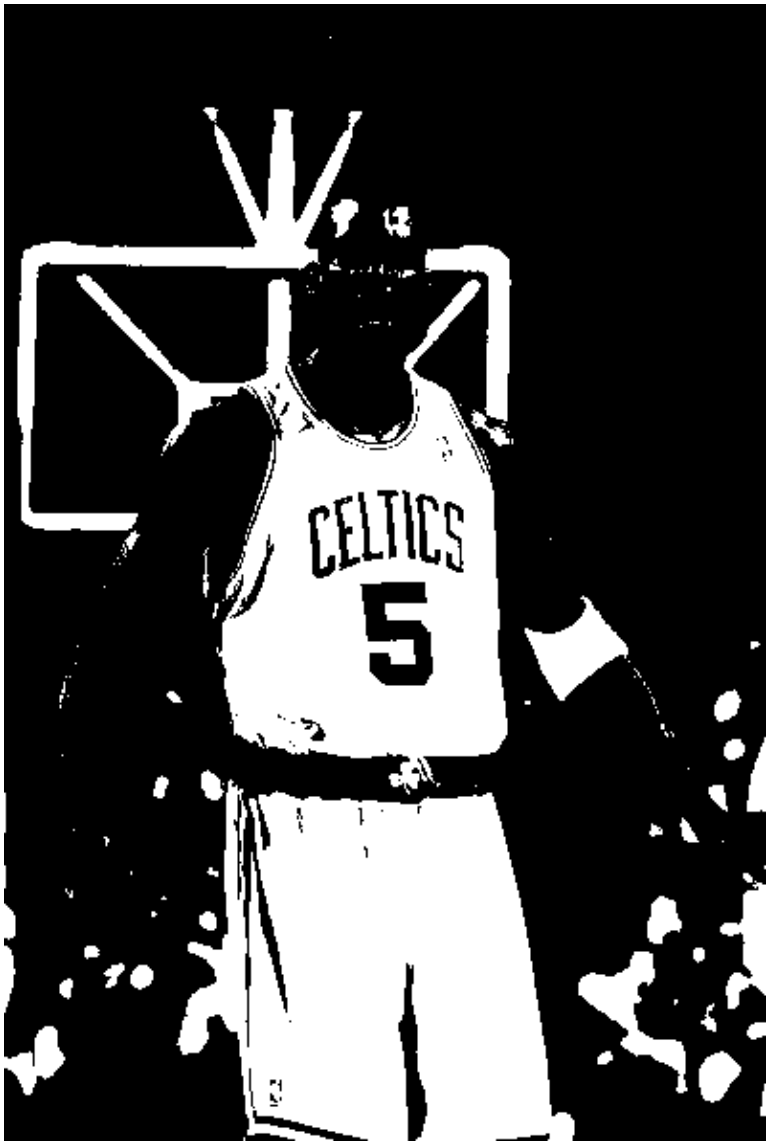


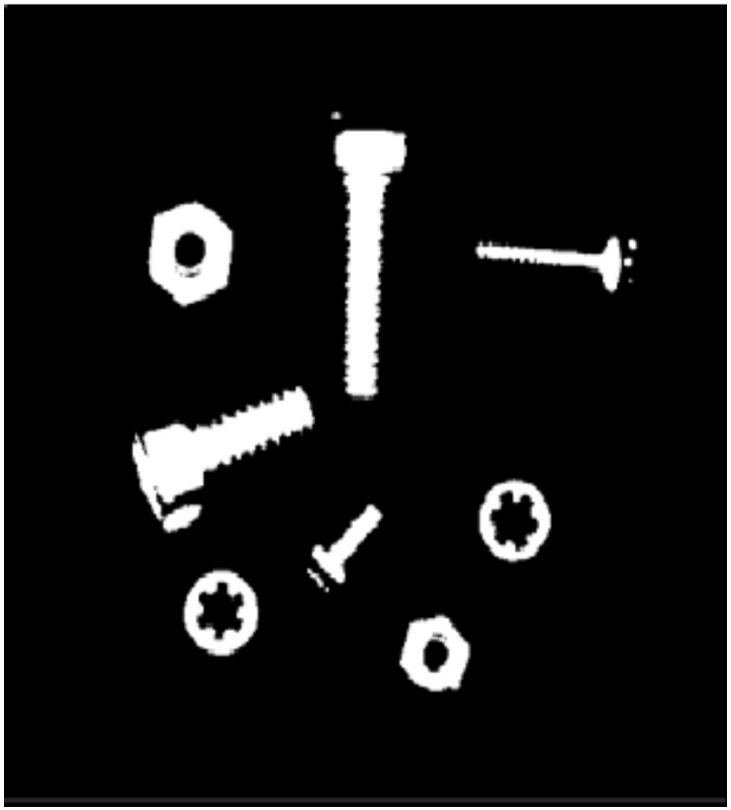


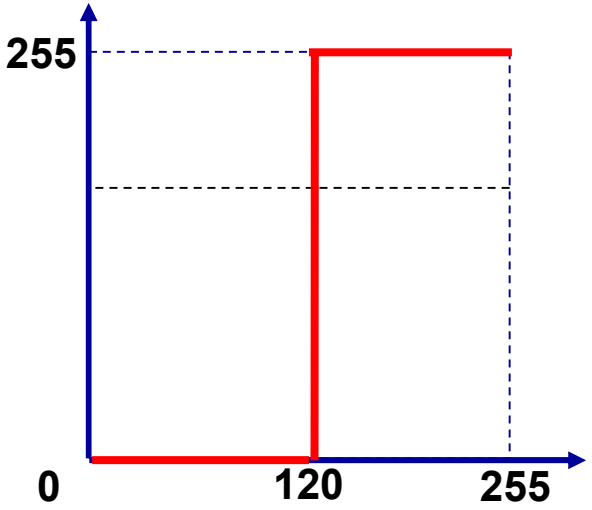
Contrast ↓


255
0
128
255

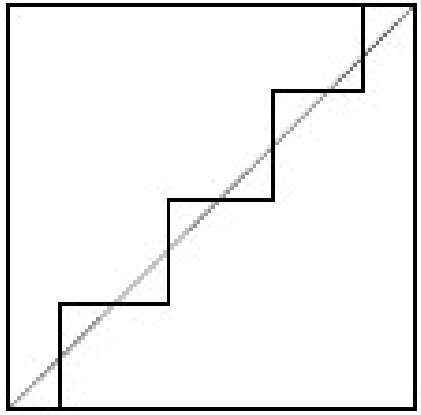

CELTICS
5


CELTICS
5

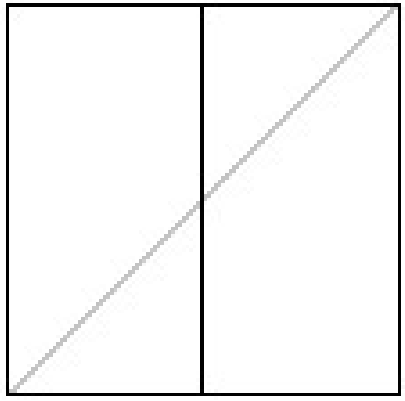



255
0
120
255

Quantize & Threshold



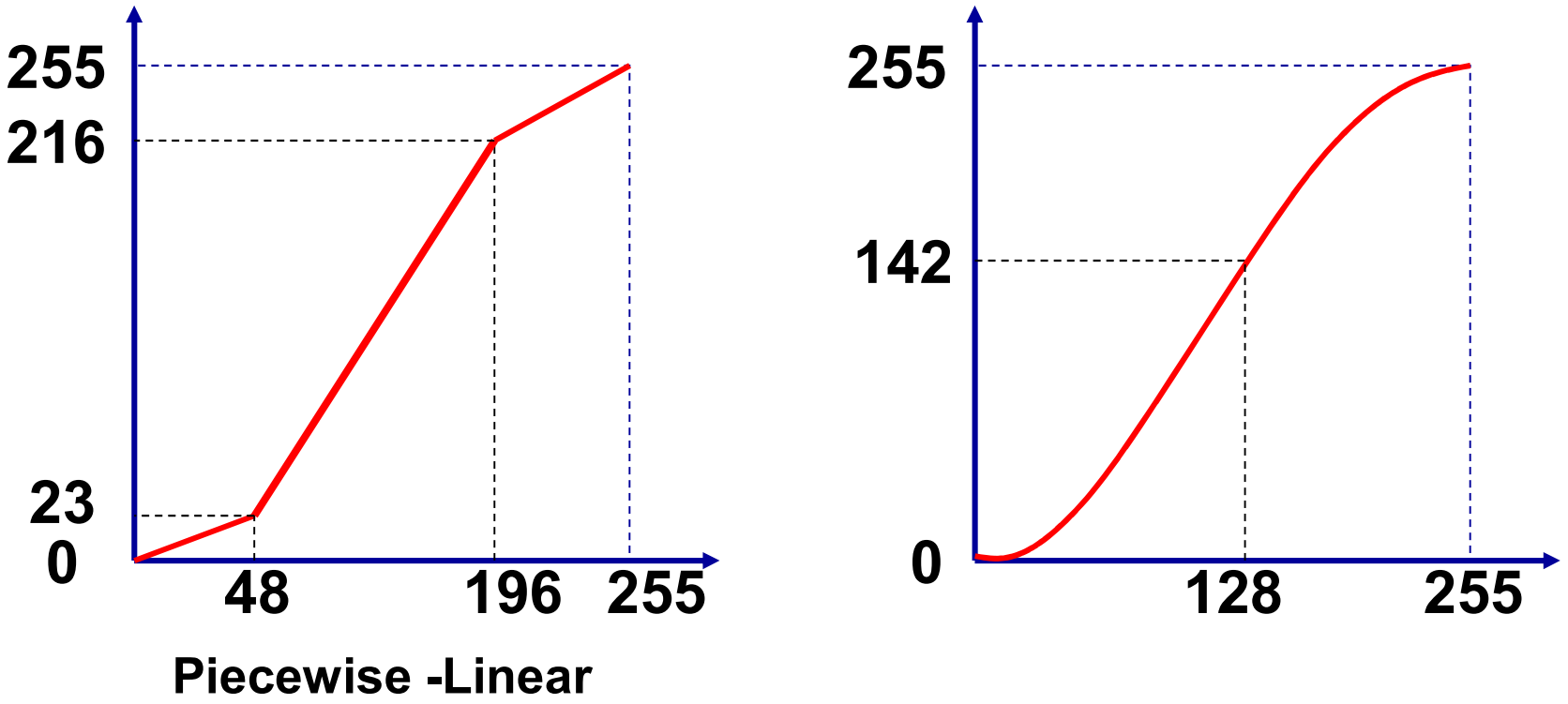
Quantize



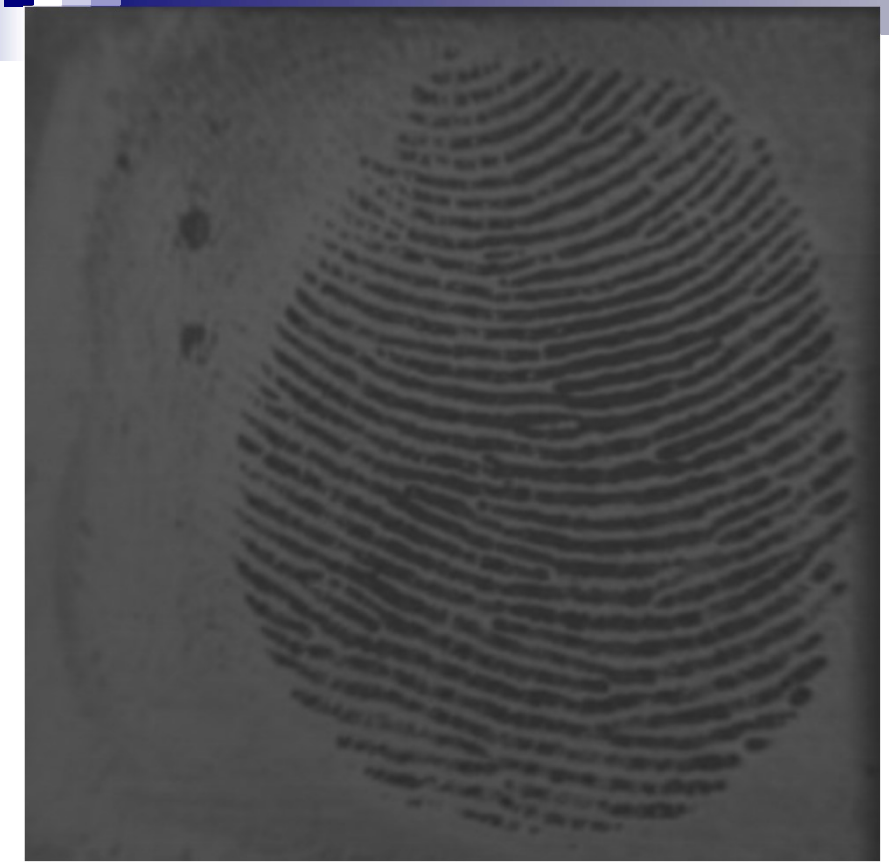
Threshold

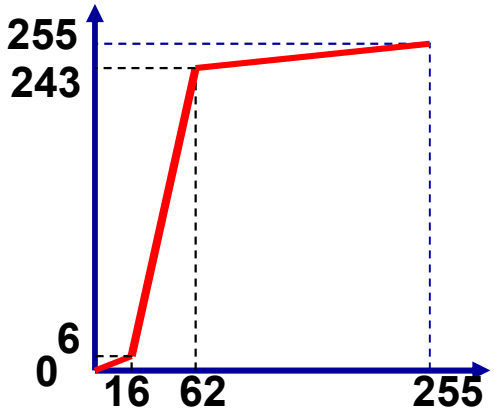
Contrast Adjustment

- **Contrast Stretching**
  - **Piecewise**

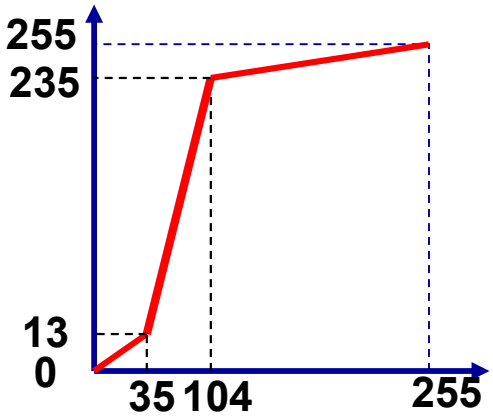


Piecewise -Linear

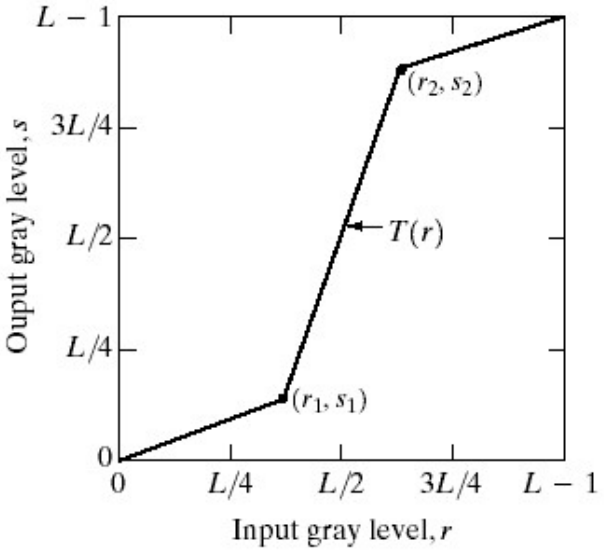


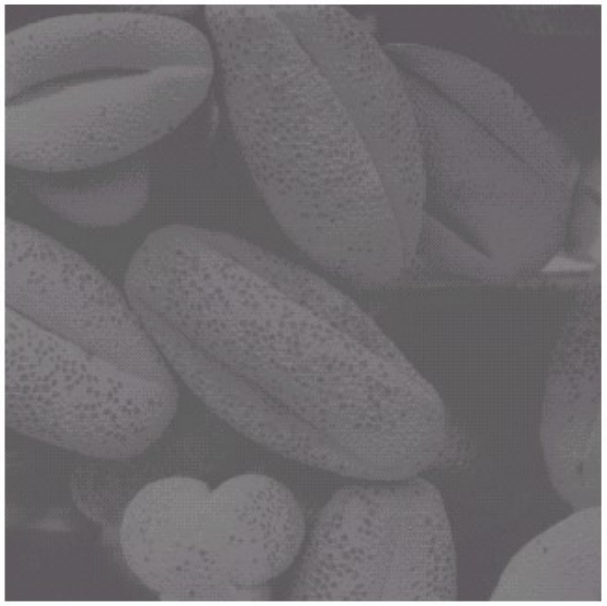

255
243
6
0
16
62
