中国科学技术大学
University of Science and Technology of China

计算机图形学
Computer Graphics

陈仁杰

renjiec@ustc.edu.cn

http://staff.ustc.edu.cn/~renjiec

copy constructor

```
Matrix Matrix::operator*(double v){
    Matrix M(rows, cols);
    for(i=0; i<rows*cols; i++) M[i] = (*this)[i]*v;
    return M;
}


Matrix A;
Matrix M = A*10;
```

```
// copy constructor
Matrix::Matrix(const Matrix& M) {
        ...
        data = new double[rows*cols];
        std::copy_n(M.data, rows*cols, data);  // deep copy
}
```

```
// move constructor // 
https://en.cppreference.com/w/cpp/language/move_constructor
Matrix::Matrix(Matrix&& M) {
        ...
        data = std::exchange(M.data, nullptr);  // no copy
}
```

printf v.s. std::cout

- printf(“x=%3.6f, y=%3.1f”, x, y);

- std::cout<<“x=“<<std::setprecision(6)<<“,y=“<<std::setprecision(1)<<y;

std::format

- c++20
- https://github.com/fmtlib/fmt

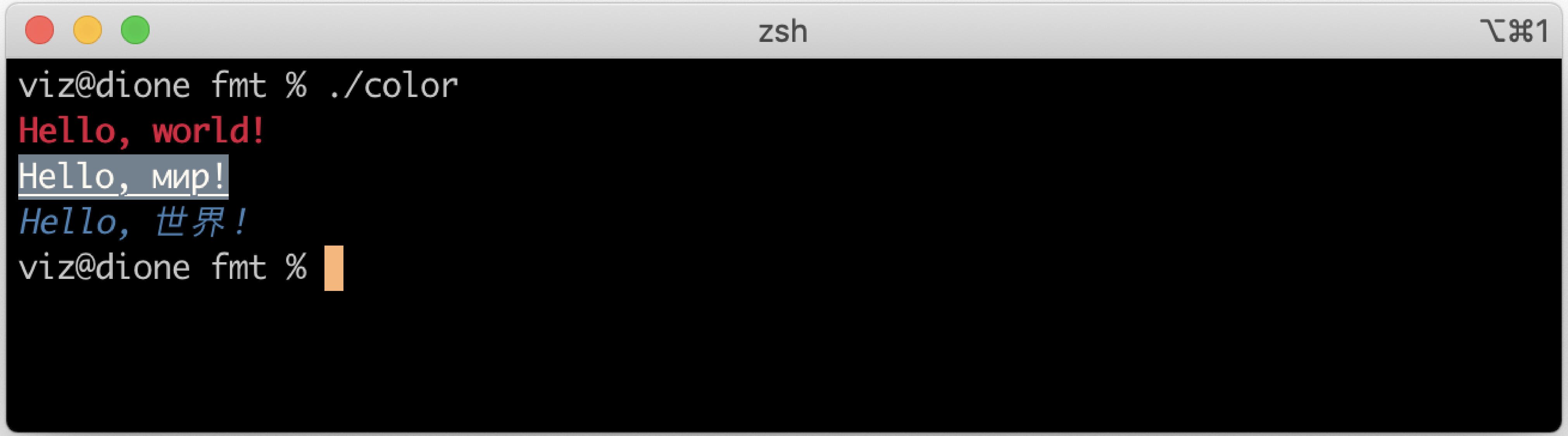
```
// Format a string

std::string s = fmt::format("The answer is {}.", 42);
// s == "The answer is 42.“

// Format a string using positional arguments

std::string s = fmt::format("I'd rather be {1} than {0}.", "right", "happy");
// s == "I'd rather be happy than right."
```

```
// Print a container (run)

#include <vector>
#include <fmt/ranges.h>

int main() {
  std::vector<int> v = {1, 2, 3};
  fmt::print("{}\n", v);
}

Output:
{1, 2, 3}
```

fmtlib

```
// Print with colors and text styles

#include <fmt/color.h>
int main() {
  fmt::print(fg(fmt::color::crimson) | fmt::emphasis::bold, "Hello, {}!\n", "world");
  fmt::print(fg(fmt::color::floral_white) | bg(fmt::color::slate_gray) | fmt::emphasis::underline, "Hello, {}!\n", "мир");
  fmt::print(fg(fmt::color::steel_blue) | fmt::emphasis::italic, "Hello, {}!\n", "世界");
}
```

Output on a modern terminal:



```
viz@dione fmt % ./color
Hello, world!
Hello, мир!
Hello, 世界!
viz@dione fmt %
```

图像处理、变形及合成

图像处理

When the modification that we would like to make to a pixel depends on the pixels around it

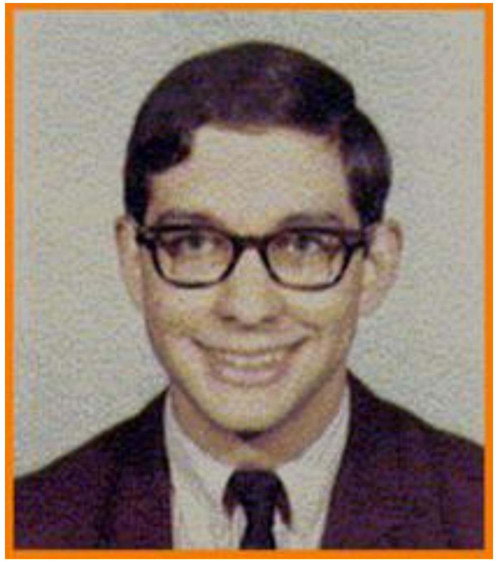
- Blurring
- Edge Detection
- etc

In the simplest case, we define a mask of weights which tells us how the values at adjacent pixels should be combined to generate the new value.

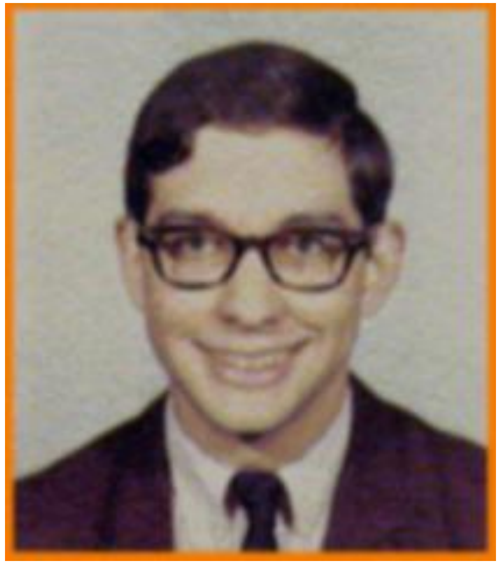
Blurring/Denoising去噪

To blur across pixels, define a mask:

- Whose value is largest at the center pixel
- Whose entries sum to one



Original



Blur

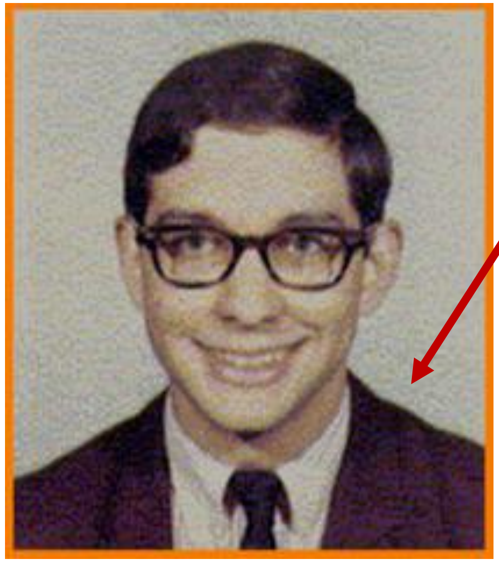
$$\text{filter} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Blurring

Pixel(x,y): red = 36

Green = 36

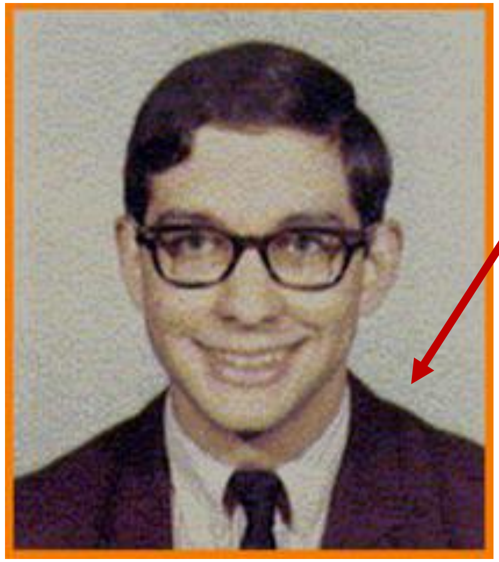
Blue = 0



$$\text{filter} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Blurring

Pixel(x,y): red = 36
Green = 36
Blue = 0



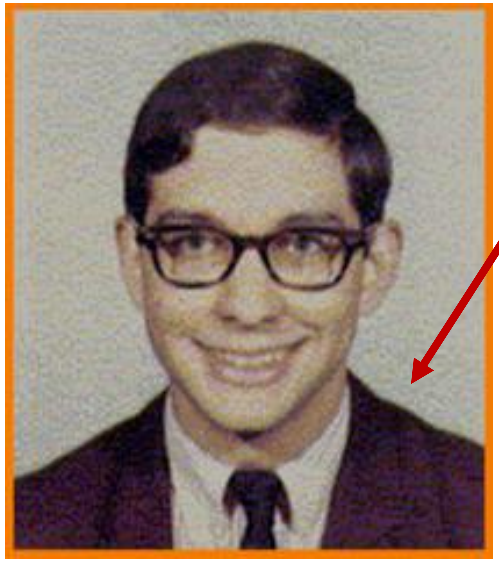
36	109	146
32	36	109
32	36	73

Pixel(x,y).red and its red neighbors

$$\text{filter} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red =
(36 * 1/16) + (109 *2/16) + (146 * 1/16)
(32 * 2/16) + (36 * 4/16) + (109 * 2/16)
(32 * 1/16) + (36 * 2/16) + (73 * 1/16)



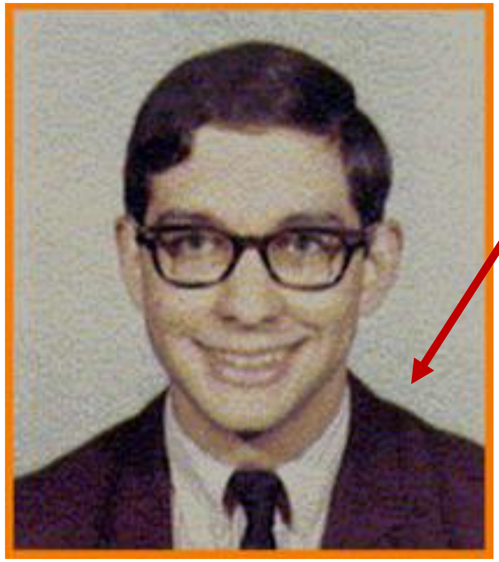
36	109	146
32	36	109
32	36	73

Pixel(x,y).red and its red neighbors

$$\text{filter} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Blurring

New value for Pixel(x,y).red = **62.69**



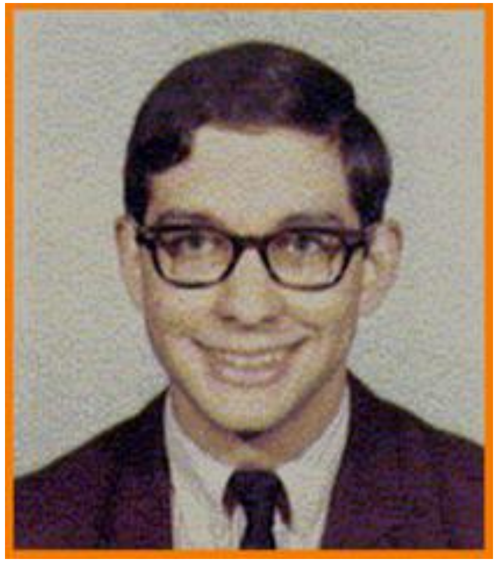
36	109	146
32	36	109
32	36	73

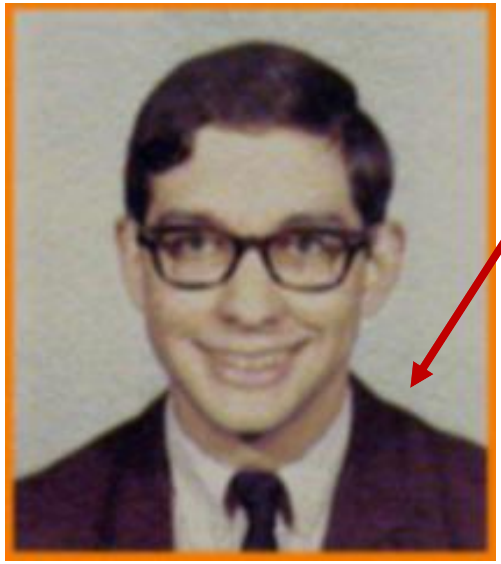
Pixel(x,y).red and its red neighbors

$$\text{filter} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red = **63**





$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Blurring

- Repeat for each pixel and each color channel
  - Note 1: Keep source and destination separate to avoid “drift”
  - Note 2: For boundary pixels, not all neighbors are used, and you

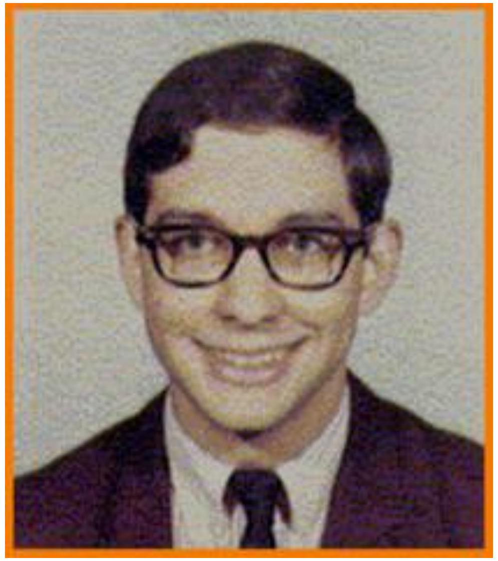
- Need to normalize the mask so that the sum of the values is correct

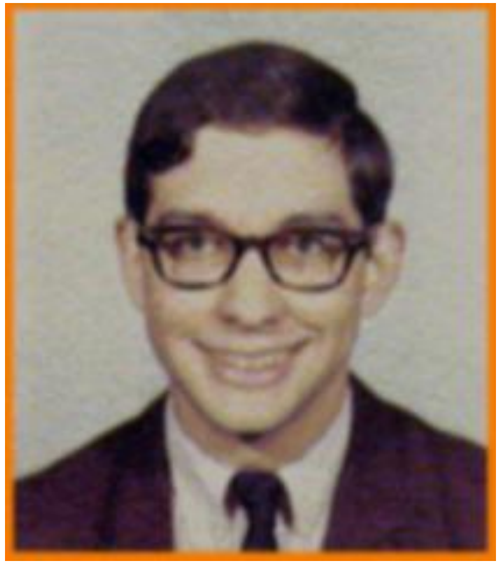
Blurring

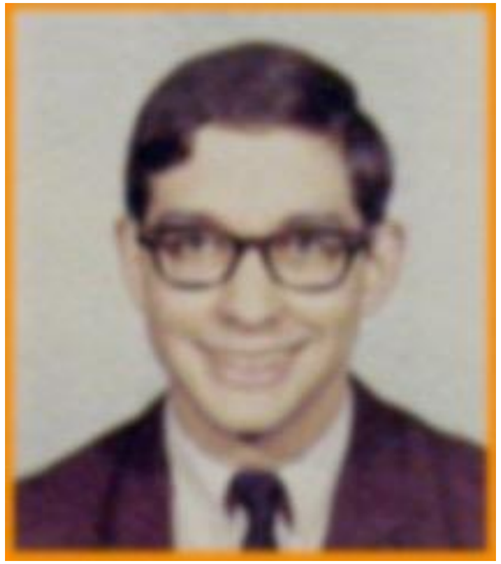
- Larger kernel gives rise to a wider blur

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$\frac{1}{48}\begin{bmatrix}0 & 1 & 2 & 1 & 0\\ 1 & 2 & 4 & 2 & 1\\ 2 & 4 & 8 & 4 & 2\\ 1 & 2 & 4 & 2 & 1\\ 0 & 1 & 2 & 1 & 0\end{bmatrix}$$

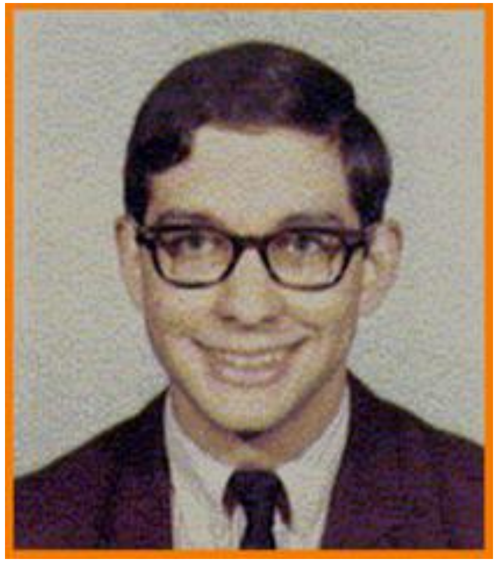




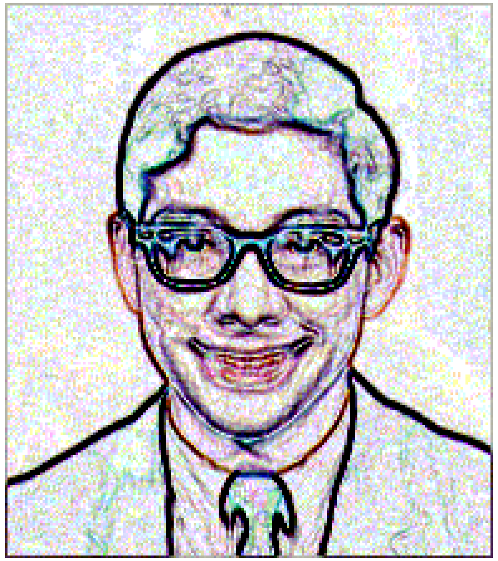


Edge Detection

- To find the edges in an image, define a mask:
  - Whose value is largest at the center pixel
  - Whose entries sum to zero.
- Edge pixels are those whose value is larger (or smaller) than those of its neighbors



Original

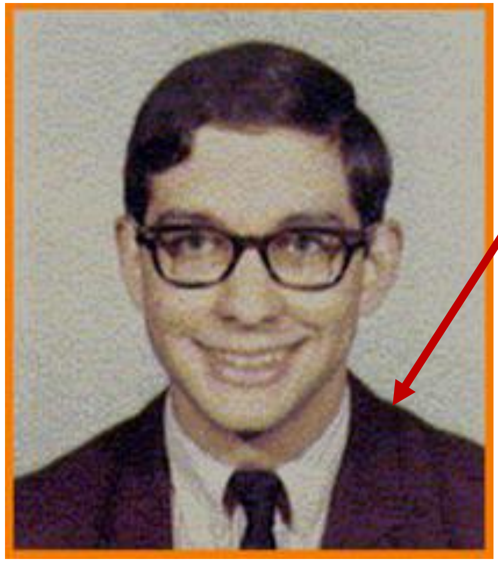


Highlighted Edges

$$\text{filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red =
(36 * -1) + (109 * -1) + (146 * -1)
(32 * -1) + (36 * 8) + (109 * -1)
(32 * -1) + (36 * -1) + (73 * -1)