255


CELTICS
5


255
235
13
0
35
104
255


CELTICS
5

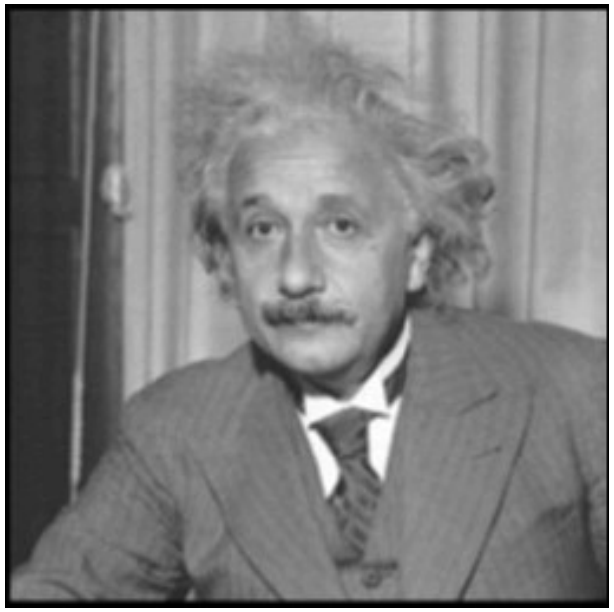


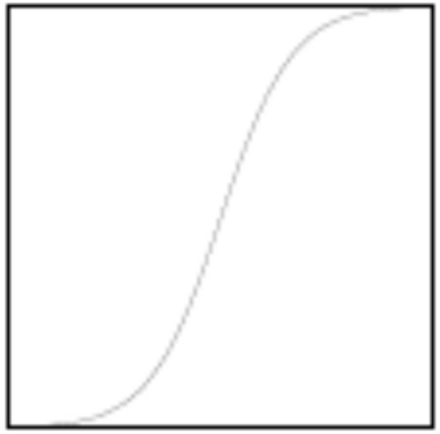


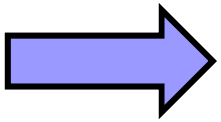
Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia)

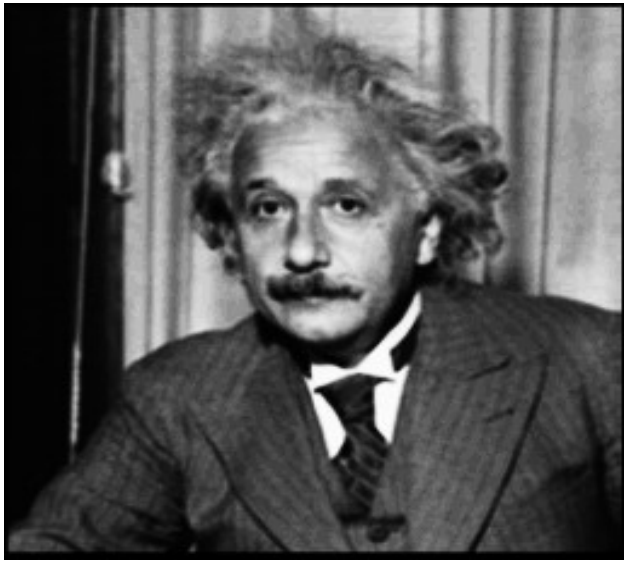
a	b
c	d

Contrast: Sigmoid









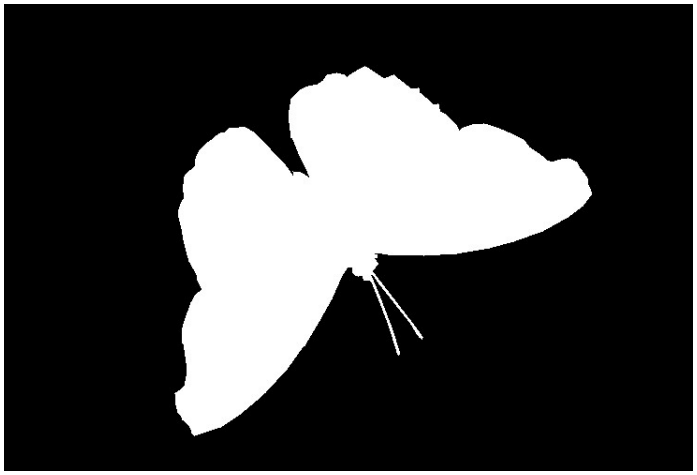
代数运算: 多幅图像

Algebraic Processing of Multiple Images

- 输入为多幅大小相同的图像
- 输出图像与输入图像大小相同
- 基于对应像素进行计算

$$O(x,y) = f[I^1(x,y), I^2(x,y), ..., I^N(x,y)]$$

X



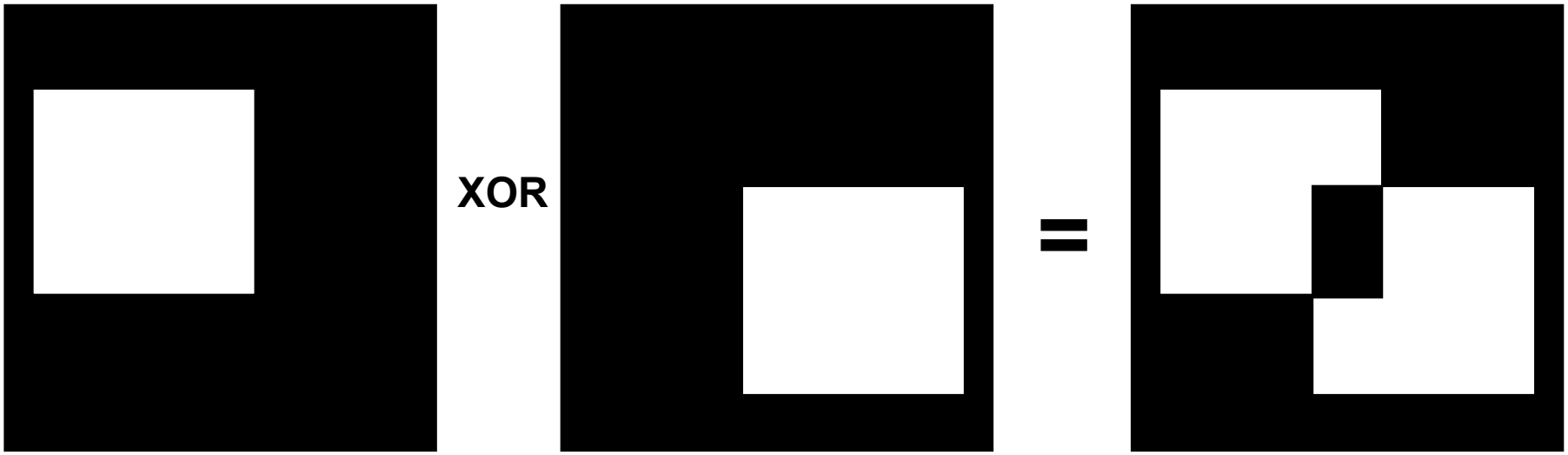
*

=

$$C(x,y) = A(x,y) * B(x,y)$$

Used in color image

XOR

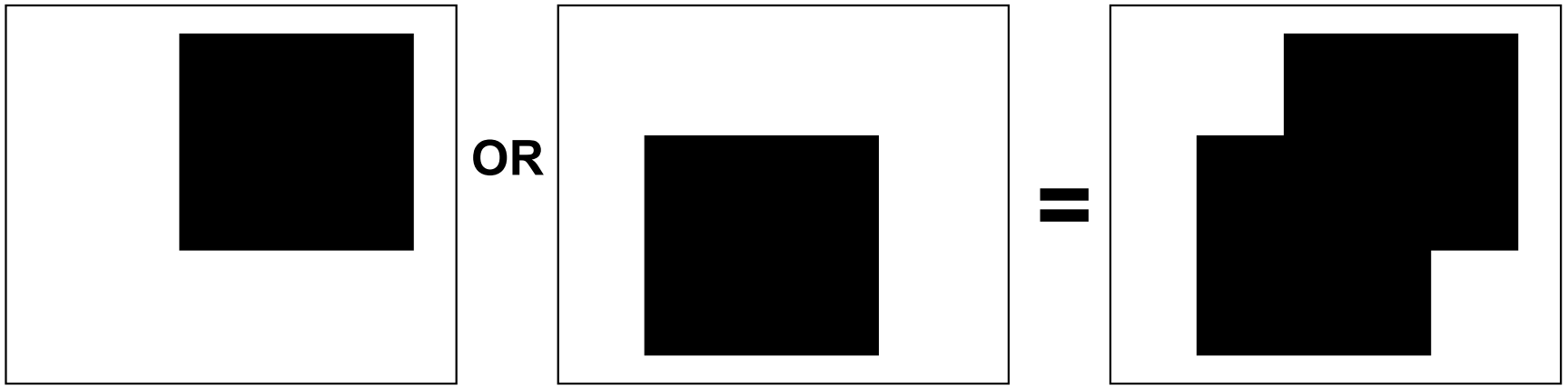


g(x,y) = f(x,y) XOR h(x,y)