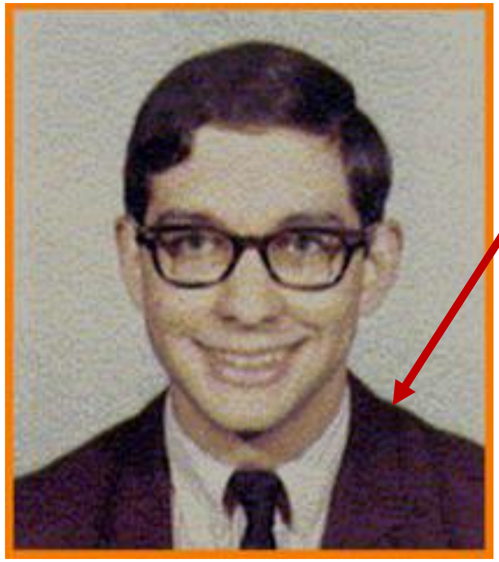
36	109	146
32	36	109
32	36	73

Pixel(x,y).red and its red neighbors

$$\text{filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = **-285**



36	109	146
32	36	109
32	36	73

Pixel(x,y).red and its red neighbors

$$\text{filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = **0**

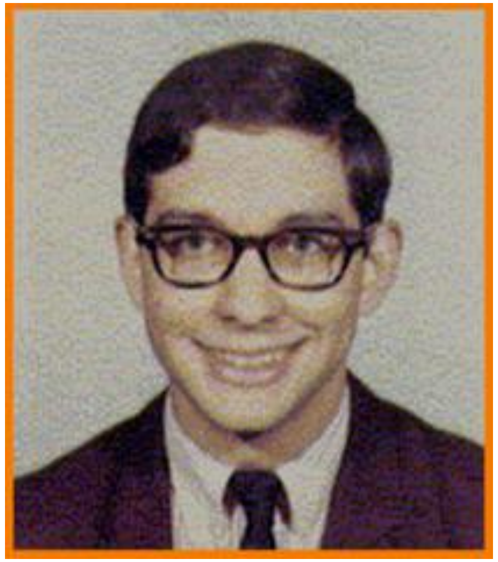
36	109	146
32	36	109
32	36	73

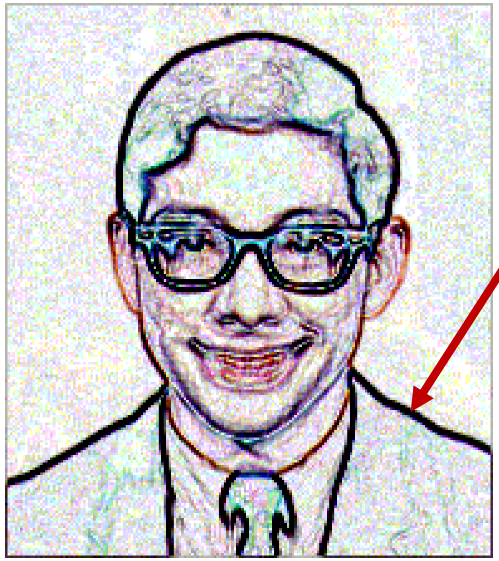
Pixel(x,y).red and its red neighbors

$$\text{filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = **0**

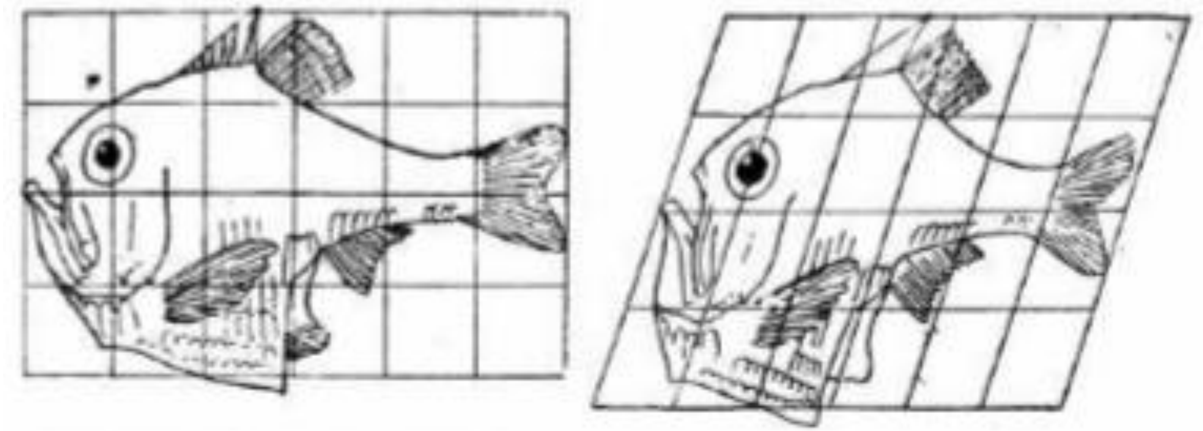


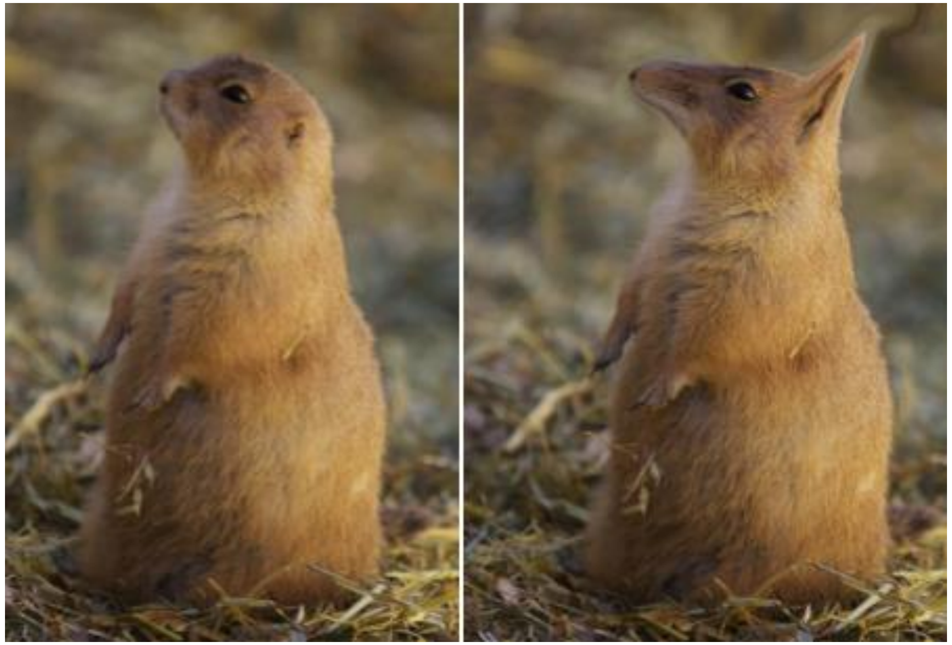


$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

图像变形Image Warping

- 通过操控图像定义域，改变图像中物体的几何形状，实现图像整体或局部的变形

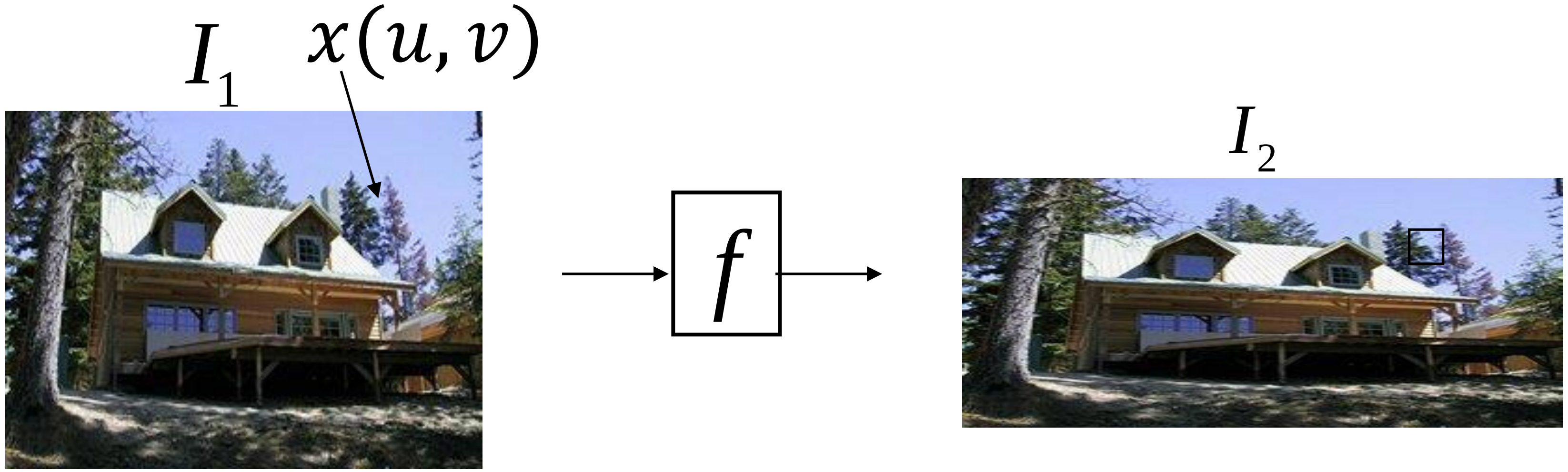


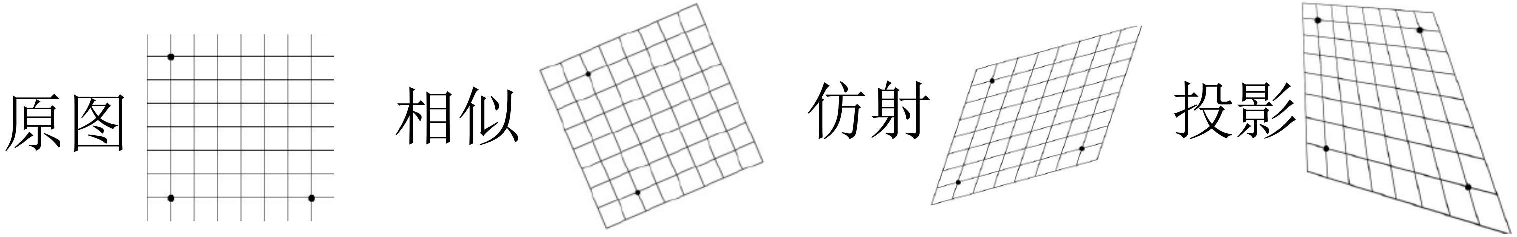


图像变形Image Warping

- 根据变形函数逐像素改变输入图像，生成变形后的图像

$$I_2(f(x(u,v))) = I_1(x(u,v))$$

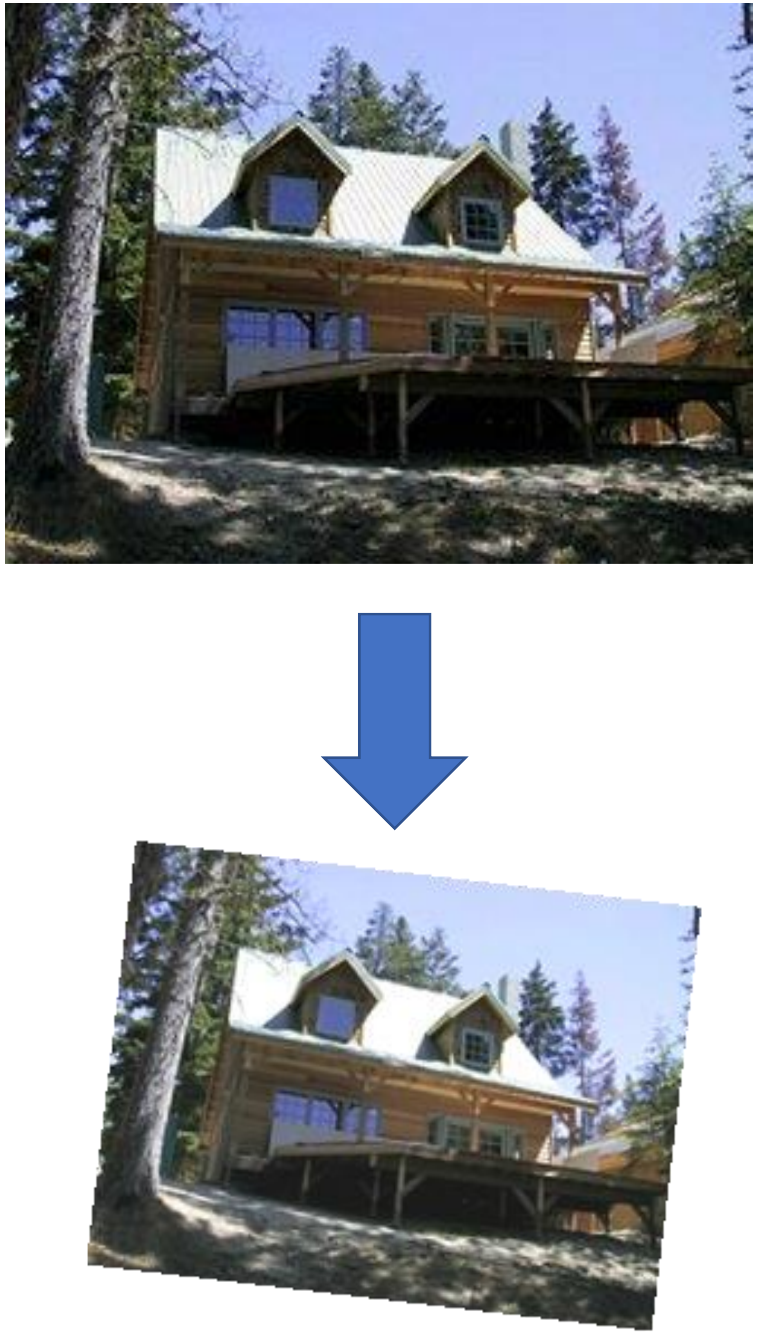




图像变形Image Warping

- 相似变换

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ d \end{bmatrix}$$



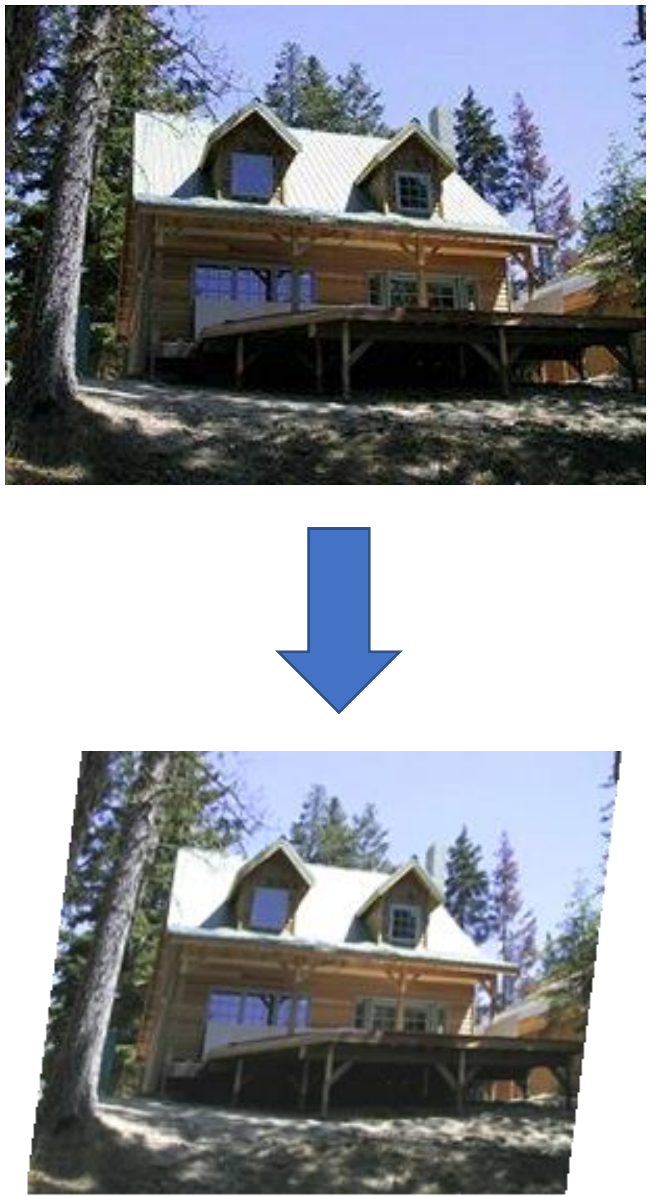
  - 二维缩放、旋转和平移变换的组合。
  - 允许一个正方形被转换成任何旋转的矩形。
  - 线之间的夹角被保留
  - 自由度为4（a，b，c，d）
  - 逆变换是相同的表示（相似性）

图像变形Image Warping

- 仿射变换

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

- 二维缩放、旋转、剪切和平移变换的组合。
- 允许一个正方形被转换成任何平行四边形。
- 自由度为6（a，b，c，d, e, f）
- 逆变换是相同的表示（相似性）

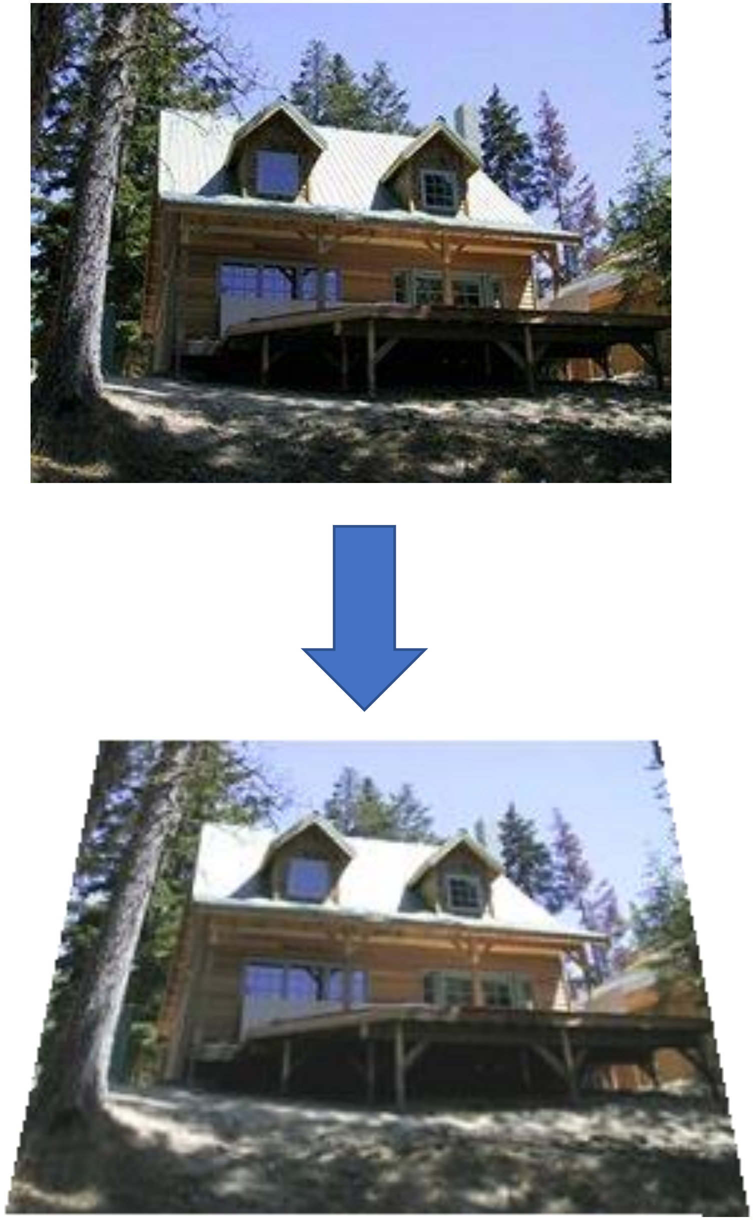


图像变形Image Warping

- 投影变换

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

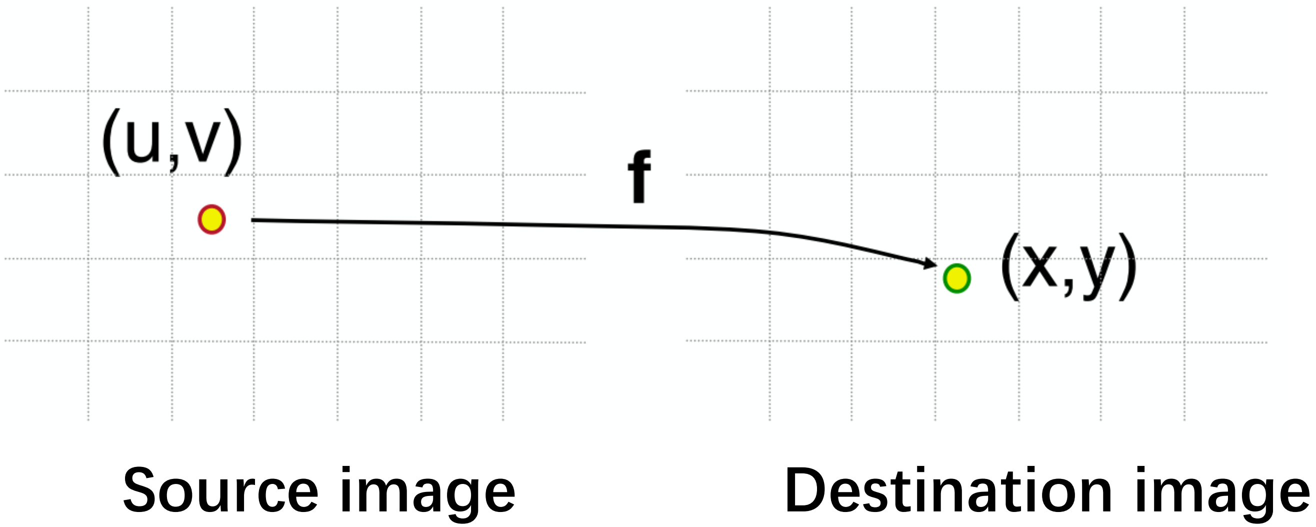
  - 如果$g = h = 0$，那么为一个特殊情况的仿射
  - 允许一个正方形被扭曲成任何四边形
  - 自由度为8（a-h）
  - 逆是同一形式（也就是投影）。