Black→0 **White→1**

OR

$$g(x,y) = f(x,y) \text{ OR } h(x,y)$$

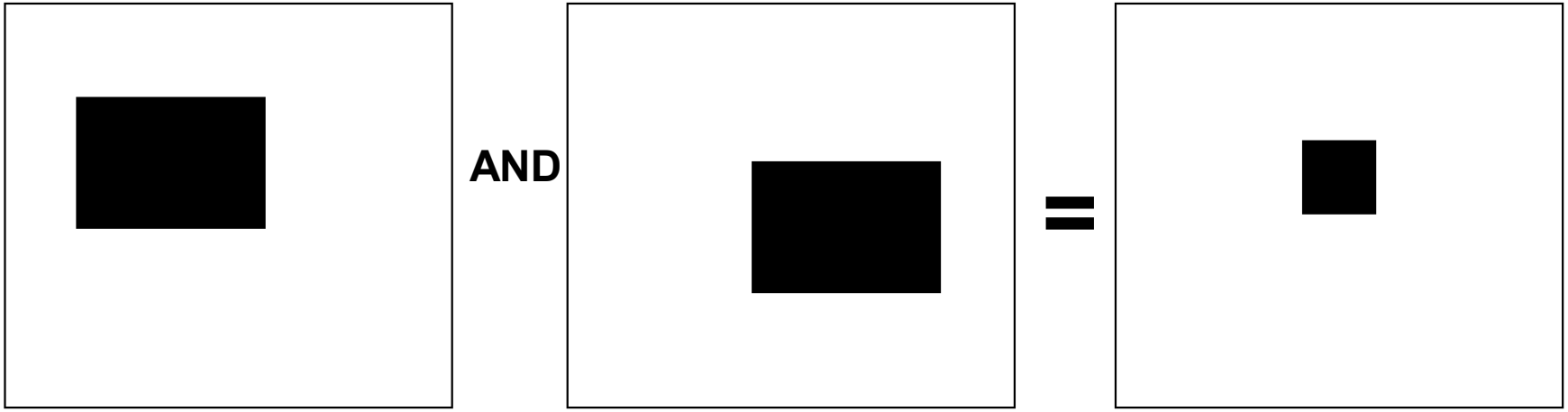


Black→1

White→0

AND

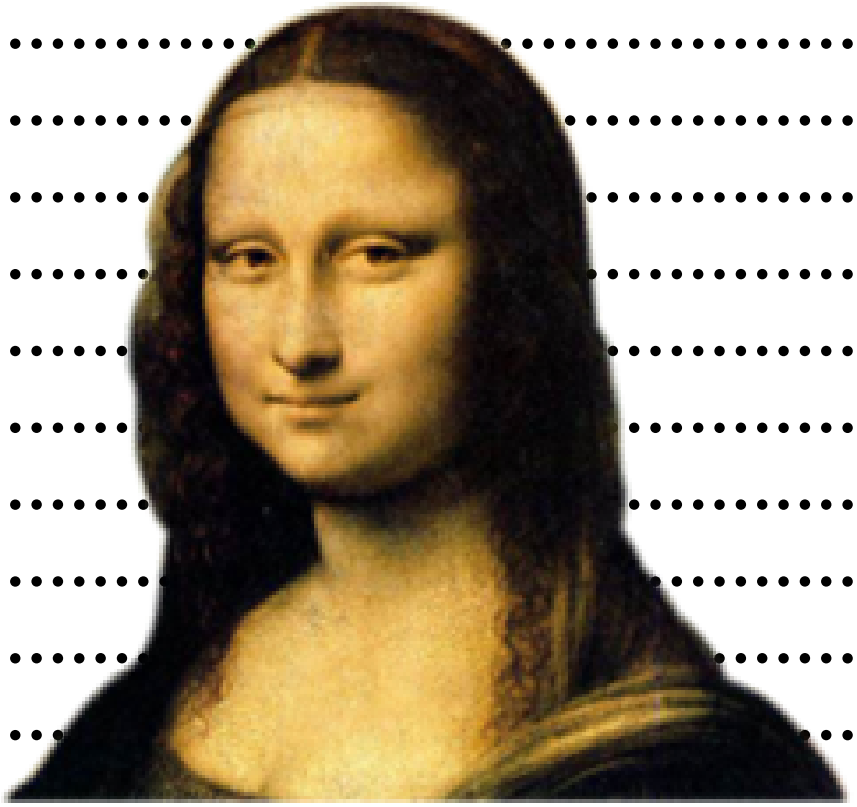
$$g(x,y) = f(x,y) \text{ AND } h(x,y)$$



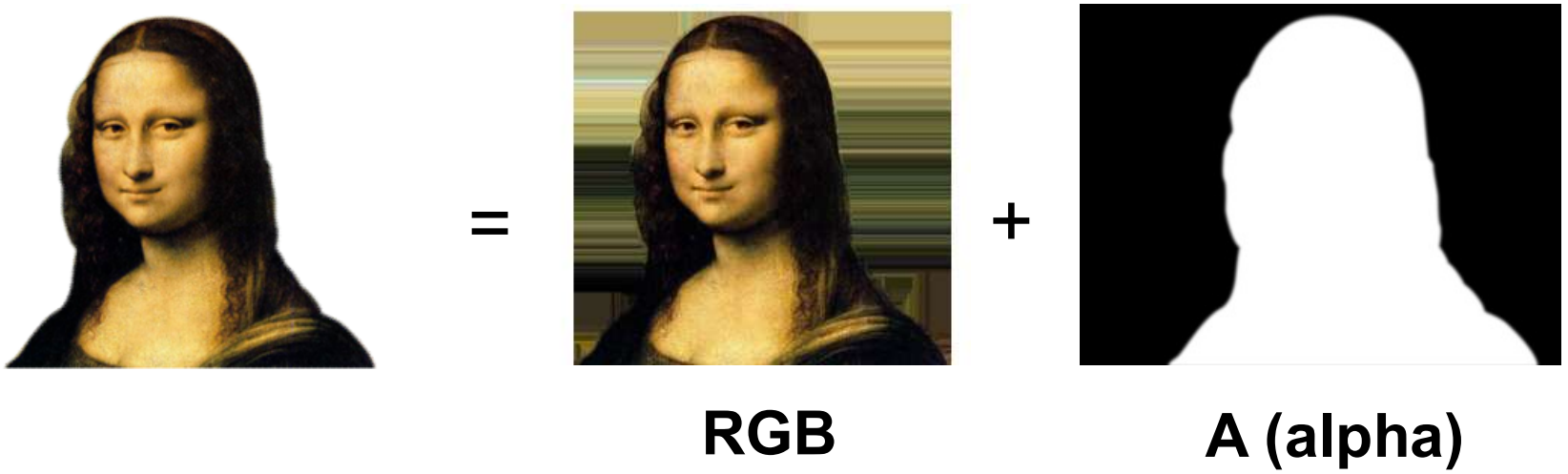
Black→1

White→0

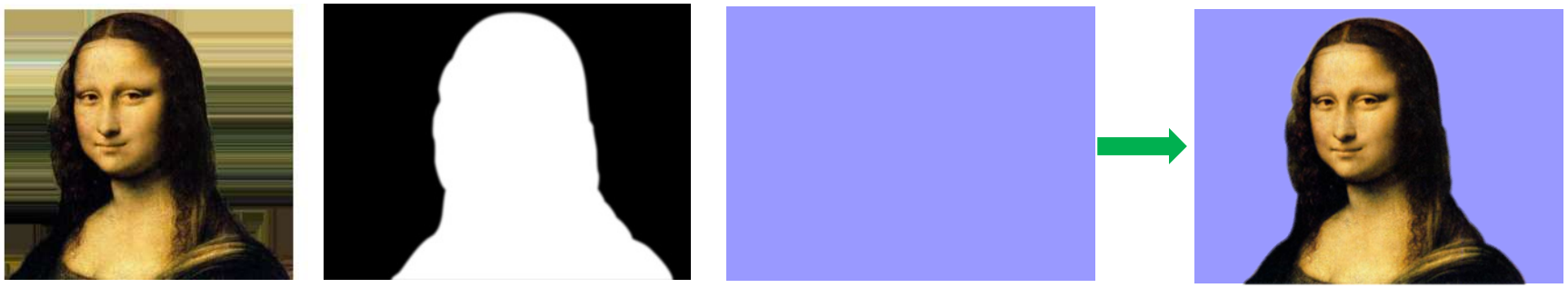
Alpha Blending

- 

Alpha Blending



Alpha Blending



F α B C

Image Composite (合成)： $C = \alpha F + (1-\alpha)B$

合成的逆？

- **Given C, to solve alpha, F, B ?**

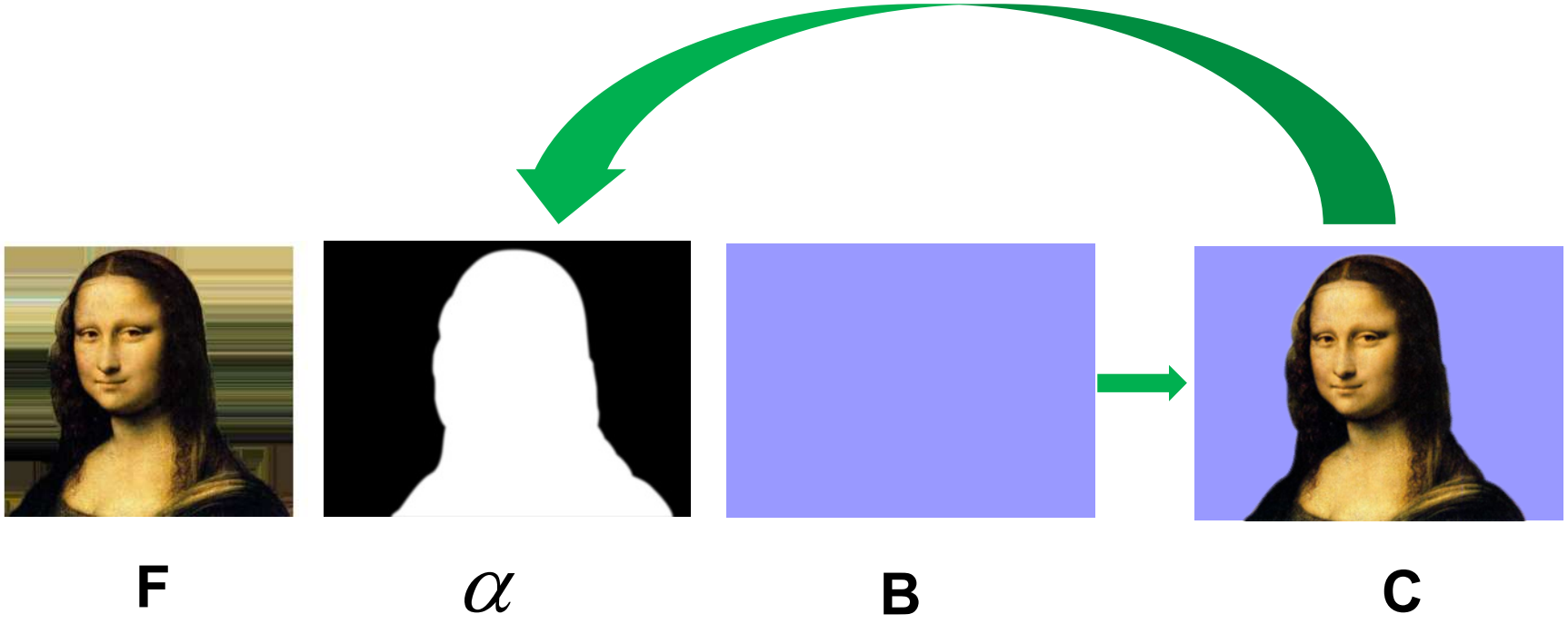
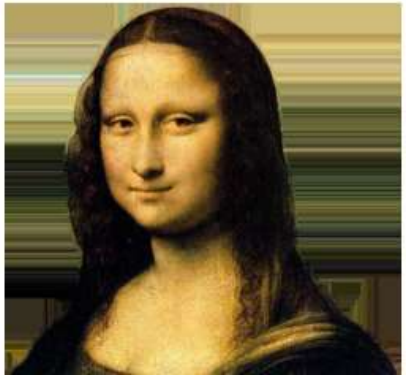


Image Matting (masking, 抠图)

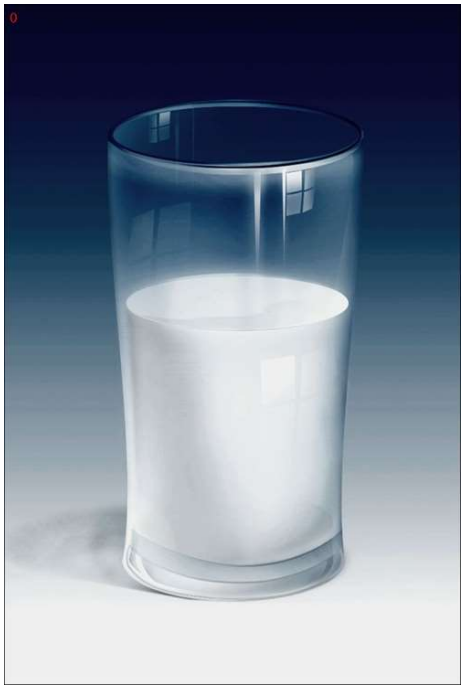
- **Extracting specific object or region from image (get the alpha map)**



C α

Image Matting

- Need to handle semi-transparent regions and hairs etc.



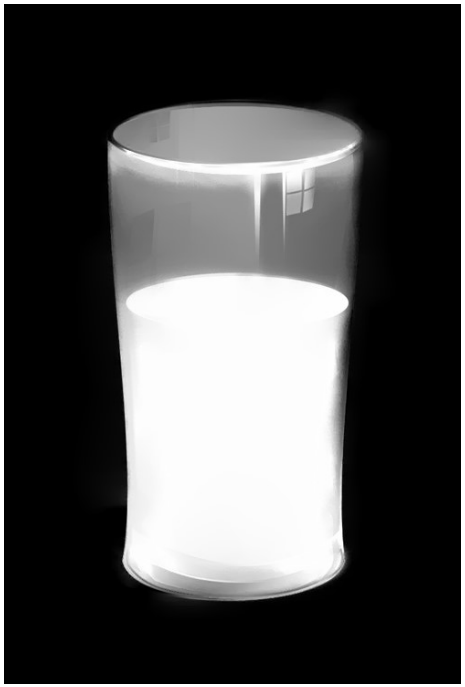
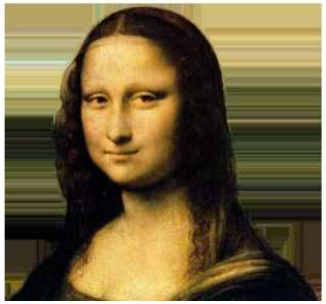


Image Matting

- Need to handle semi-transparent regions and hairs etc.

Need to get F, B?



抠图

F, B ?

C α

合成

F α B C’

Use C as F ?

Get F by foreground restoration (前景恢复)

源图

以C为F

正确的F

Background Color B?

- ….

背景相减（Background Subtraction)

当前帧 *I*

-

背景 *B*

?

T= 500

T= 1000

T= 2000

?

T= 4000

$$I(x,y) = (r,g,b)$$
$$B(x,y) = (r,g,b)$$

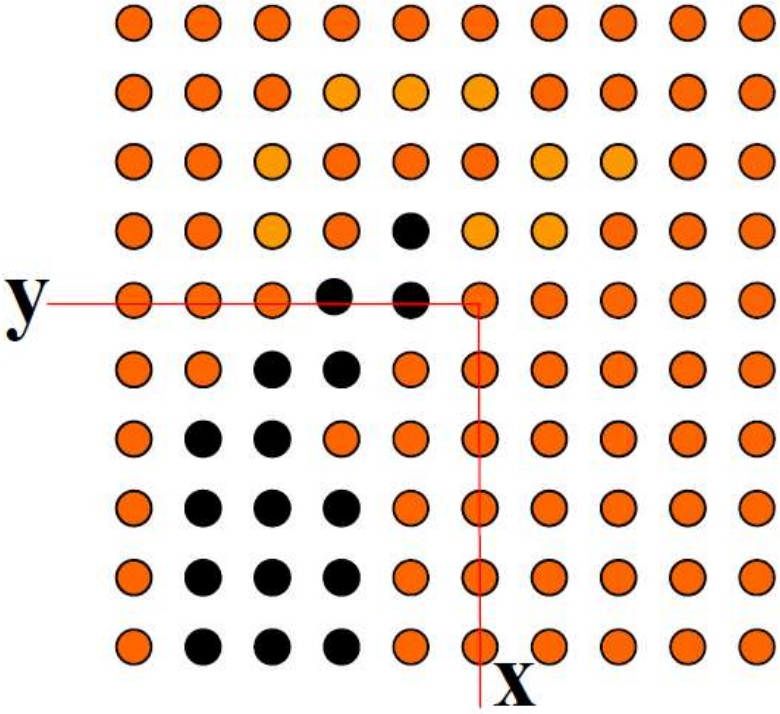
$$Diff(x,y) = \| I(x,y) - B(x,y) \|^2$$

$$Is\ foreground\ \ if\ \ Diff(x,y) > T$$ (a given threshold)

基本几何处理

The Geometry of Image

- A 2D array of points (pixels)



Basic geometric processing

- 水平&垂直翻转 (flip)
- 缩放 (resize / zoom in /zoom out /scale)
- 旋转 (rotation)
- ……
- 仿射变换 (Affine Transform)
- 透视变换 (Perspective Transform)
- ……
- 图像变形 (Image Warping)