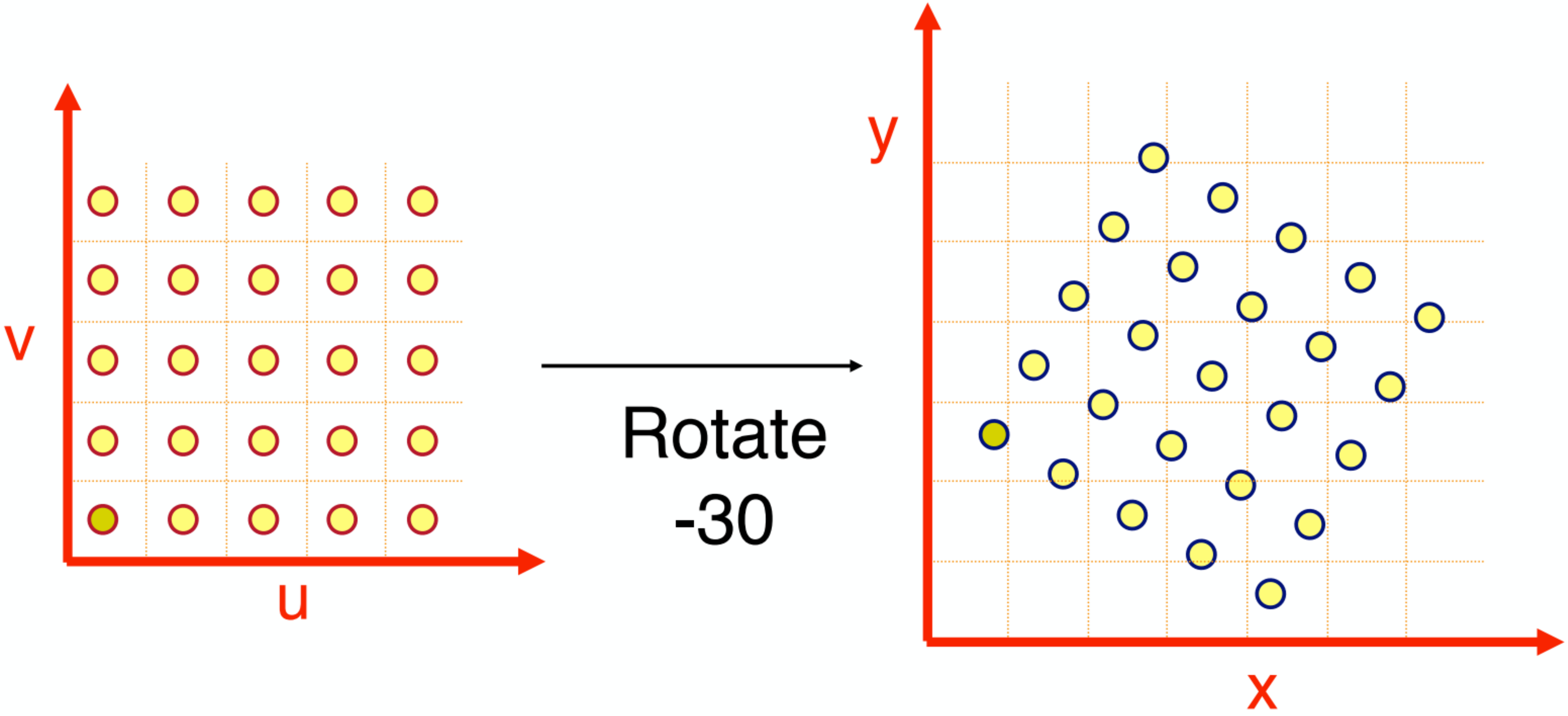
前向映射和逆向映射

基于前向映射的图像变形

```
for (int u = 0; u < umax; u++)
  for (int v = 0; v < vmax; v++){
    float x = fx(u,v);
    float y = fy(u,v);
    dst(x,y) = src(u,v);
  }
```



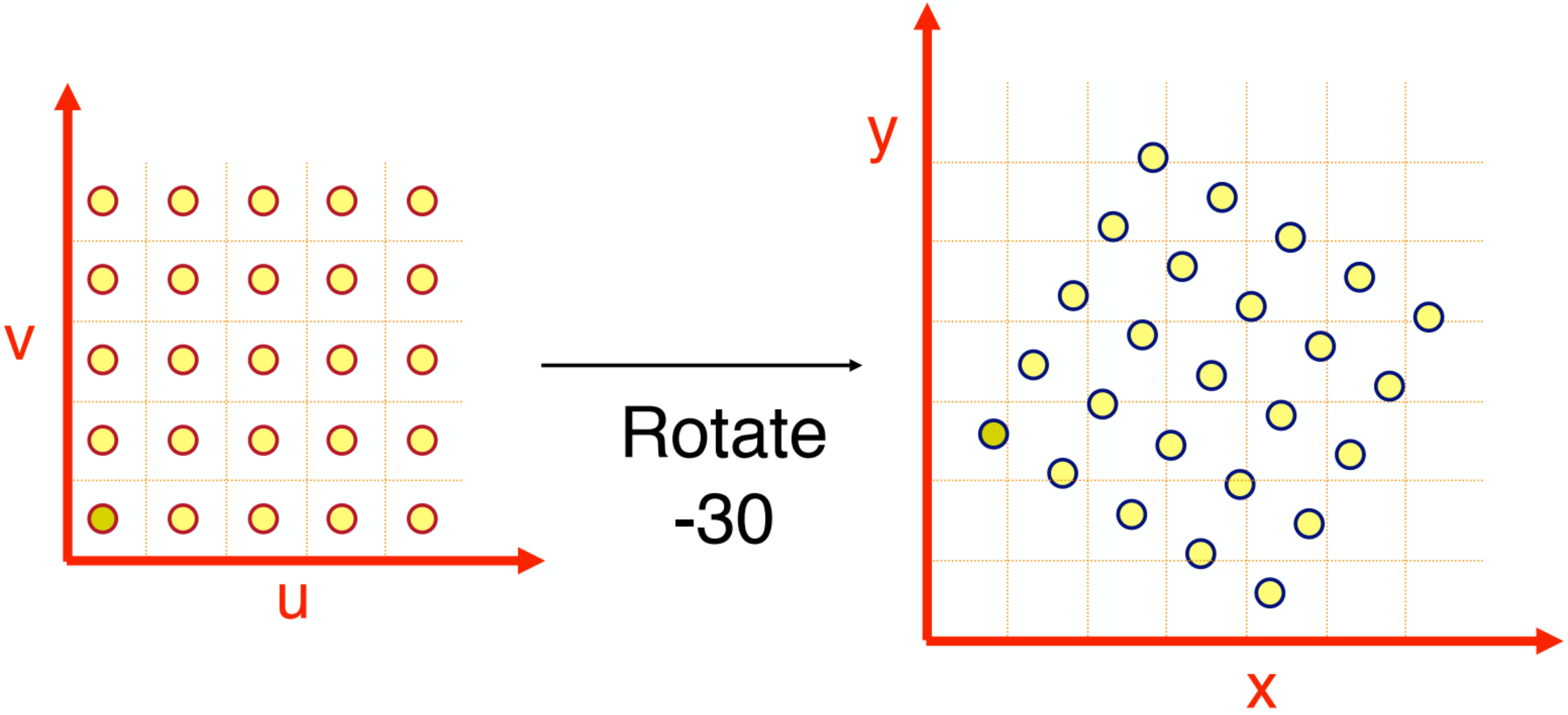
前向映射



Rotate -30

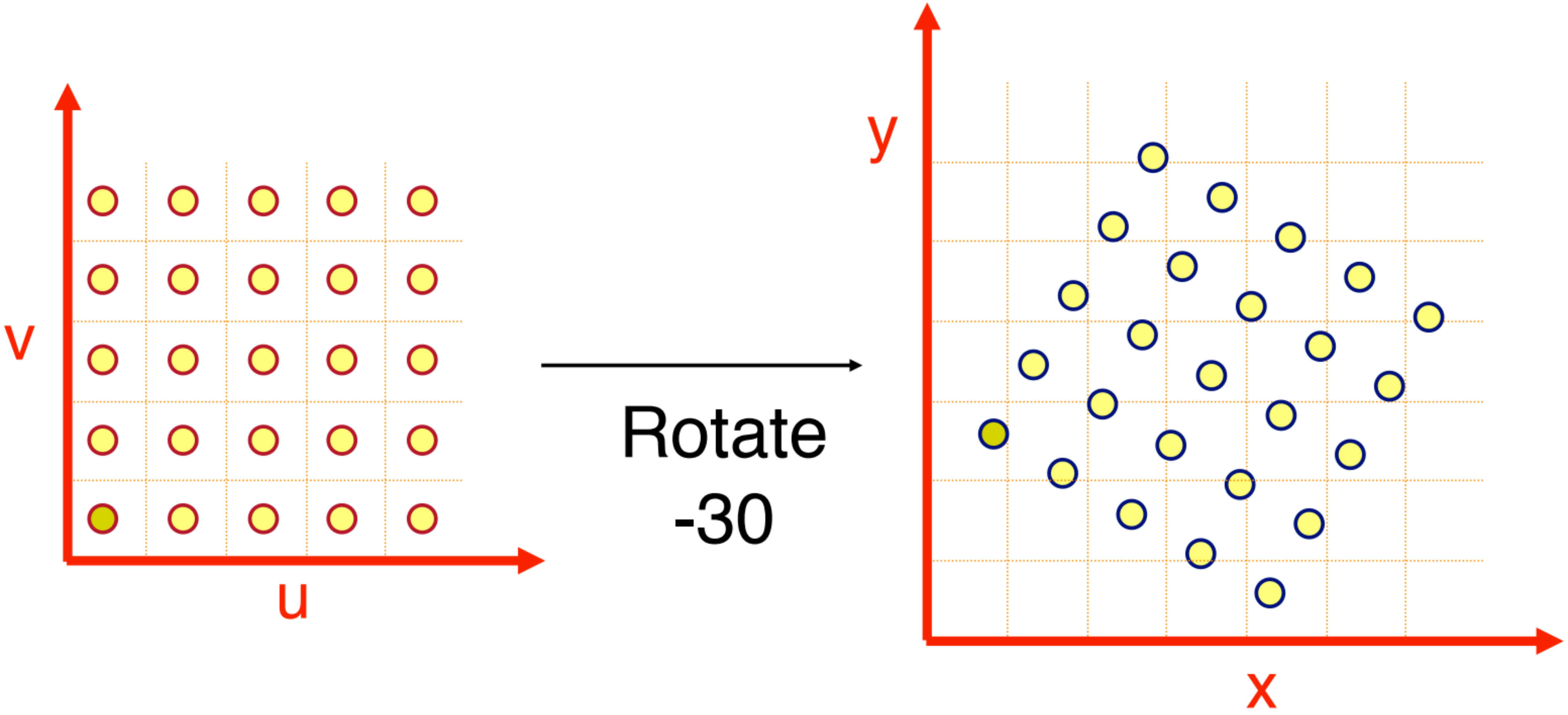
前向映射

Multiple source pixels may be mapped to same destination pixel



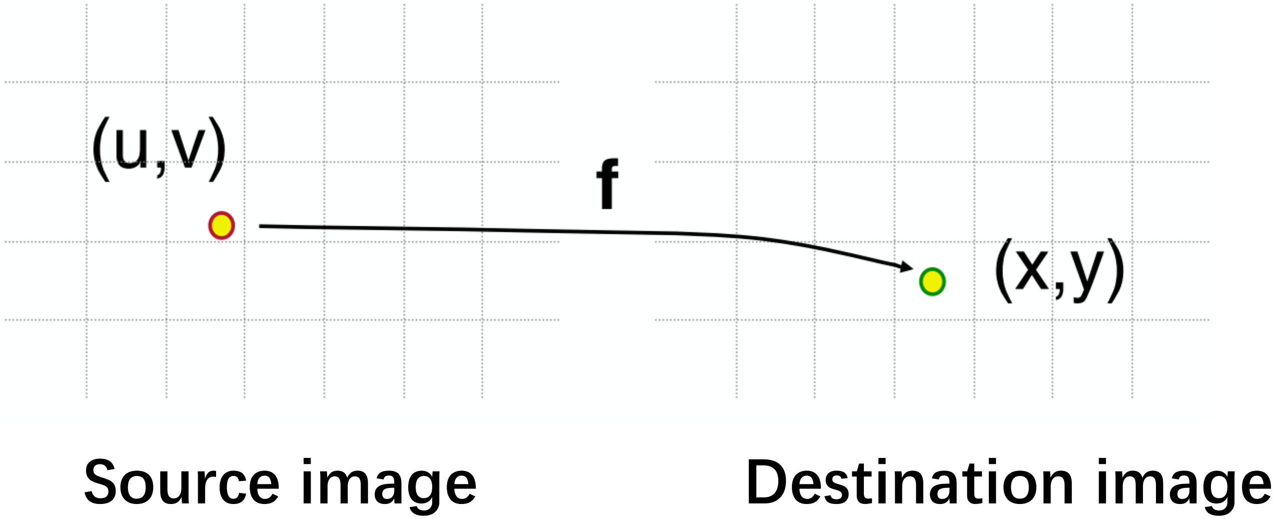
前向映射

Some destination pixels may not be covered



基于逆向映射的图像变形

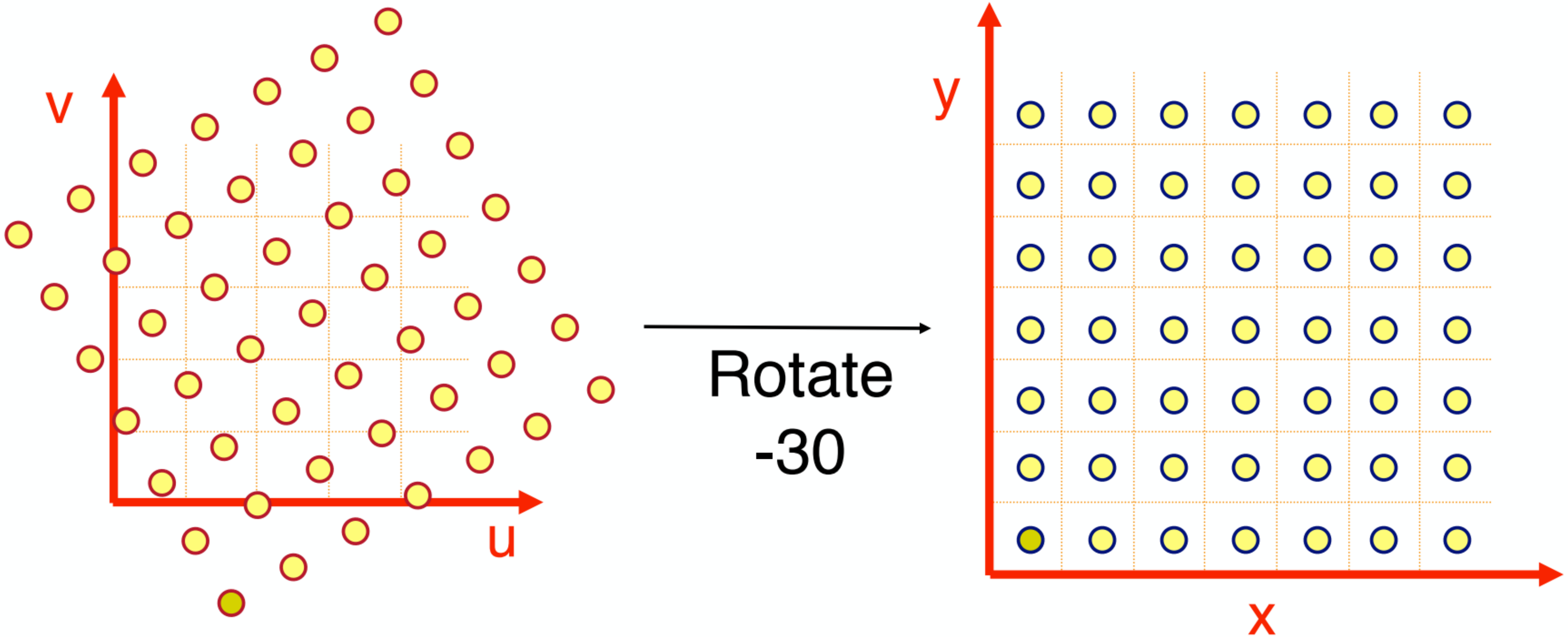
```
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++){
    float u = $fx^{-1}(x,y)$;
    float v = $fy^{-1}(x,y)$;
    dst(x,y) = src(u,v);
  }
```



逆向映射 - 完美!

Iterate over destination image

- Must resample source
- May oversample, but much simpler!



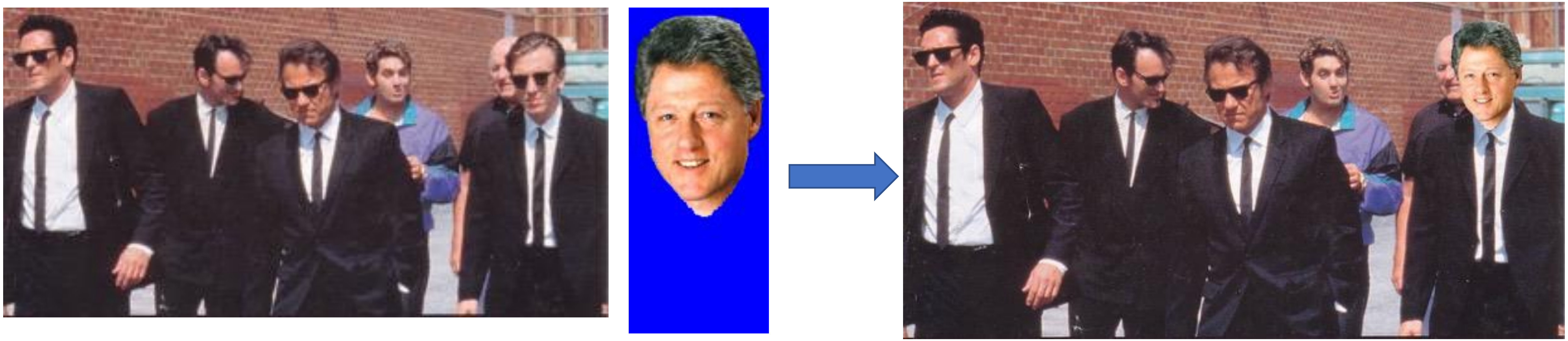
图像合成Image Compositing

- Separate an image into “elements”
  - Render independently
  - Composite together
- Applications
  - Cel animation
  - Chroma-keying
  - Blue-screen matting

Bill makes ends meet by going into film

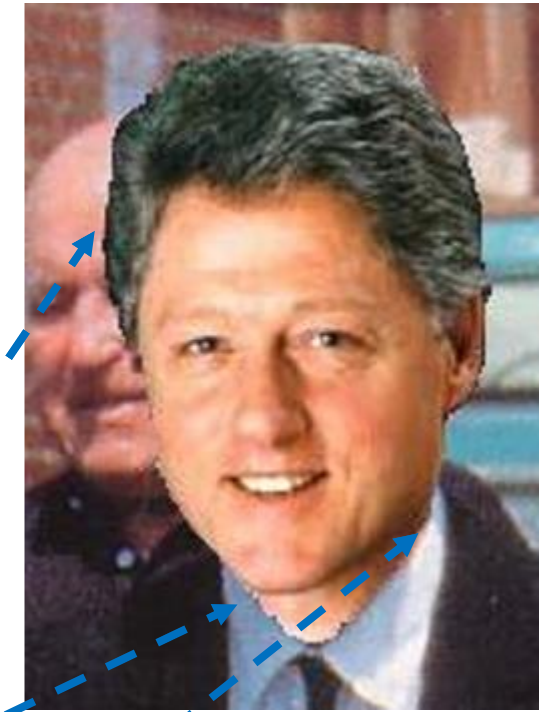
Blue-Screen Matting蓝幕抠图

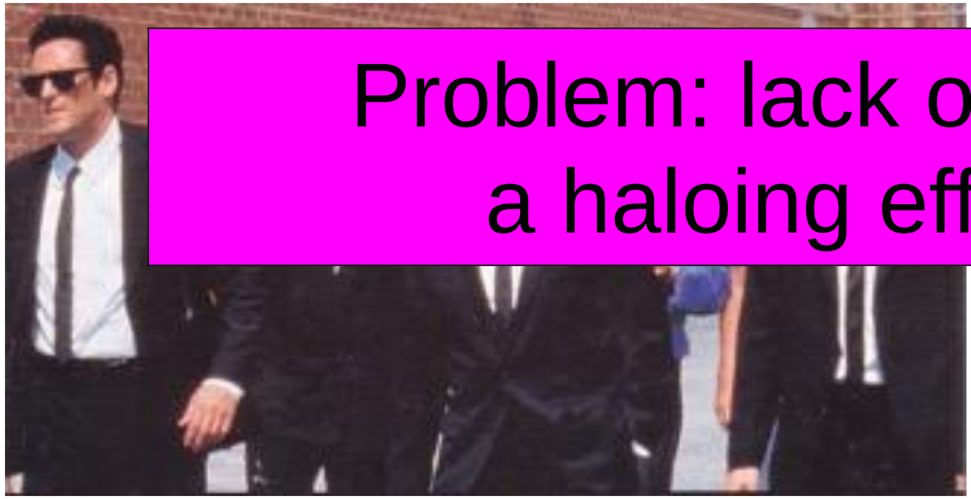
- Composite foreground and background images
  - Create background image
  - Create foreground image with blue background
  - Insert non-blue foreground pixels into background

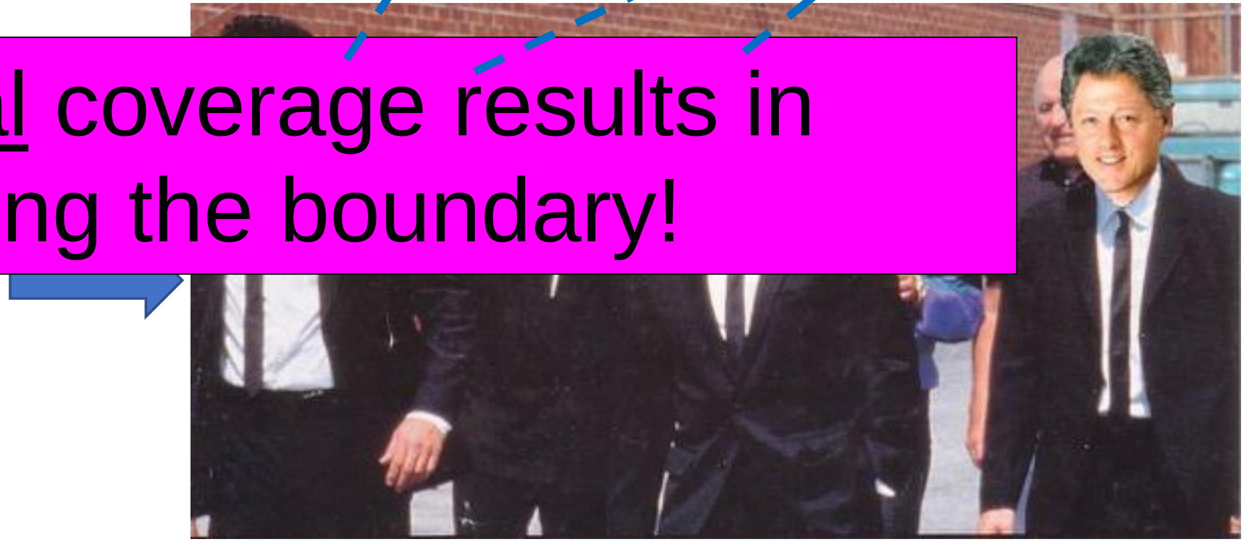


Blue-Screen Matting蓝幕抠图

- Composite foreground and background images
  - Create background image
  - Create foreground image with blue background
  - Insert non-blue foreground pixels into background





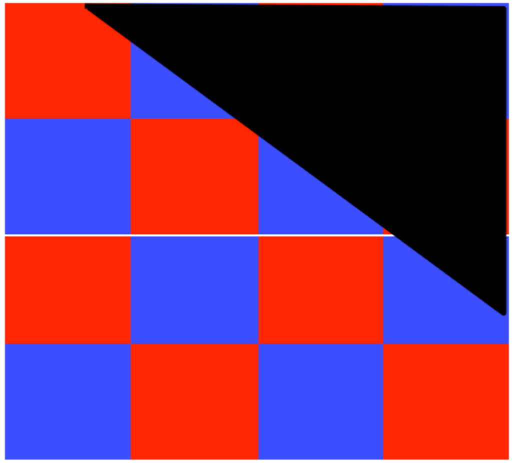


Problem: lack of partial coverage results in a haloing effect along the boundary!

Alpha通道

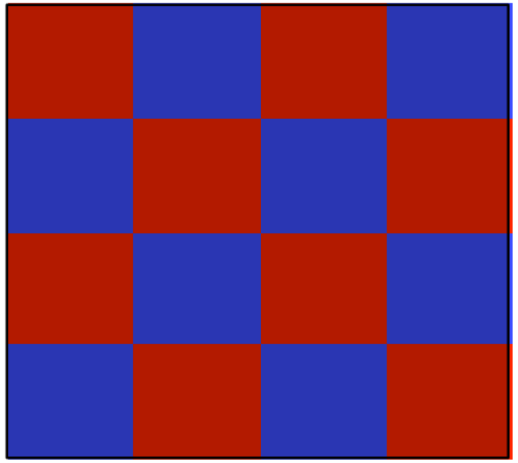
- Encodes pixel coverage information
  - $\alpha = 0$: no coverage (or transparent)
  - $\alpha = 1$: full coverage (or opaque)
  - $0 < \alpha < 1$: partial coverage (or semi-transparent)

- • Single Pixel Example: $\alpha = 0.3$



Partial Coverage

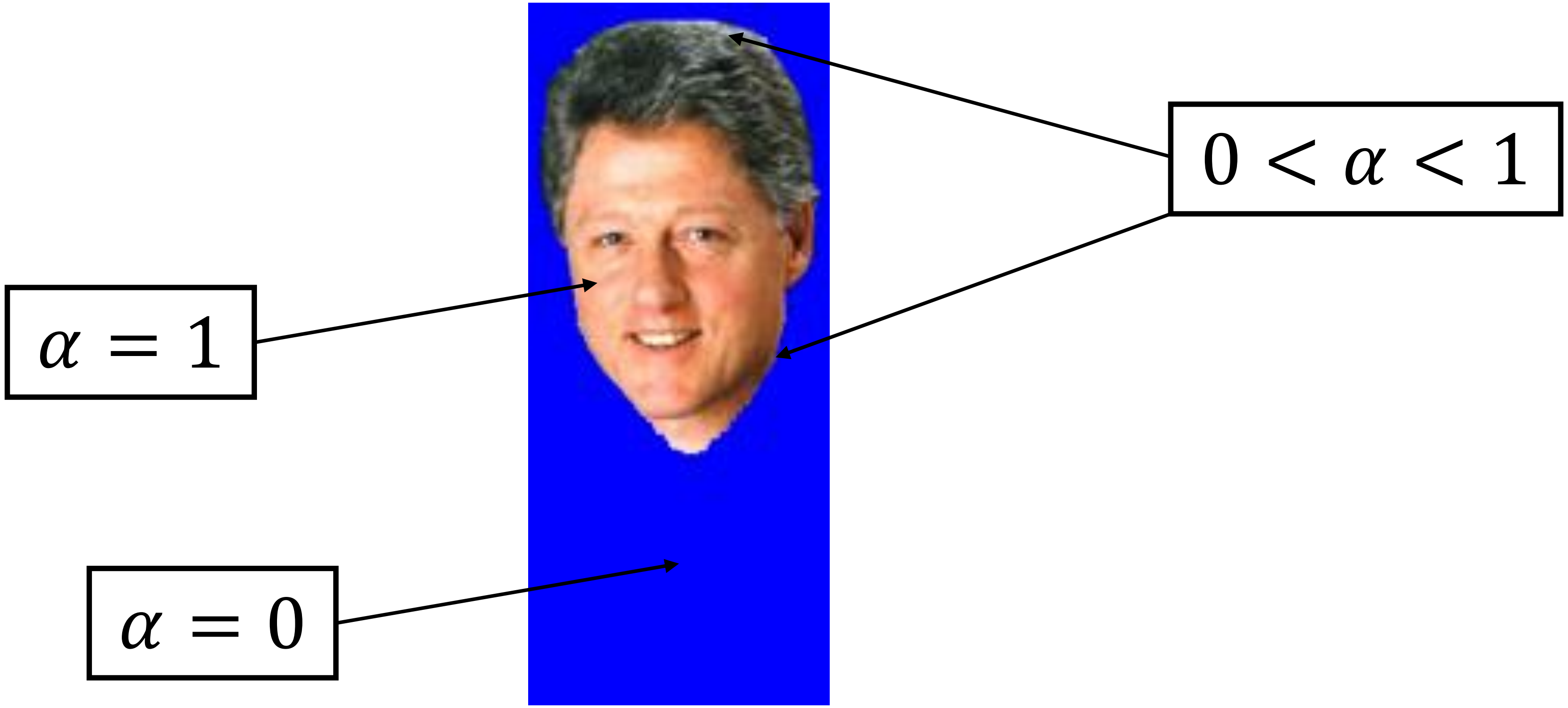
or



Semi-Transparent

Compositing with Alpha

- Controls the linear interpolation of foreground and background pixels when elements are composited