Flip

Horizontal Flip

Vertical Flip

实现？

```
void vflip(const void *in, int width, int height, int istep,  int pix_size, void *out, int
    ostep)
{//vertical
}
```

```
void hflip(const void *in, int width, int height, int istep,  int pix_size, void *out, int
    ostep)
{//horizontal
}
```

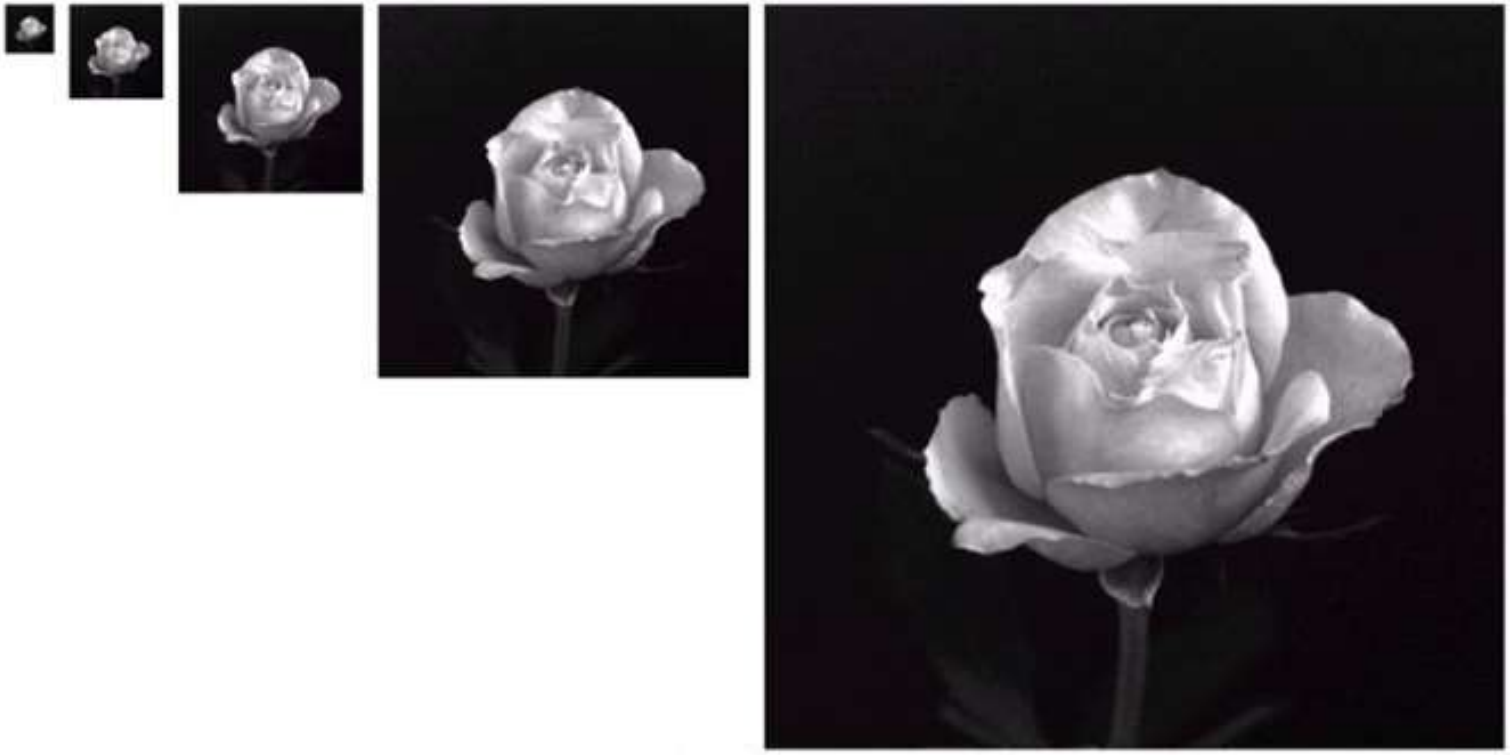
```
void vflip(const void *in, int width, int height, int istep,  int pix_size, void *out, int
    ostep)
{
     out=(char*)out+(height-1)*ostep;
     for(int yi=0; yi<height; ++yi, in=(char*)in+istep, out=(char*)out-ostep)
     {
         memcpy(out, in, width*pix_size);
      }
}
```

```
void hflip(const void *in, int width, int height, int istep,  int pix_size, void *out, int
    ostep)
{
     char * _in=(char*)in;
     char *_out=(char*)out+(width-1)*pix_size;

     for(int yi=0; yi<height; ++yi, _in+=istep, _out+=ostep)
     {
         char *in_x=_in,
         char *out_x=_out;
          for(int xi=0; xi<width; ++xi, in_x+=px_size, out_x-=px_size)
              memcpy(out_x, in_x, px_size) ;
      }
}
```

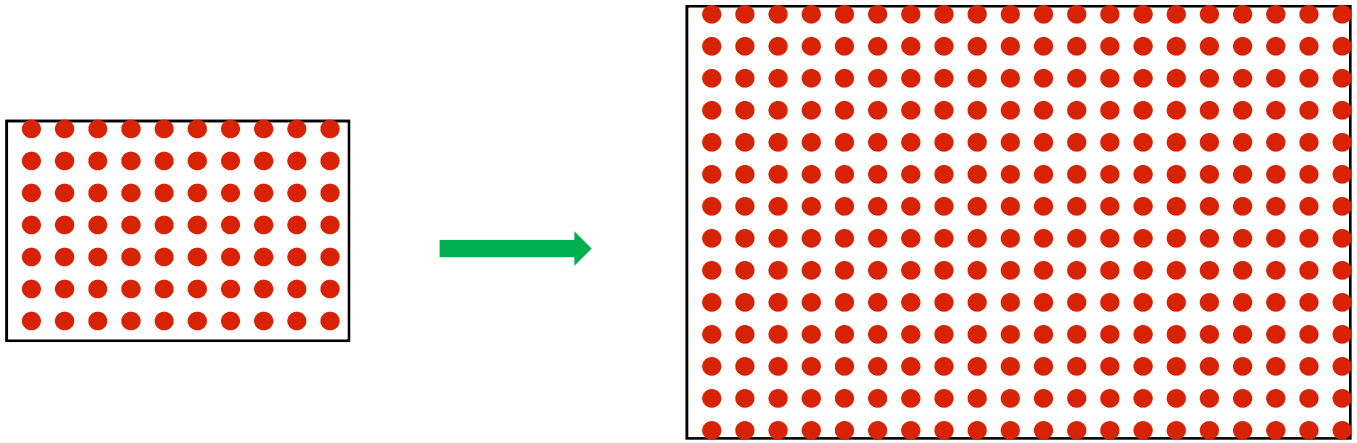
Resize（zoom in/zoom out）

- ？？



Zoom in (放大)

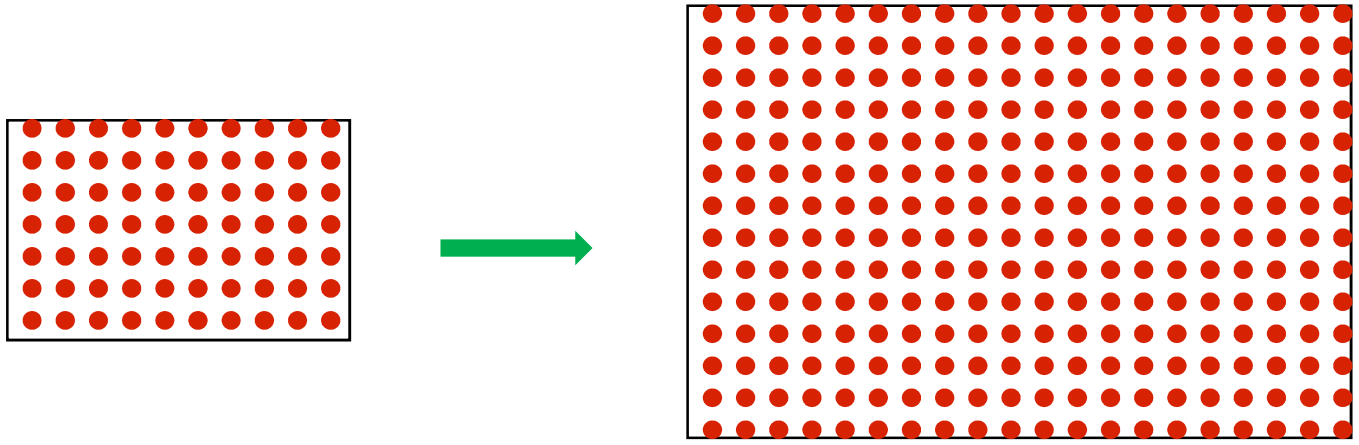
- The correspondence of pixels before and after resize?



$$[x', y'] = [s_x.x, s_y.y]$$

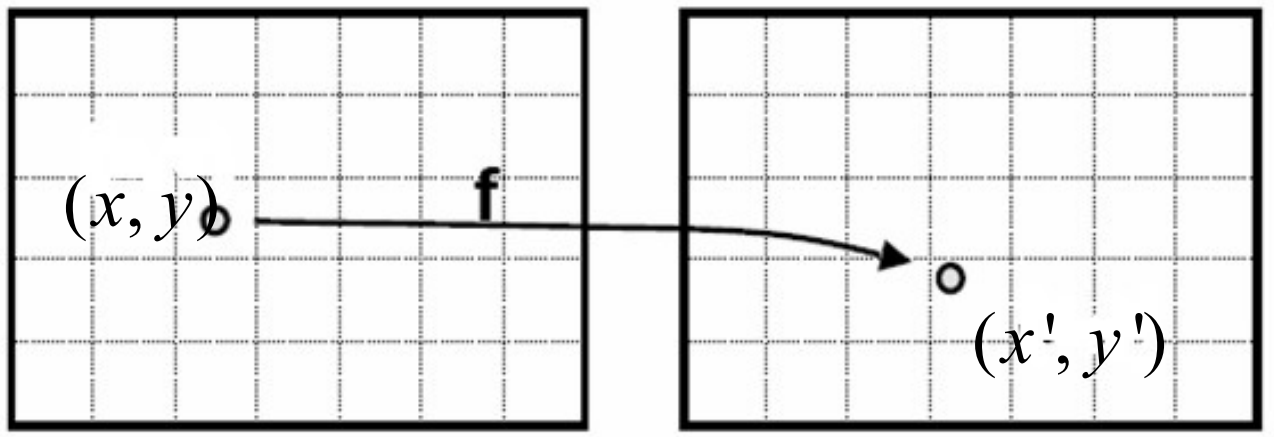
Zoom in (放大)

- How to fill the new added pixels?



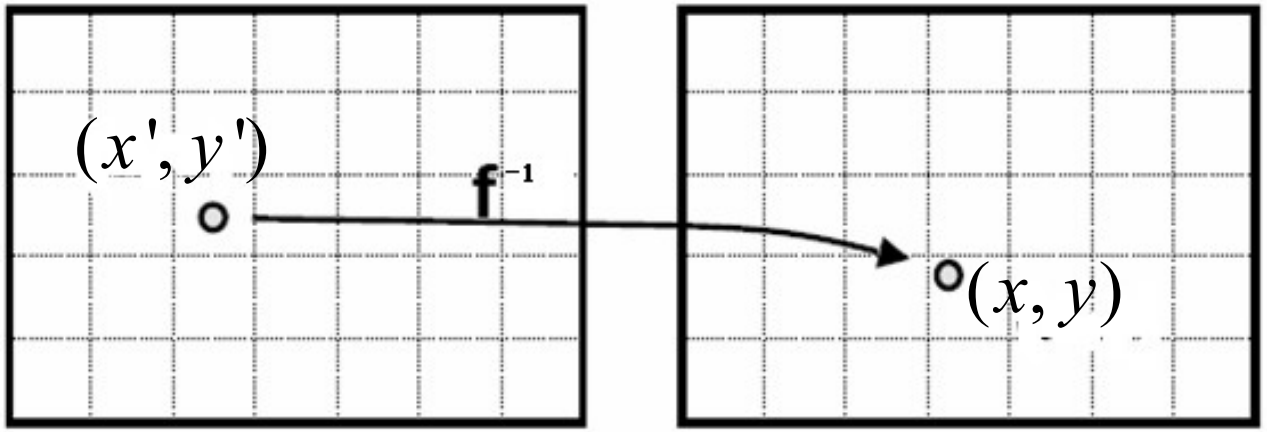
Solution-1?

- **Projection (from source to target):**
  - 对小图中的每个像素，计算其在大图中对应的像素，再拷贝小图的像素值到大图。



Solution-2?

- **Lookup (from target to source):**
  - 对大图中的每个像素，计算其在小图中对应的像素，再拷贝小图的像素值到大图。

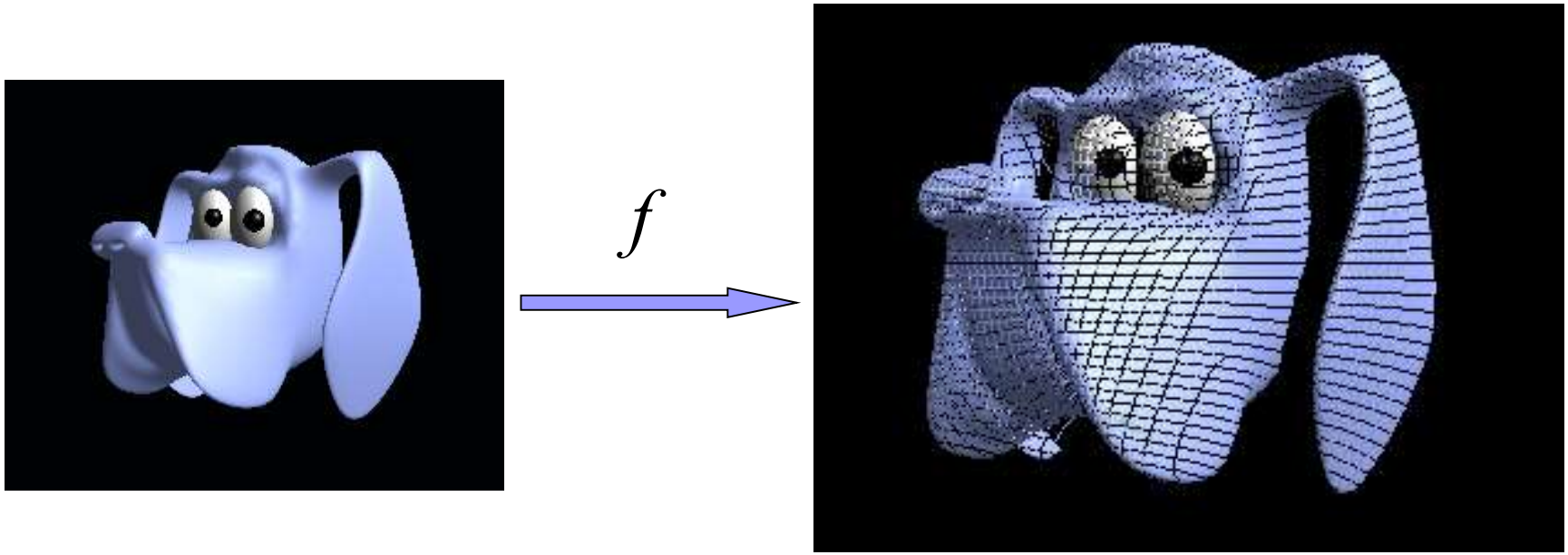


Projection vs Lookup?

- ….

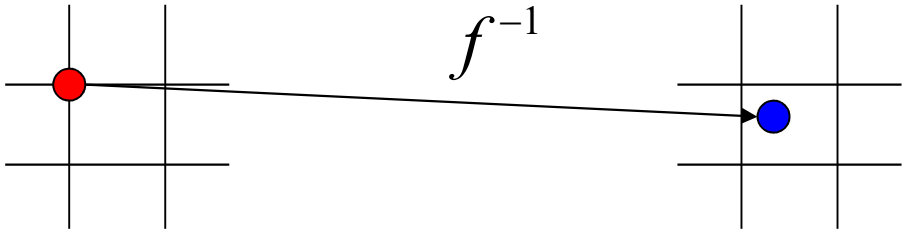
Projection

- **Some pixels may not be filled…..**



Lookup

- How to get pixel color in fractional coordinates, e.g. (1.3, 2.7)?



Target Source

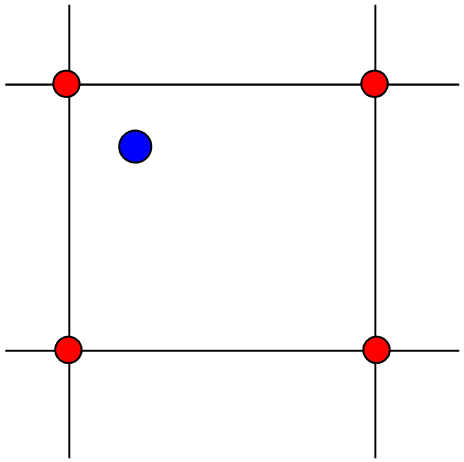
Resampling (重采样)

- 基于邻近像素的值，计算非整数位置上的颜色值
  - 最近邻 （**Nearest Neighbor)**
  - 双线性插值 **(Bilinear Interpolation)**
  - 双三次插值 **(Bicubic Interpolation)**

Nearest **Bilinear** **Bicubic**

Nearest Neighbor (最近邻)

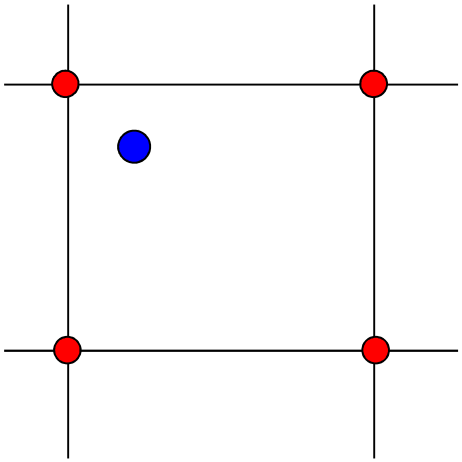
- **Find the nearest source pixel, and output its color**

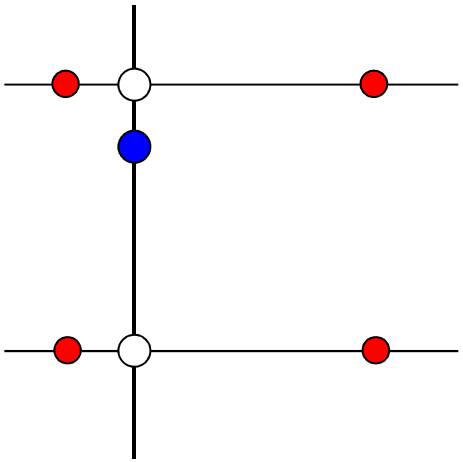


$$x = \text{int}(x + 0.5)$$
$$y = \text{int}(y + 0.5)$$

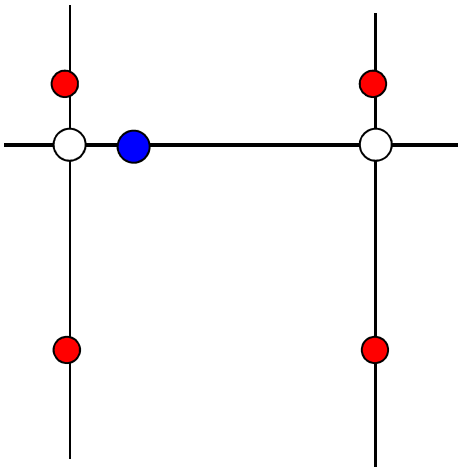
Bilinear Interpolation

- For the 4-neighboring pixels, do horizontal and vertical 1D linear interpolation, respectively.
  - 2次水平，1次垂直
  - 2次垂直，1次水平





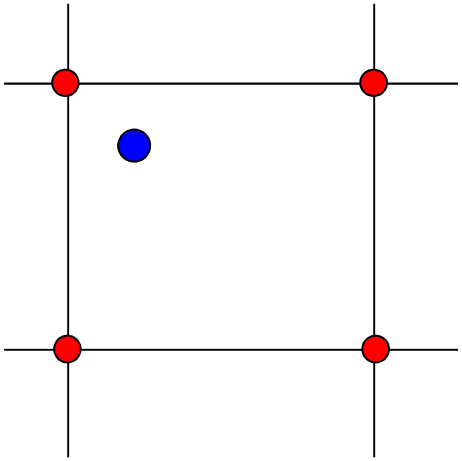
2水平+1垂直

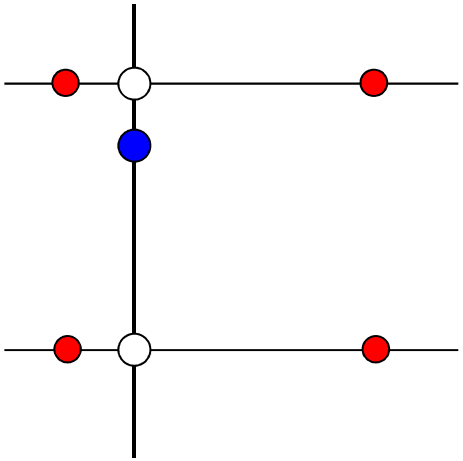


2垂直+1水平

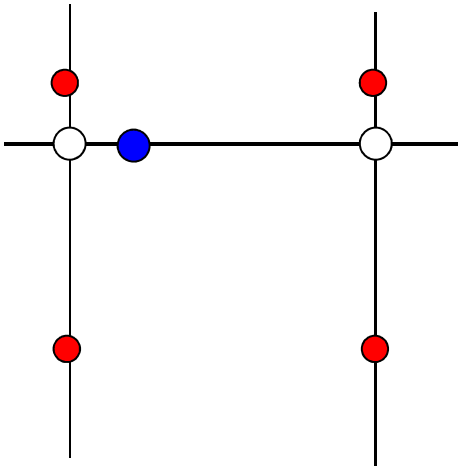
Bilinear Interpolation

- 两种方式是否等价？





2水平+1垂直



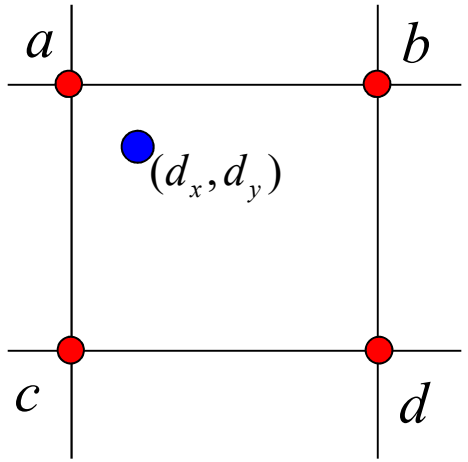
2垂直+1水平

Implementation

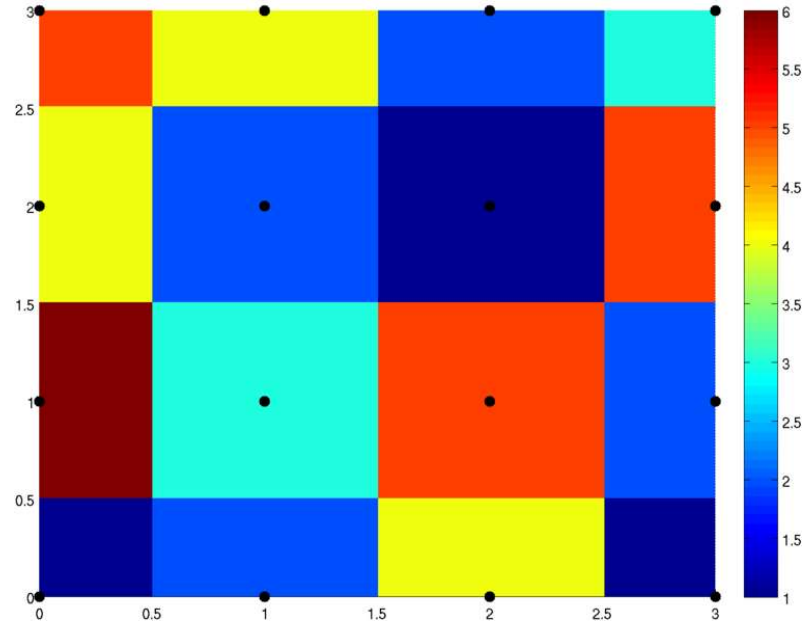
- 一次双线性插值最少需多少次乘法运算？

Implementation

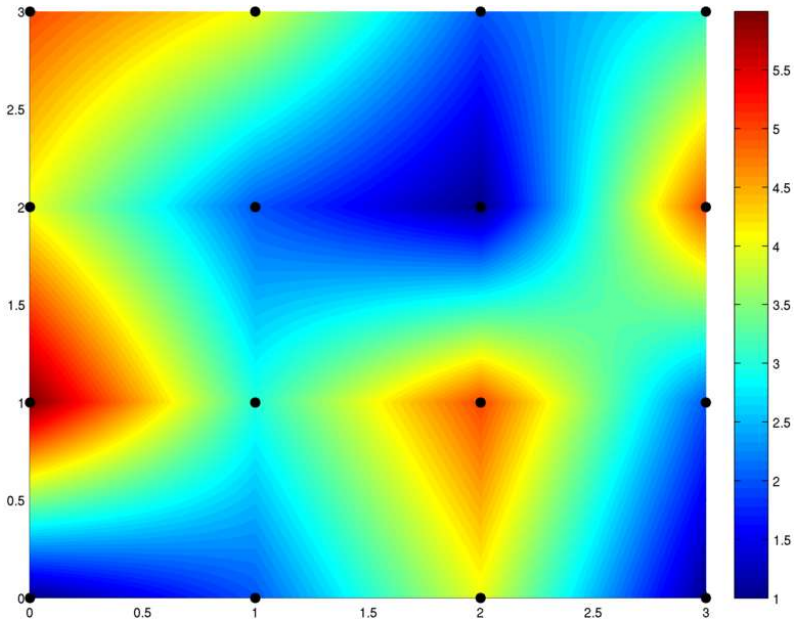
```
float bilinear(float a, float b, float c, float d, float dx, float dy)
{
    float  h1=a+dx*(b-a);          // = (1-dx)*a + dx*b
    float  h2=c+dx*(d-c);
    return h1+dy*(h2-h1);
}
```



NN vs Bilinear

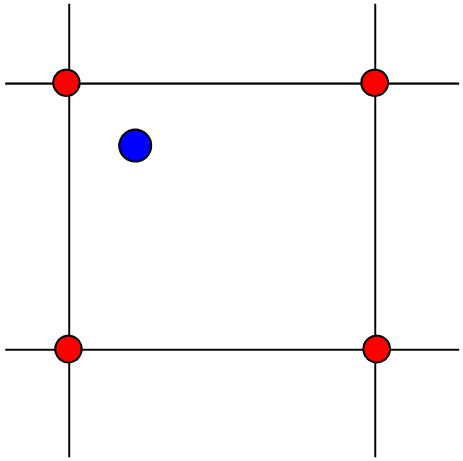


NN



Bilinear

Bicubic Interpolation

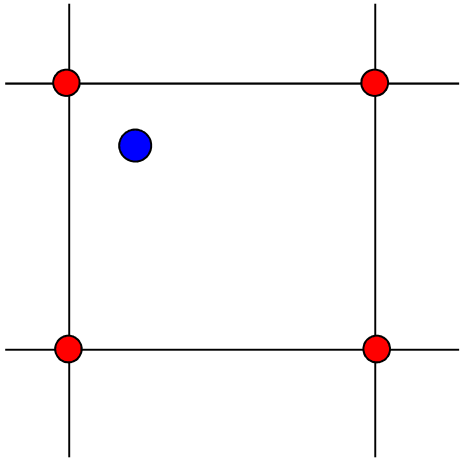


?

$y = ax + b$

?