$0 < \alpha < 1$

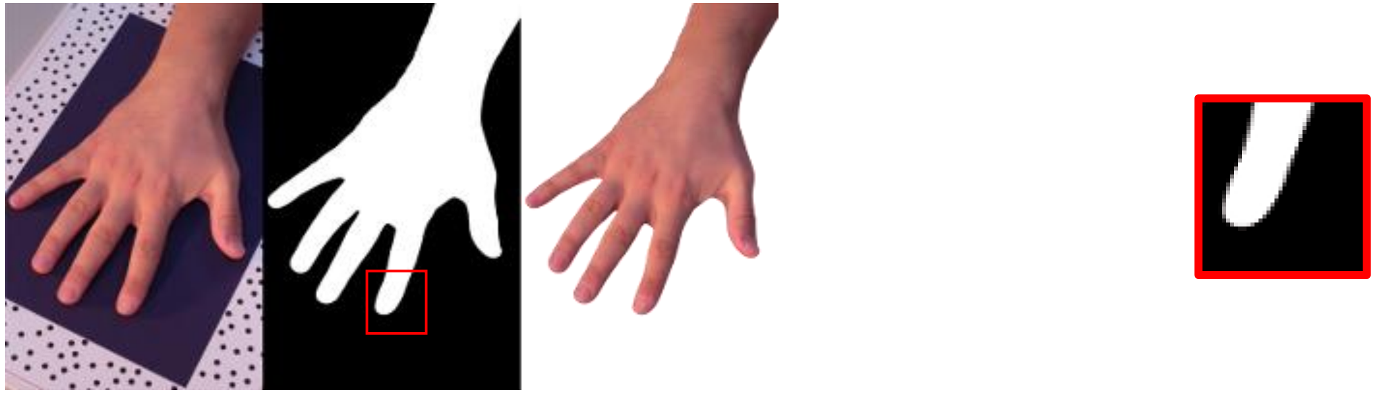
$\alpha = 1$

$\alpha = 0$

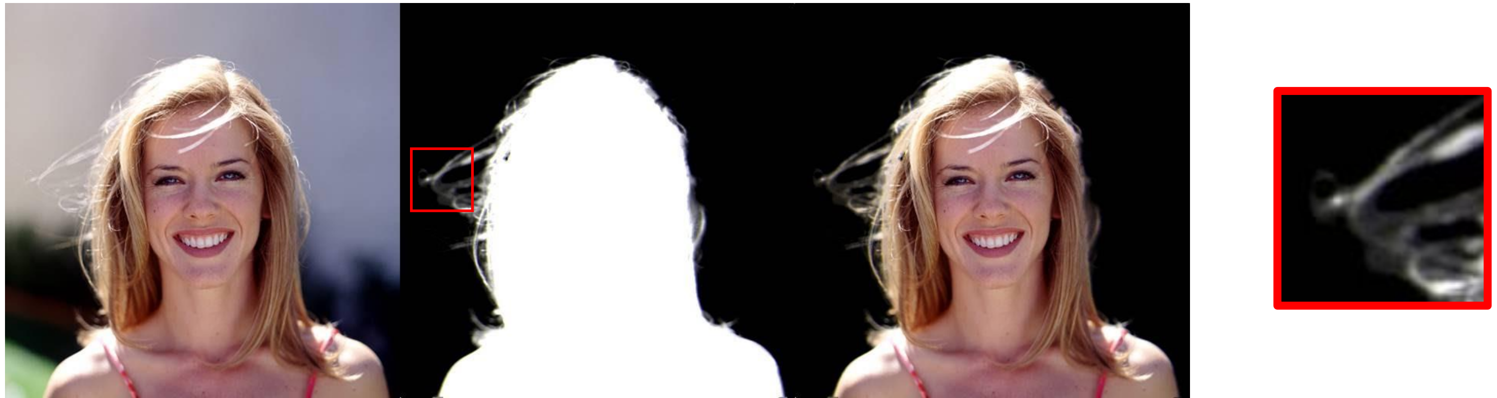
Matting and Segmentation 分割与抠图

- 将图像或视频划分为多个区域的过程

➢ **分割（segmentation）**: 硬分割 ‘0’ 或‘1’

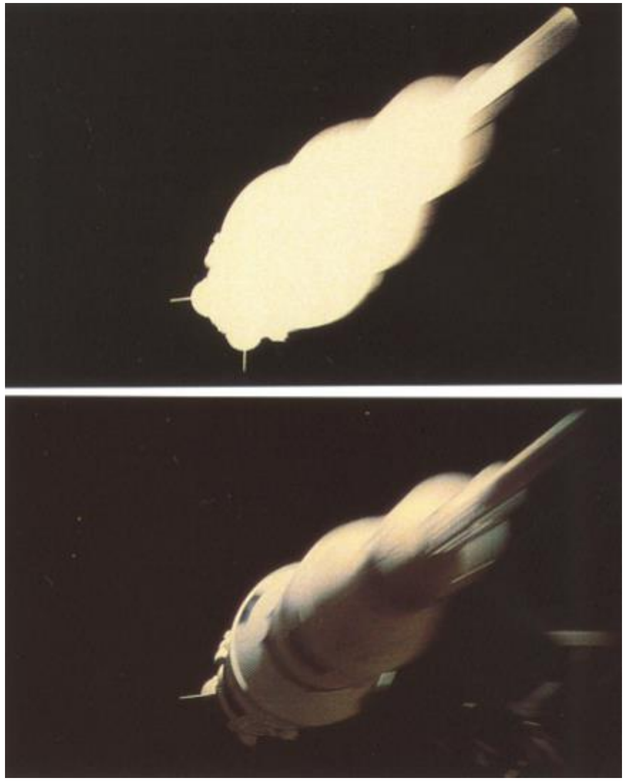


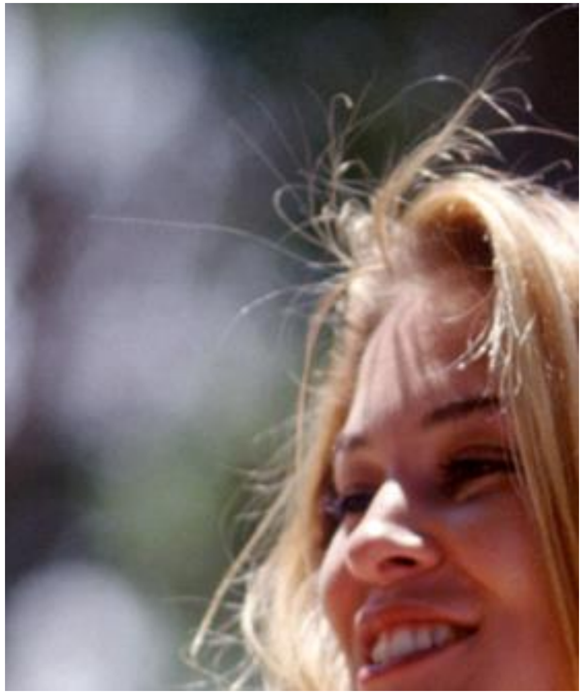
➢ **抠图（matting）**: 软分割 ‘0~1’

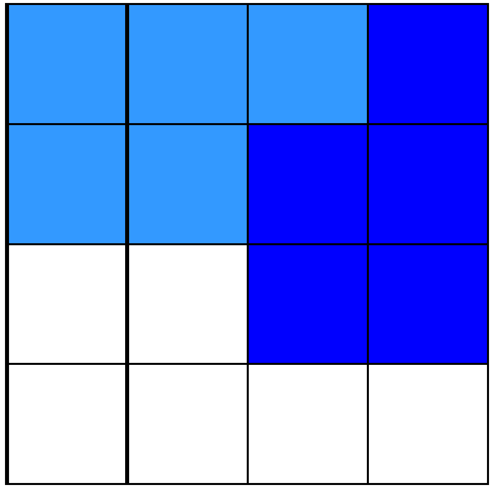


Matting and Segmentation 分割与抠图

- 对于微小特征的物体，抠图提供了更精细的划分方式
  - 运动模糊或者微小特征，比如头发引起的像素的部分遮挡

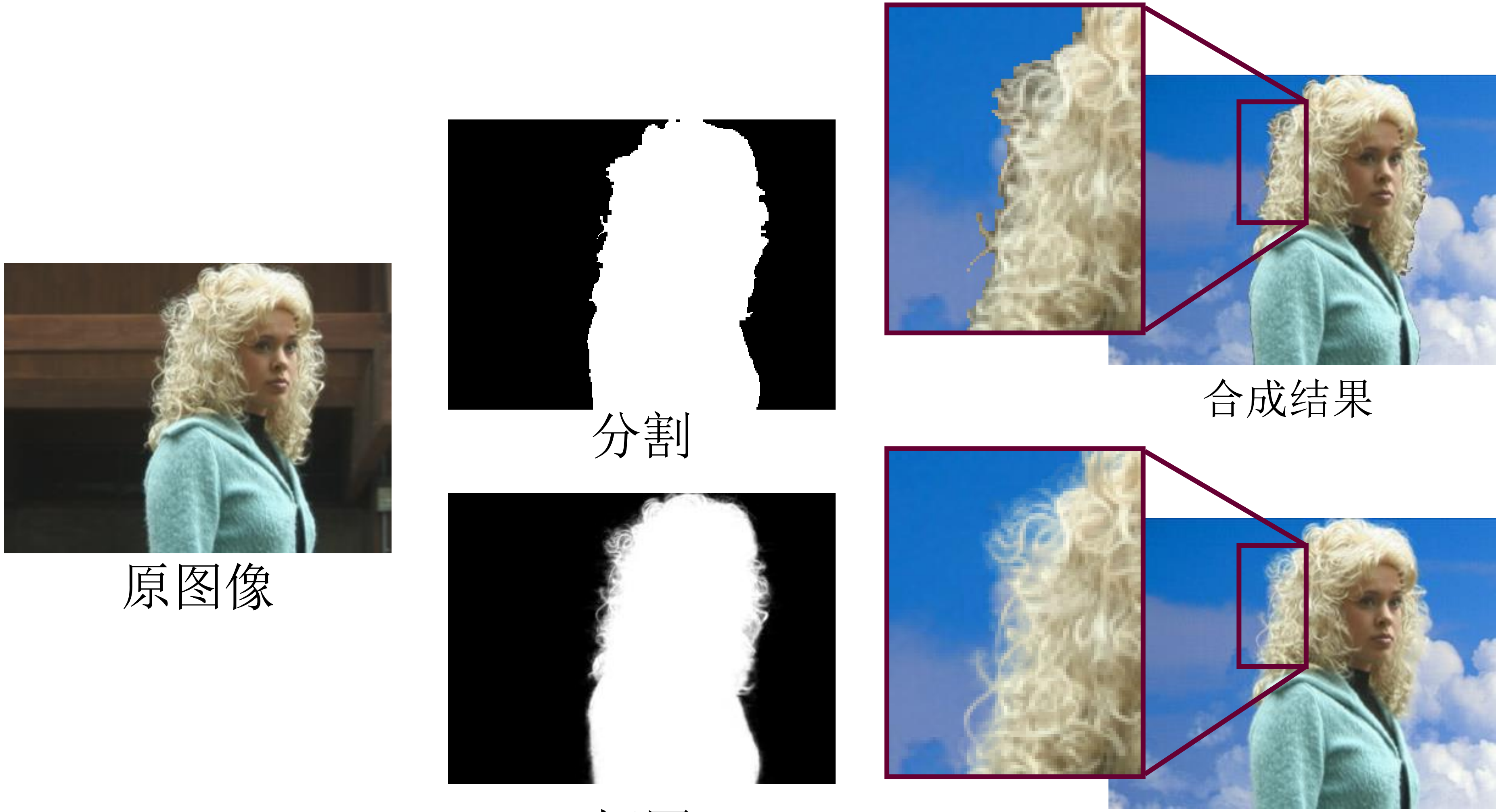






像素 ⇨ 超级采样像素

Matting and Segmentation 分割与抠图



原图像

分割

抠图

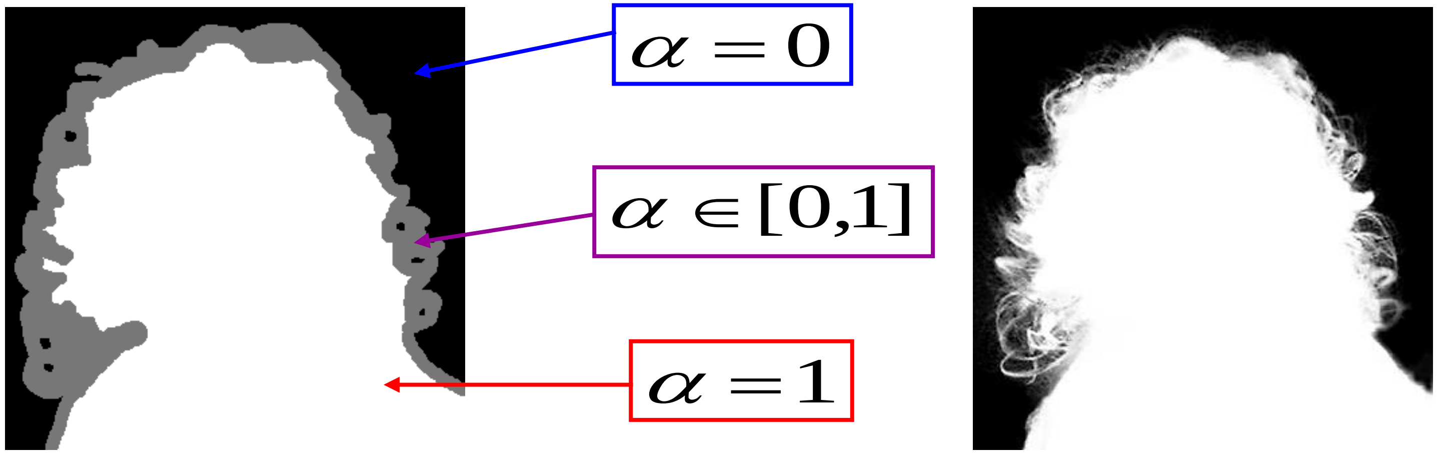
合成结果

合成结果

抠图定义

- 对每个像素I_z赋予 ‘0~1’的值
  - $\alpha_z = 0$: 明确的背景
  - $\alpha_z = 1$: 明确的前景
  - 否则是混合的

$$\mathrm{I}_z = \alpha_z \mathrm{F}_z + (1 - \alpha_z)\mathrm{B}_z$$



$\alpha = 0$

$\alpha \in [0,1]$

$\alpha = 1$

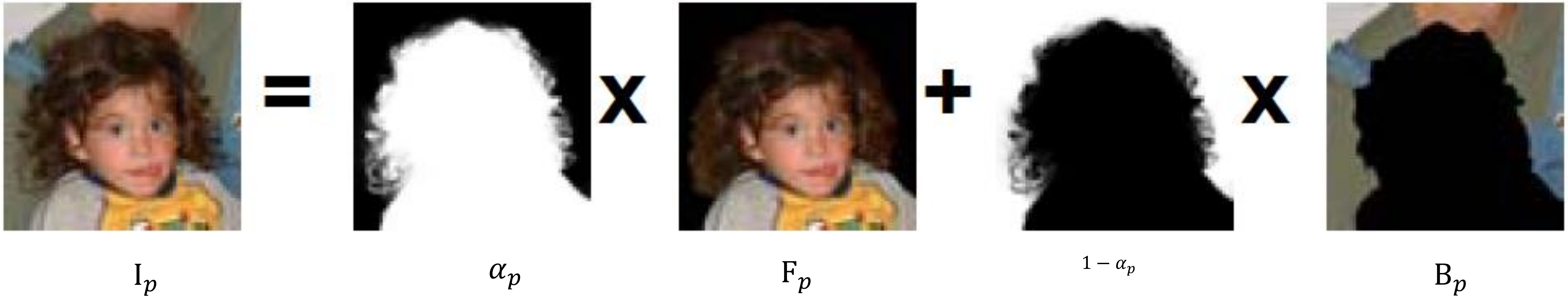
抠图定义

- 高度“病态”问题：对每个像素而言，有7个未知量，但是只有3个方程

$$\mathrm{I}_z = \alpha_z \mathrm{F}_z + (1 - \alpha_z)\mathrm{B}_z$$

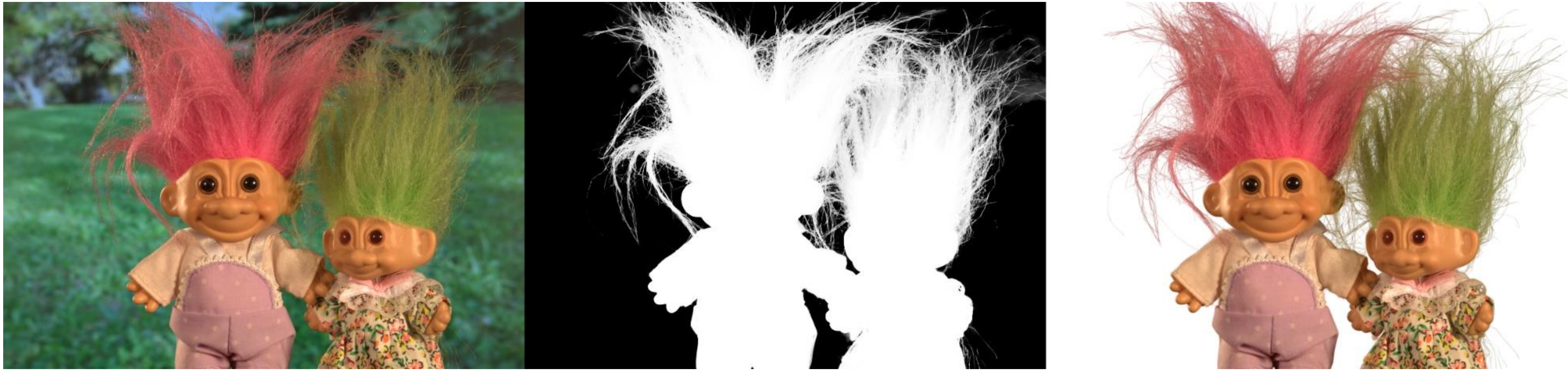
⇨

$$\begin{cases} \mathrm{I}_{R_1} = \alpha \mathrm{F}_R + (1 - \alpha)\mathrm{B}_{R_1} \\ \mathrm{I}_{G_1} = \alpha \mathrm{F}_G + (1 - \alpha)\mathrm{B}_{G_1} \\ \mathrm{I}_{B_1} = \alpha \mathrm{F}_B + (1 - \alpha)\mathrm{B}_{B_1} \end{cases}$$



策略

- 引入先验知识作为约束，将病态问题转化为可求解问题
- 减少未知量的数目，最优化抠图函数

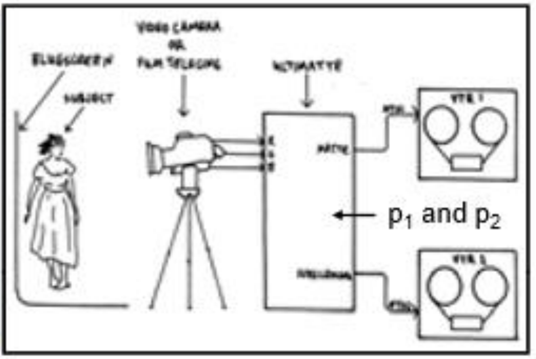


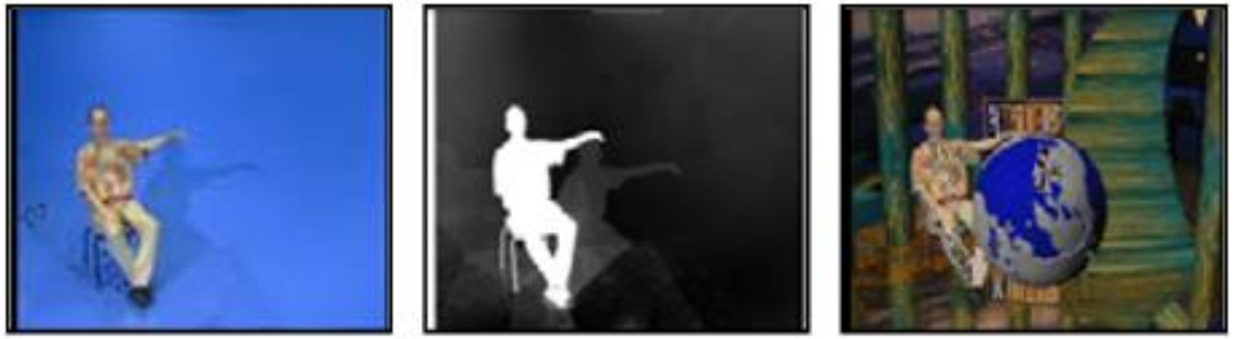
图像抠图方法

- 蓝屏抠图（背景已知）
- 自然图像抠图（背景未知）

蓝屏抠图

- **思想：**指定单一的背景颜色，将待抠取的物体置于背景前面
- **方法：**早期采用蓝色背景，后来绿色更为流行
  - 50年代，Petros Vlahos发明了蓝屏抠图ultimatte®，曾获奥斯卡终身成就奖





蓝屏抠图

- 背景颜色和前景部分颜色作为已知条件

  $B_R = 0, B_G = 0$

- 抠图函数：对于每个像素具有4个未知量，3个方程

$$\begin{cases} \mathrm{I}_R = \alpha \mathrm{F}_R + (1-\alpha)\mathrm{B}_R \\ \mathrm{I}_G = \alpha \mathrm{F}_G + (1-\alpha)\mathrm{B}_G \\ \mathrm{I}_B = \alpha \mathrm{F}_B + (1-\alpha)\mathrm{B}_B \end{cases} \Rightarrow \left\{ \boxed{\begin{matrix} \mathrm{I}_R \\ \mathrm{I}_G \\ \mathrm{I}_B \end{matrix}} \begin{matrix} = \alpha \mathrm{F}_R + (1-\alpha) \\ = \alpha \mathrm{F}_G + (1-\alpha) \\ = \alpha \mathrm{F}_B + (1-\alpha) \end{matrix} \boxed{\begin{matrix} 0 \\ 0 \\ \mathrm{B} \end{matrix}} \right.$$

已知　　　　　　已知

蓝屏抠图

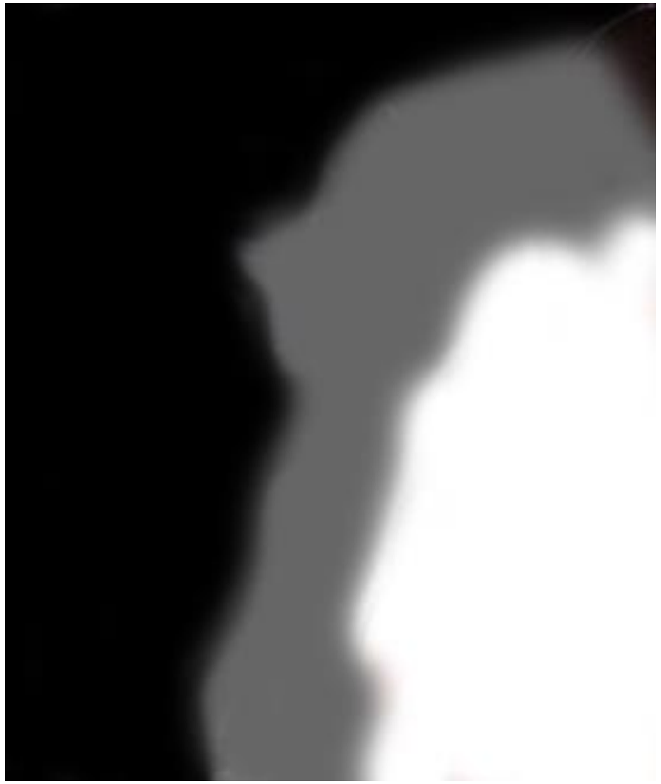
- 如果前景物体不含B通道颜色$F_B = 0$
- 简化方程为3个未知数
- 依次计算α, F_R, F_G

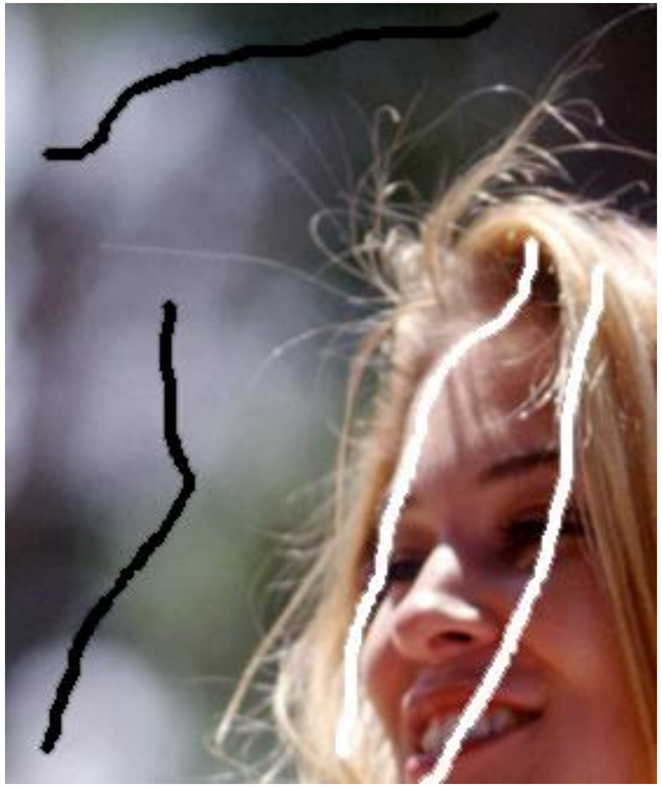
$$\begin{cases} \mathrm{I}_R = \alpha \mathrm{F}_R \\ \mathrm{I}_G = \alpha \mathrm{F}_G \\ \mathrm{I}_B = \alpha 0 + (1 - \alpha)\mathrm{B} \end{cases} \Leftarrow \begin{cases} \mathrm{I}_R = \alpha \mathrm{F}_R + (1 - \alpha)0 \\ \mathrm{I}_G = \alpha \mathrm{F}_G + (1 - \alpha)0 \\ \mathrm{I}_B = \alpha \mathrm{F}_B + (1 - \alpha)\mathrm{B} \end{cases}$$