Bicubic Interpolation

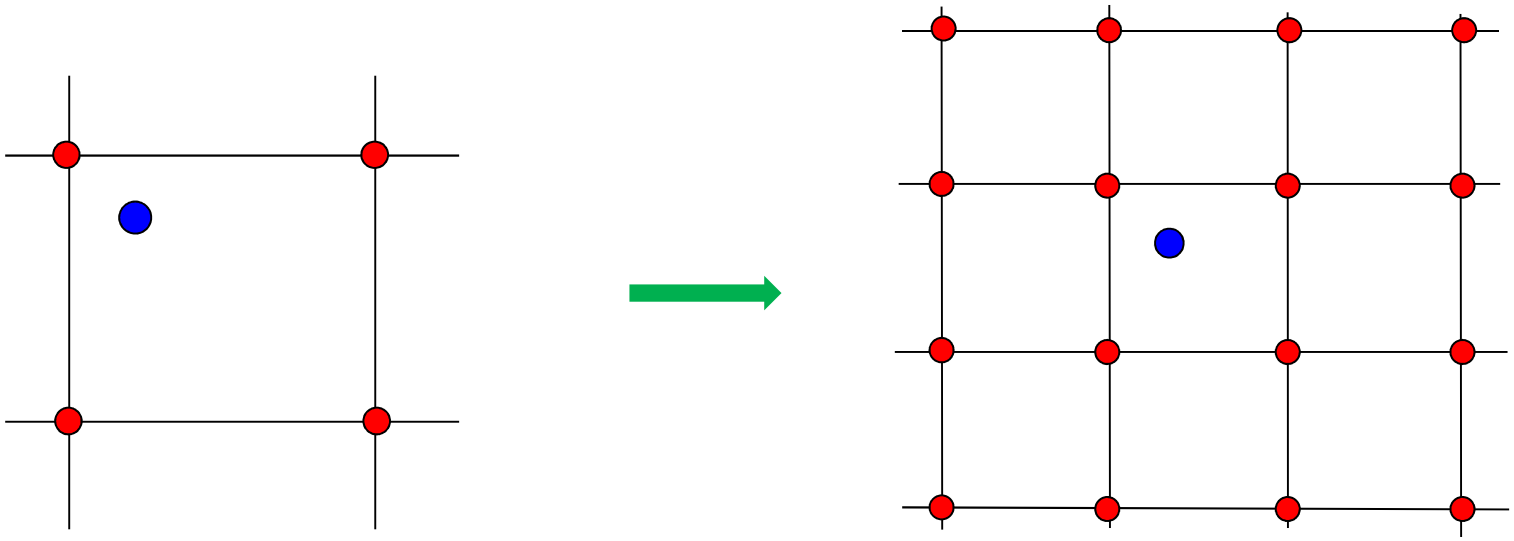


?

$y = ax + b$

$y = ax^3 + bx^2 + cx + d$

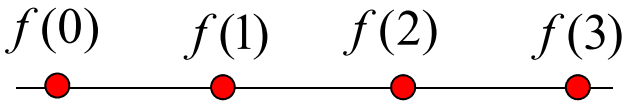
Bicubic Interpolation



$$y = ax + b$$

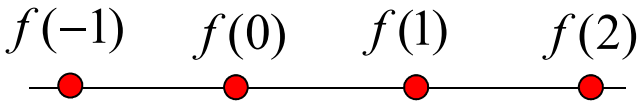
$$y = ax^3 + bx^2 + cx + d$$

a, b, c, d ?



$$\begin{cases} f(0) = d \\ f(1) = a + b + c + d \\ f(2) = 8a + 4b + 2c + d \\ f(3) = 27a + 9b + 3c + d \end{cases} \longrightarrow [a, b, c, d]$$

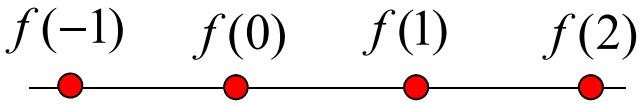
正确解法



$$y' = 3ax^2 + 2bx + c$$

$$\begin{cases} f(0) = d \\ f(1) = a + b + c + d \\ f'(0) = c \\ f'(1) = 3a + 2b + c \end{cases} \longrightarrow [a, b, c, d]$$

The derivative of discrete function



$$f'(x) = \frac{[f(x+1) - f(x)] + [f(x) - f(x-1)]}{2}$$

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j.$$

The interpolation problem consists of determining the 16 coefficients a_{ij}. Matching $p(x,y)$ with the function values yields

1. $f(0,0) = p(0,0) = a_{00}$
2. $f(1,0) = p(1,0) = a_{00} + a_{10} + a_{20} + a_{30}$
3. $f(0,1) = p(0,1) = a_{00} + a_{01} + a_{02} + a_{03}$
4. $f(1,1) = p(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}$

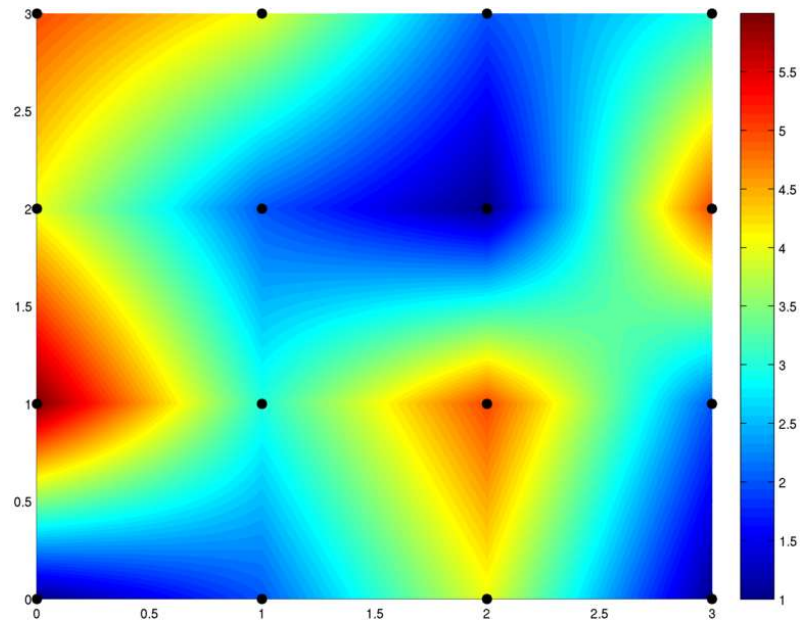
Likewise, eight equations for the derivatives in the x-direction and the y-direction

1. $f_x(0,0) = p_x(0,0) = a_{10}$
2. $f_x(1,0) = p_x(1,0) = a_{10} + 2a_{20} + 3a_{30}$
3. $f_x(0,1) = p_x(0,1) = a_{10} + a_{11} + a_{12} + a_{13}$
4. $f_x(1,1) = p_x(1,1) = \sum_{i=1}^{3}\sum_{j=0}^{3} a_{ij}i$
5. $f_y(0,0) = p_y(0,0) = a_{01}$
6. $f_y(1,0) = p_y(1,0) = a_{01} + a_{11} + a_{21} + a_{31}$
7. $f_y(0,1) = p_y(0,1) = a_{01} + 2a_{02} + 3a_{03}$
8. $f_y(1,1) = p_y(1,1) = \sum_{i=0}^{3}\sum_{j=1}^{3} a_{ij}j$

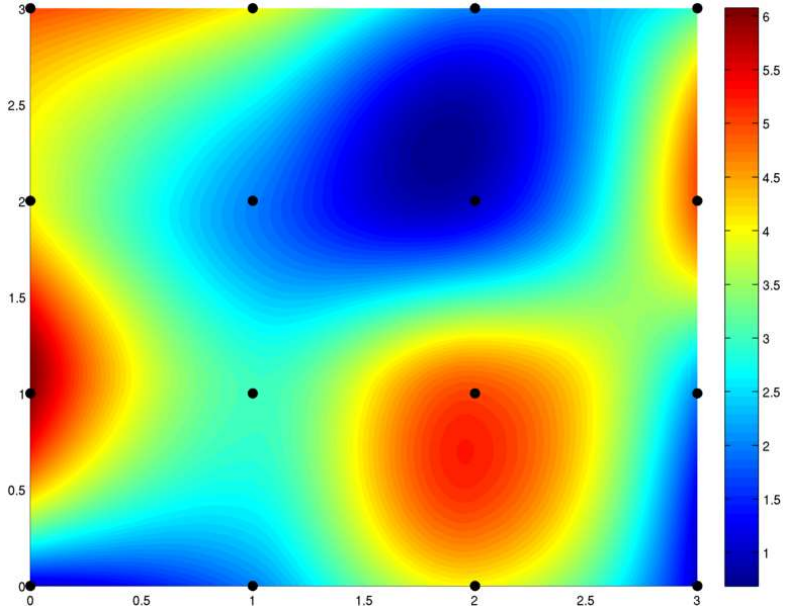
And four equations for the cross derivative xy.

1. $f_{xy}(0,0) = p_{xy}(0,0) = a_{11}$
2. $f_{xy}(1,0) = p_{xy}(1,0) = a_{11} + 2a_{21} + 3a_{31}$
3. $f_{xy}(0,1) = p_{xy}(0,1) = a_{11} + 2a_{12} + 3a_{13}$
4. $f_{xy}(1,1) = p_{xy}(1,1) = \sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij}ij$

Bilinear & Bicubic



Bilinear



Bicubic

NN

Bilinear

Bicubic

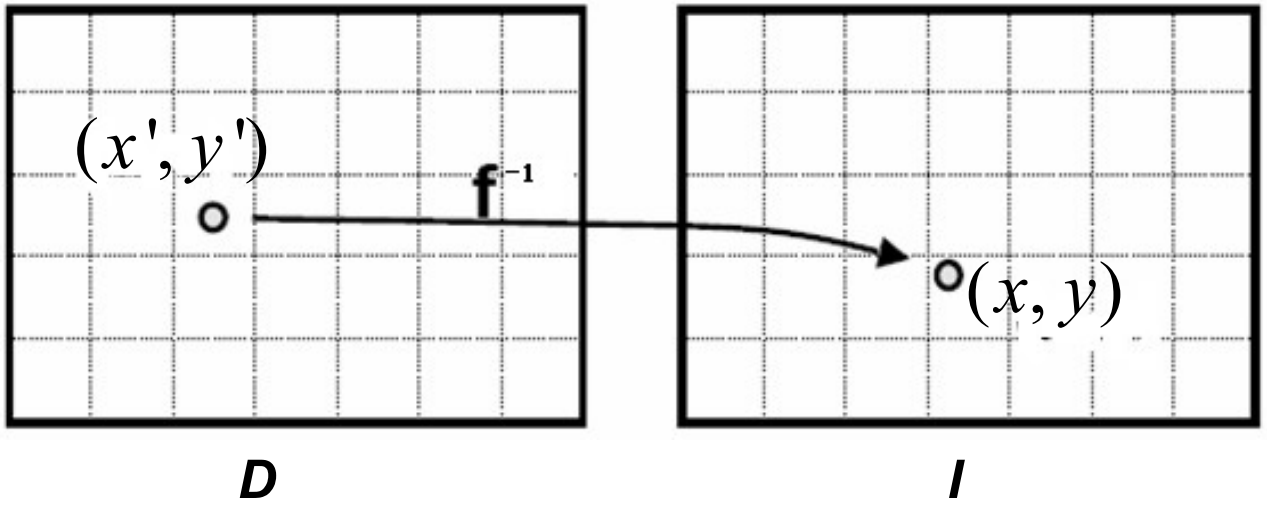
Super-Resolution (超分辨率)

常见几何操作

- 水平&垂直翻转 (flip)
- 缩放 (resize / zoom in /zoom out /scale)
- 旋转
- ……
- 仿射变换 (Affine Transform)
- 透视变换 (Perspective Transform)
- ……
- 图像变形 (Image Warping)

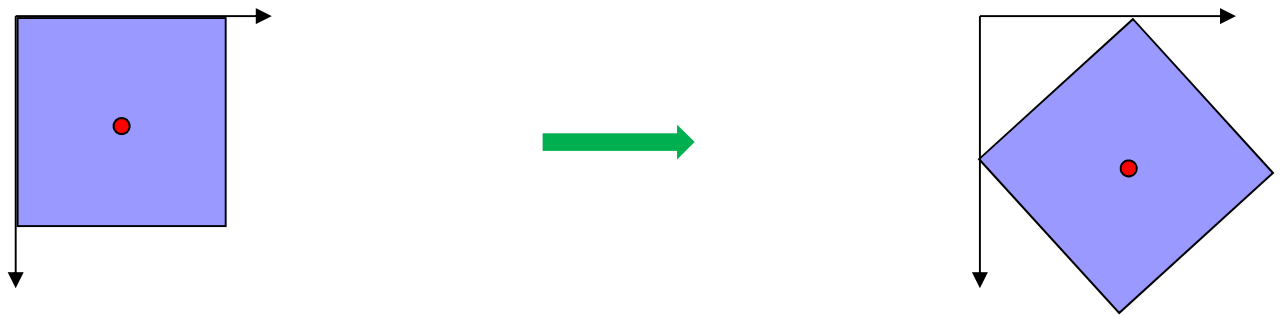
Image Transform (图像变换)

$$D = f(I)$$



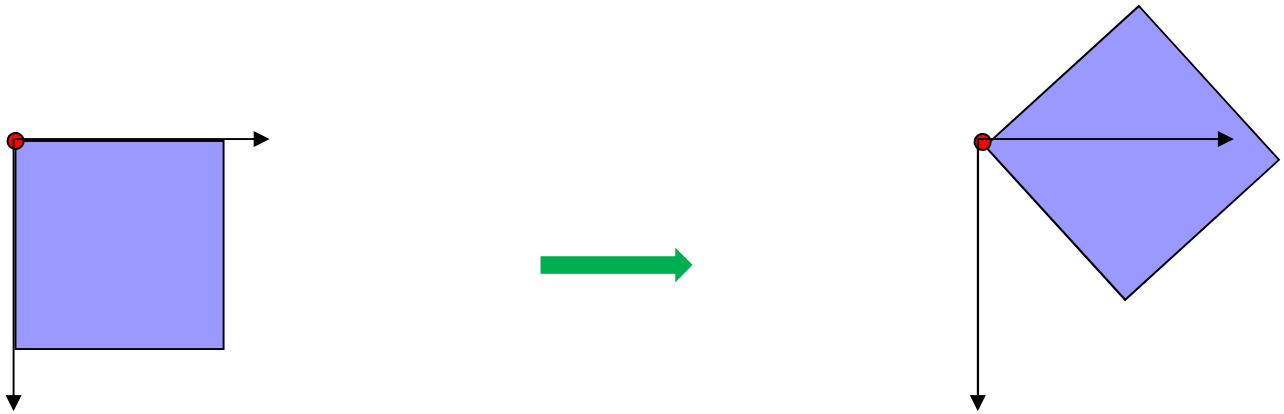
Rotation

$$f: \quad [x', y'] = [x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta]$$



? ?

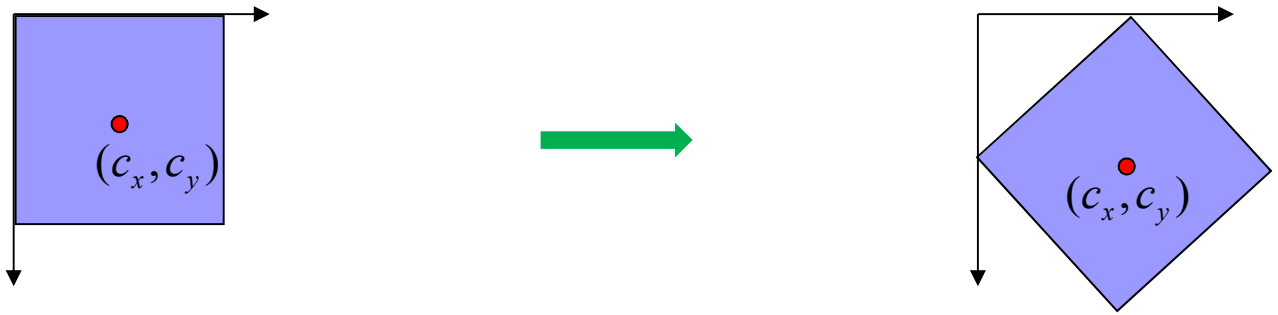
The center of rotation



Rotation around specified center?

$$f(x, y \mid \theta) = [x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta]$$

$$f(x, y \mid \theta, c_x, c_y) = ??$$



更复杂情况

- 先绕**p1**旋转一定角度，再绕**p2**旋转一定角度…
- 依次绕**N**个中心旋转？
- ……
- 旋转、平移、缩放等复合？

Homogeneous Coordinates & Transform Matrix

homogeneous coordinates:

(齐次坐标)

$$[x, y] = [x, y, 1] = [\omega x, \omega y, \omega]$$

2D Affine Transform:

（2维仿射变换）

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A_{2\times 3} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Perspective Transform:

（2维透视变换）

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = A_{3\times 3} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transform Matrix

Translation (平移):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale (缩放):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation (旋转):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

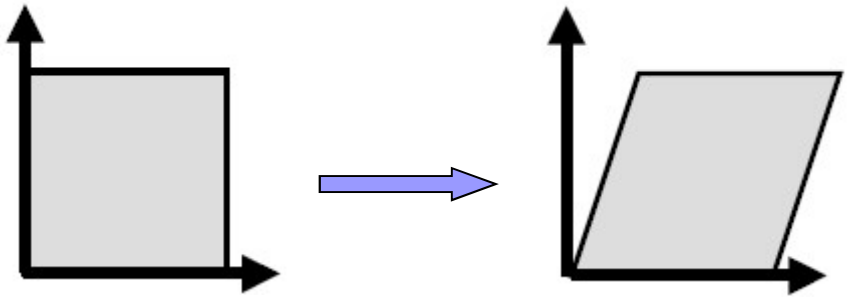
Transform Matrix

X切变（Shear):

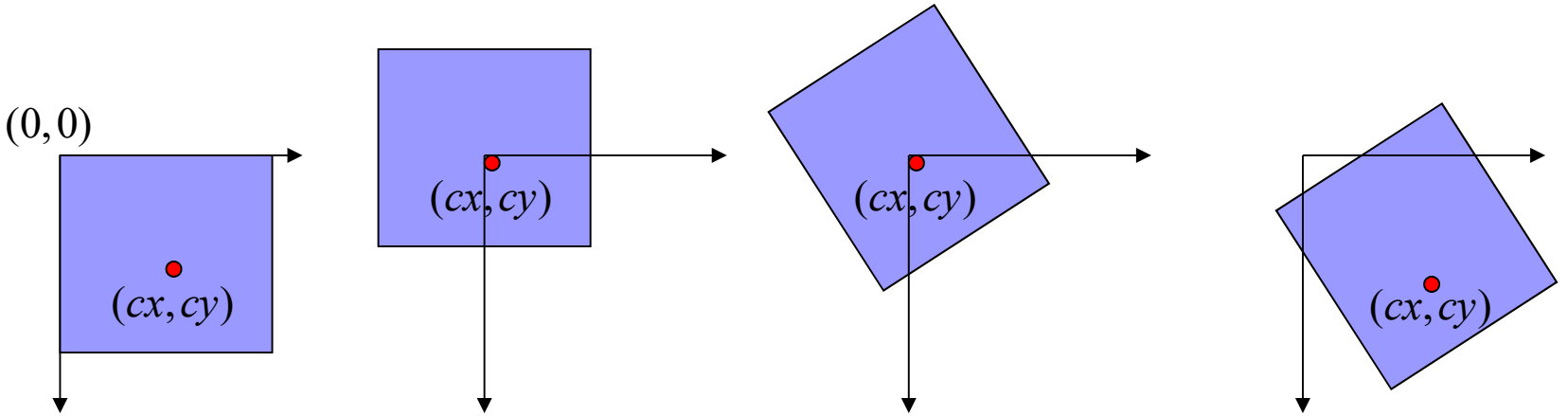
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + sy \\ y \end{bmatrix} = \begin{pmatrix} 1 & s & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Y切变:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ sx + y \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ s & 1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Transformations in Matrix Form



=　平移（-cx, -cy）　+　旋转 angle　+　平移 （cx, cy）

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transformations in Matrix Form

- 多个变换的复合可表示为其变换矩阵的乘积

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

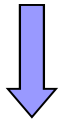
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & -c_x\cos\theta + c_y\sin\theta + c_x \\ \sin\theta & \cos\theta & -c_x\sin\theta - c_y\cos\theta + c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

More complex cases ??

- 先绕p1旋转一定角度，再绕p2旋转一定角度…
- 依次绕N个中心旋转？
- ……
- 旋转、平移、缩放等复合？

Interface Design of functions

```
void  Rotate(const MyImage &input, MyImage &output, float angle);
```



更好的设计

```
void  WarpAffine(const MyImage &input, MyImage &output, const Matrix23 &mat
```

```
void  CalcRotationMatrix(Matrix23 &matrix, float angle);
```

Affine Transform

- **Linear Transform:**

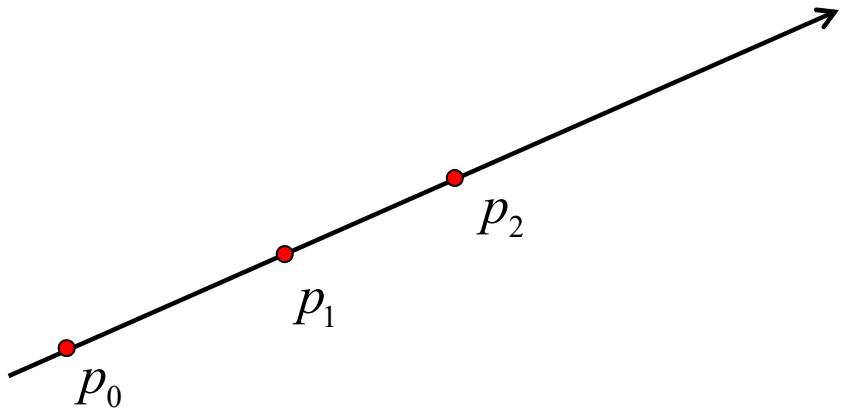
$$f(a+b) = f(a) + f(b)$$
$$f(ka) = kf(a)$$

- **Affine = Linear + Translation** $f(a)+t$

e.g 2D仿射变换： $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{pmatrix} u \\ v \end{pmatrix}$

Affine Transform

- **Keeping the following geometric properties:**
  - 共线性
  - 共线向量的长度比
  - 重心坐标



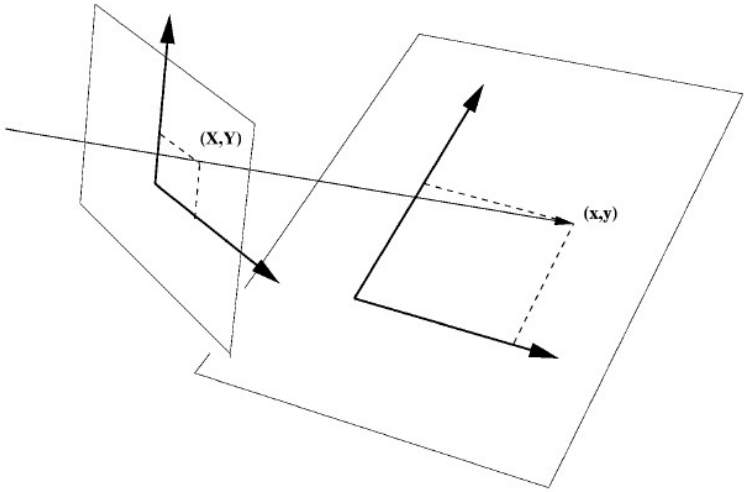
Special Affine Transform

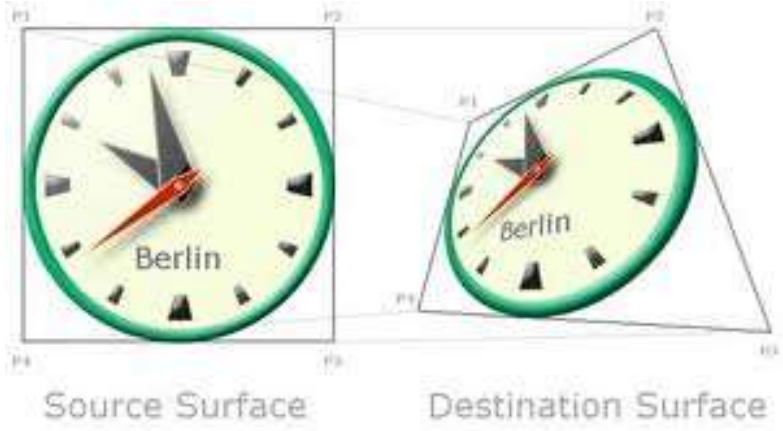
- 刚性变换（**Rigid Transformation)**
  - 只包含平移和旋转
  - 保持物体的形状(保角）和尺寸
  - 相当于正交变换

- 相似变换（**Similar Transformation)**
  - 只包含平移、旋转和等比缩放
  - 保持物体的形状

Perspective Transform

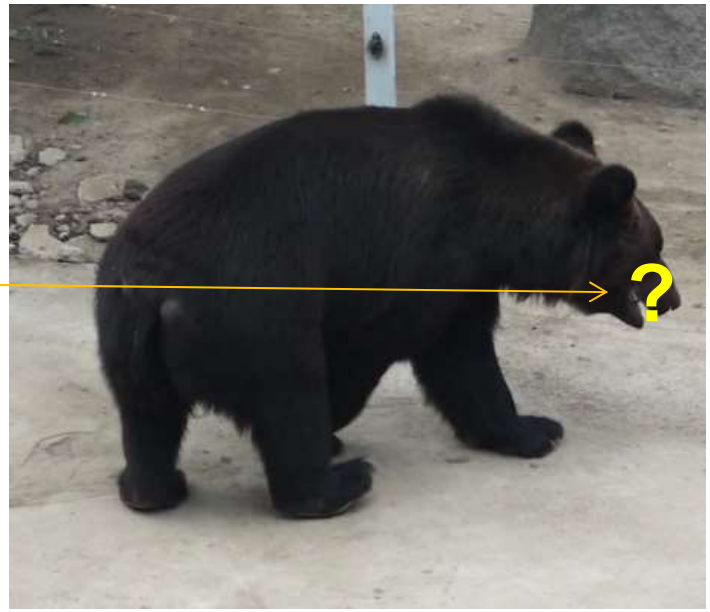
- **Affine Transform：**
  - 在同一平面内部的变换
- **Perspective Transform**
  - 可表示不同视角观察到的同一平面，或同一视角观察到的不同平面之间的变换





应用：图像匹配

t

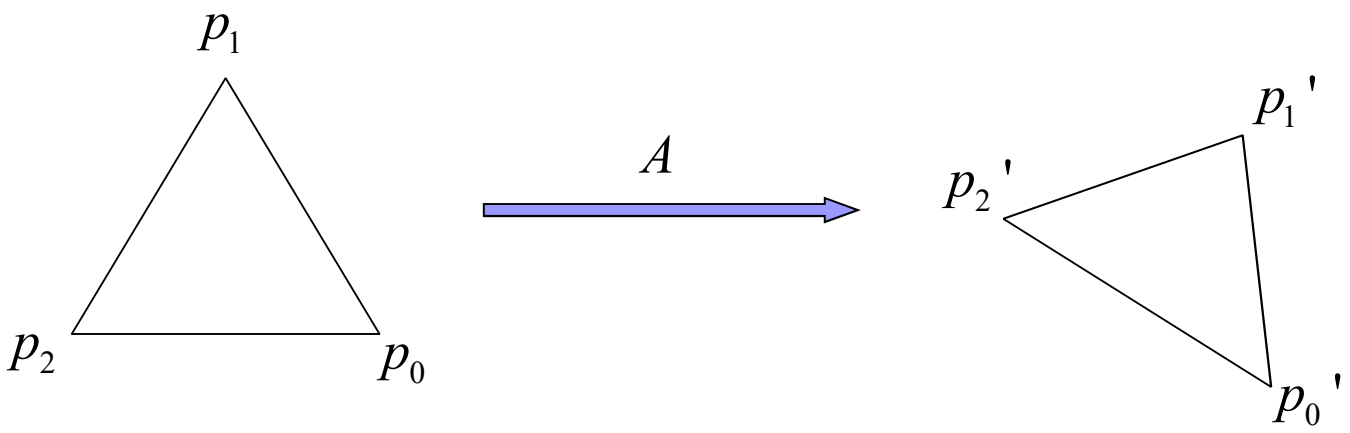


$t+1$

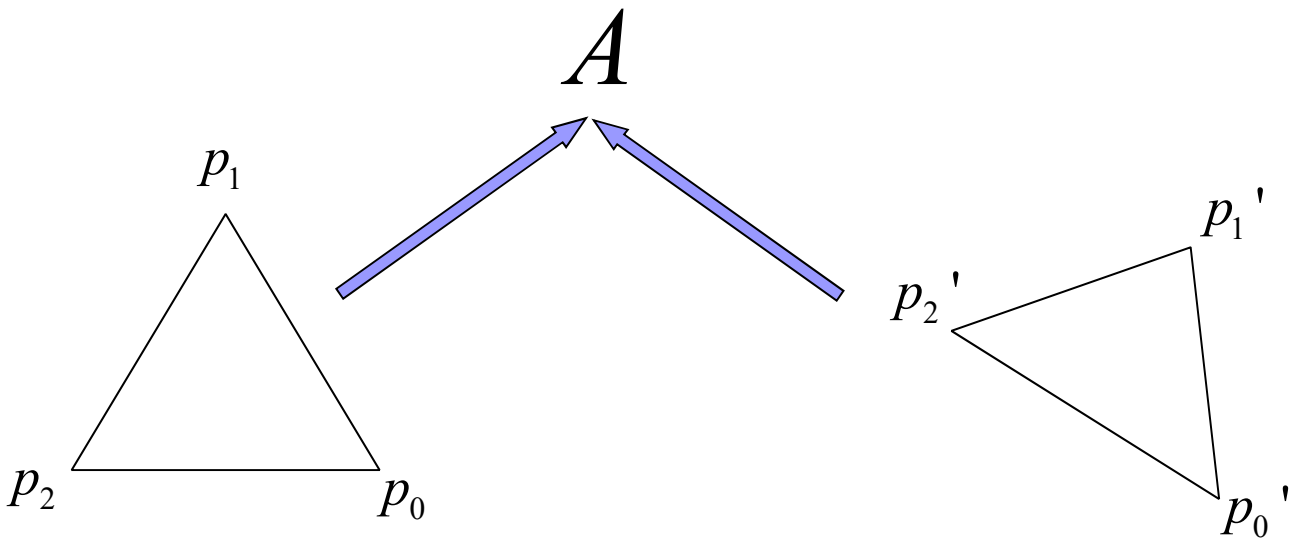
应用：图像匹配

- 基于仿射变换的图像匹配
  - 先计算从第t帧到第t+1帧的仿射变换A
  - 利用A对第t帧的图像进行变换，将变换的结果作为与第t+1帧配准的图像。

- 估计两副图像之间的仿射变换?



- 估计两副图像之间的仿射变换?



- 不共线的3个平面点对决定一个二维仿射变换

$$\begin{bmatrix} u_i' \\ v_i' \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \longrightarrow \begin{cases} u_0 a + v_0 b + c = u_0' \\ u_0 d + v_0 e + f = v_0' \\ u_1 a + v_1 b + c = u_1' \\ u_1 d + v_1 e + f = v_1' \\ u_2 a + v_2 b + c = u_2' \\ u_2 d + v_2 e + f = v_2' \end{cases}$$

$$p_i = [u_i, v_i]$$

- **>3 个点对？**

$$A = \arg\min_{A} \sum_{i} \| Ap_i - p_i' \|^2$$

点对？

- 通过特征点跟踪获得

应用：图像匹配

- 基于仿射变换的图像匹配
  - 在第t帧检测特征点（特征检测）
  - 计算特征点在第t+1帧的对应（特征跟踪）
  - 根据特征点对估计第t帧到第t+1帧的仿射变换A
  - 利用A对第t帧的图像进行变换，将变换的结果作为与第t+1帧配准的图像：

$$Ap_t \leftrightarrow p_{t+1}$$

常见几何操作

- 水平&垂直翻转 (flip)
- 缩放 (resize / zoom in /zoom out /scale)
- 旋转
- ……
- 仿射变换 (Affine Transform)
- 透视变换 (Perspective Transform)
- ……
- 图像变形 (Image Warping)

图像变形

实验1.3：图像变形

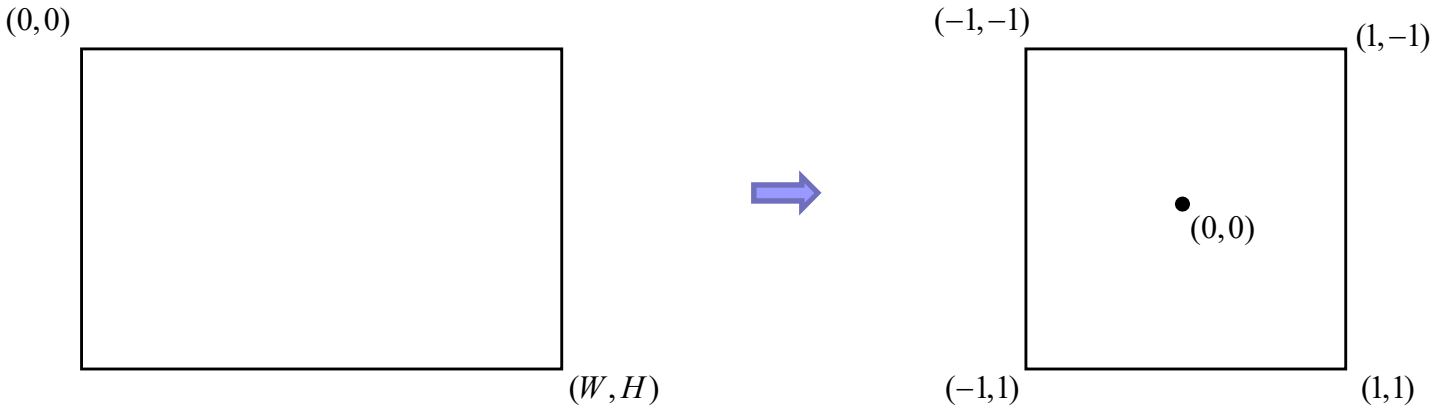
- 记[x’, y’]=f([x, y])为像素坐标的一个映射，实现f所表示的图像形变。f的逆映射为：

$$[x,y]=f^{-1}([x',y'])=\begin{cases}[x',y'] & \text{if } r>1\\ [\cos(\theta)x'-\sin(\theta)y',\sin(\theta)x'+\cos(\theta)y'] & \text{otherwise}\end{cases}$$

其中： $r=\sqrt{x'^2+y'^2}$ $\theta=(1-r)^2$

$[x,y],[x',y']$ 都是中心归一化坐标，请先进行转换；

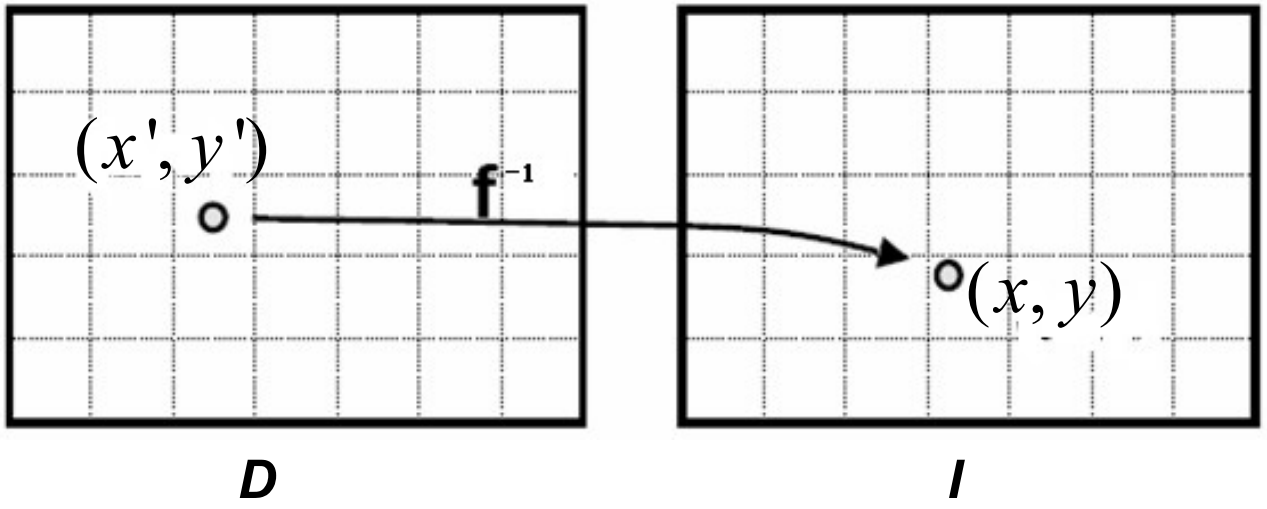
中心归一化坐标



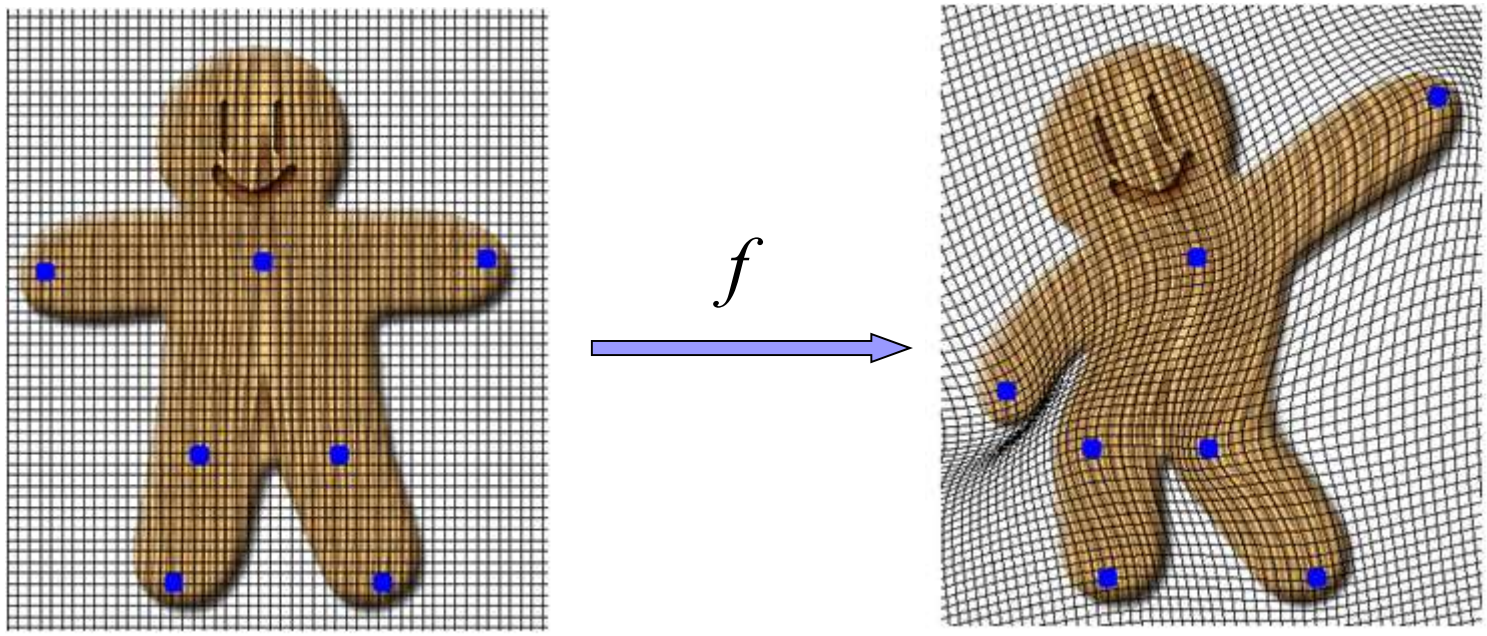
$$x' = \frac{x - 0.5W}{0.5W} \qquad y' = \frac{y - 0.5H}{0.5H}$$

逆向查找的缺点/局限性？

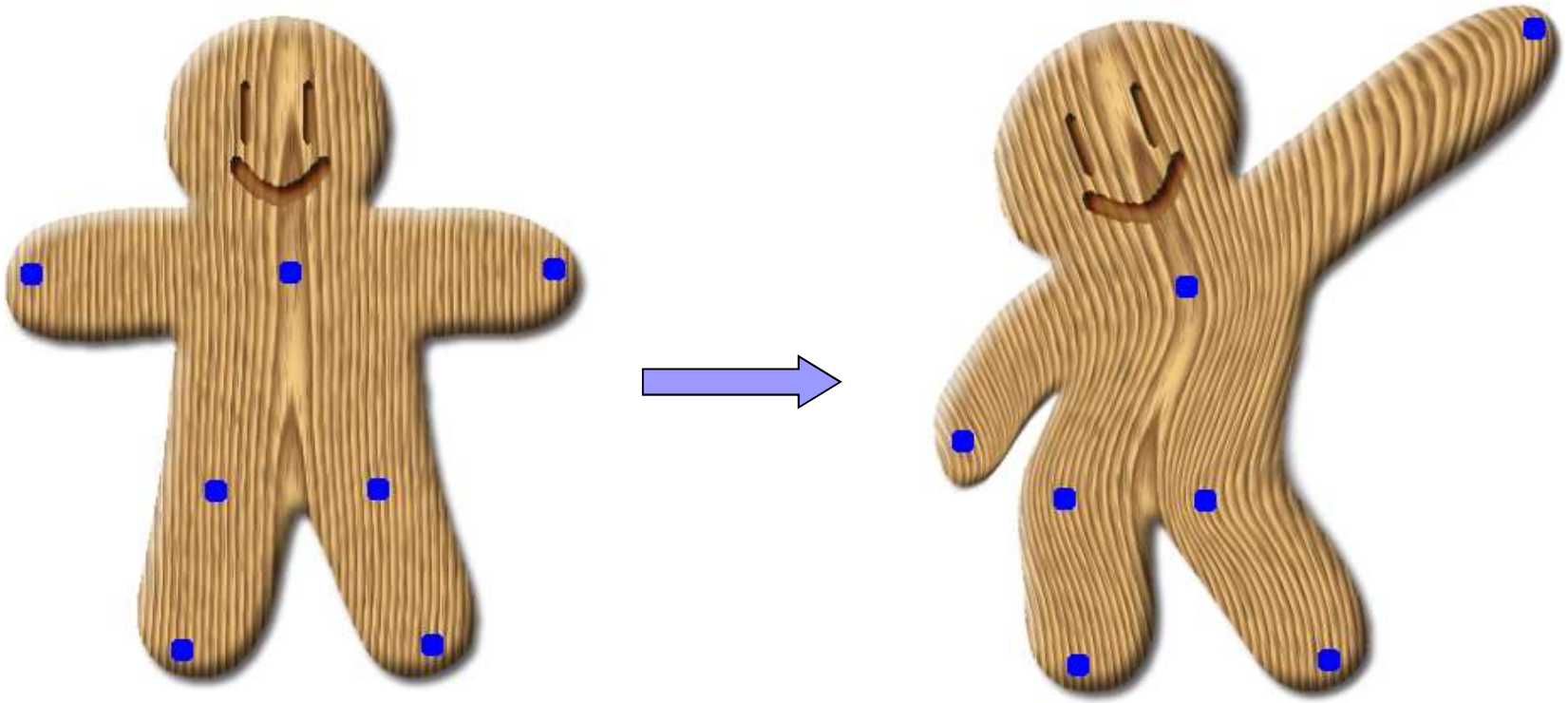
$$D = f(I)$$



正向变换如何实现?



交互式图像变形



- 简述图像几何变换两种方式：正向投射和逆向查找的过程，并说明两种方式分别需要解决的关键问题。

- 三次插值和双线性插值相比，抗锯齿和马赛克的效果更好，请说明为什么。

- 已知两幅图像中的N个点对(p_i , q_i)， i=1,…,N, N>3, 请简述求两幅图像之间符合这N个点对约束的仿射变换A的方法。