已知　　已知

自然图像抠图

- 输入：1个输入图像，背景未知
- 先验：用户交互
- 优化
  - 贝叶斯
  - 泊松
  - 最小二乘法
  - 闭形式
  - …

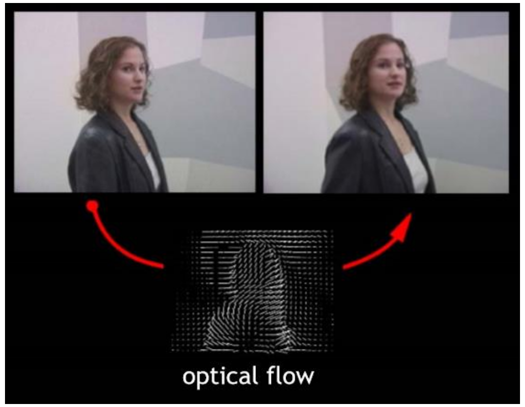




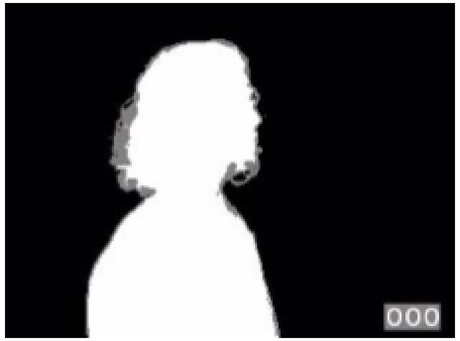
视频抠图

- **思想：** 结合视频运动在帧间传播抠图值

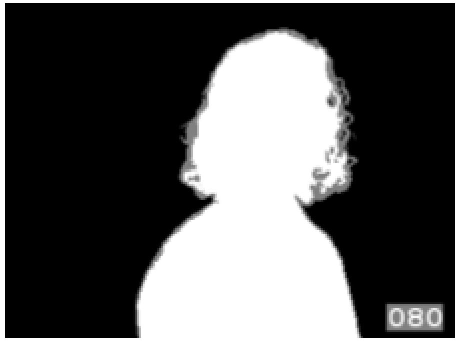
输入视频



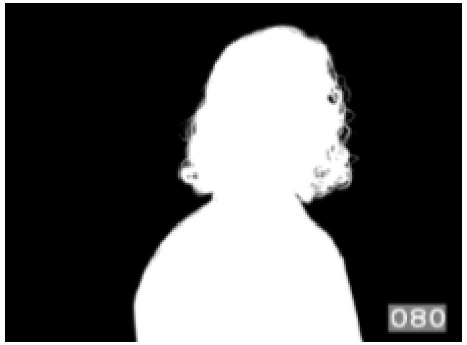
关键帧图



插值图



α



视频抠图

- **方法：** 以关键帧的抠图值作为初始值，通过双向传播插值中间帧的抠图值，并进一步精细处理，生成中间帧抠图结果

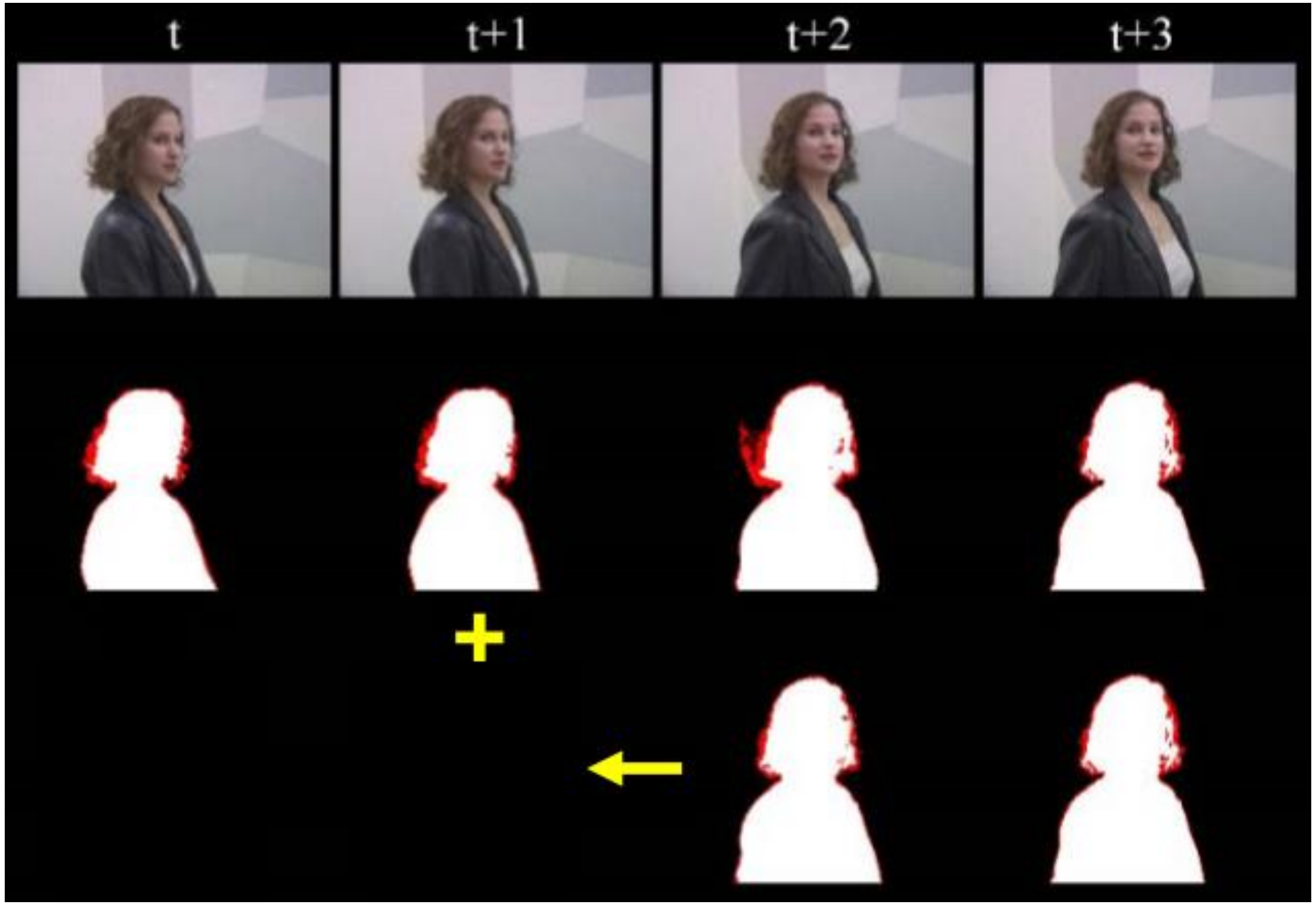
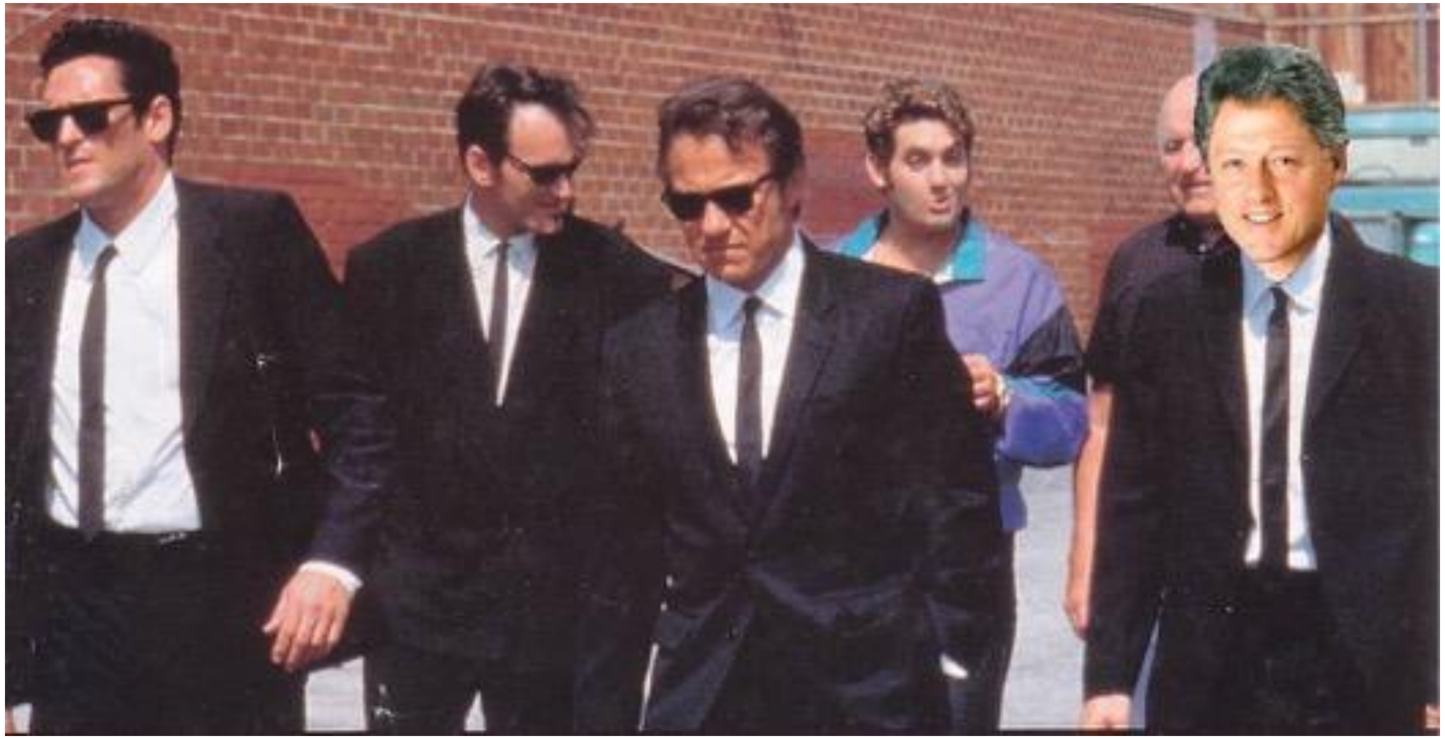


Image Composition “Goofs”

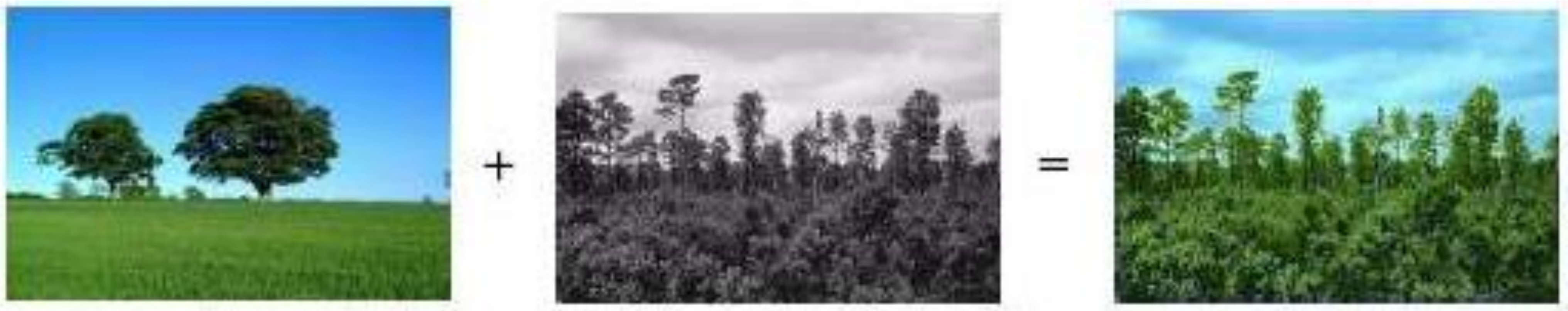
- Visible hard edges
- Incompatible lighting/shadows
- Incompatible camera focal lengths

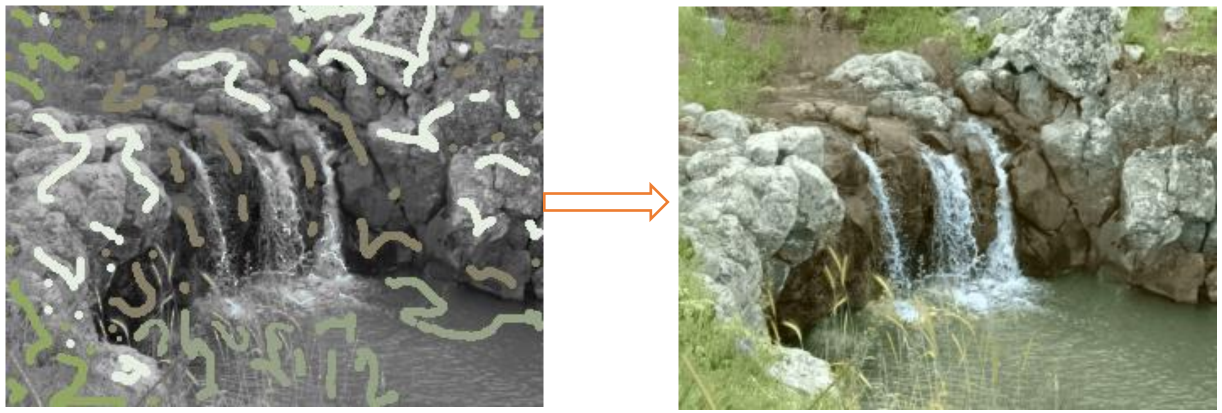


颜色迁移、编辑传播

颜色迁移

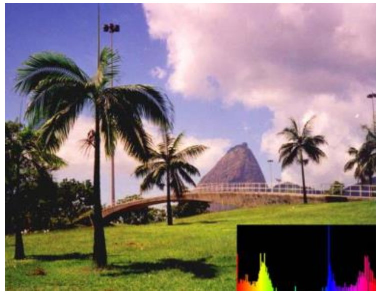
- 从其他图像或用户交互中提供的色彩作为模板，修正目标图像的颜色，使其满足模板色彩

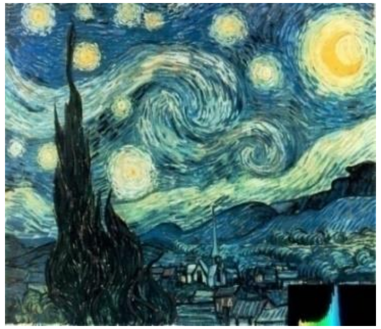


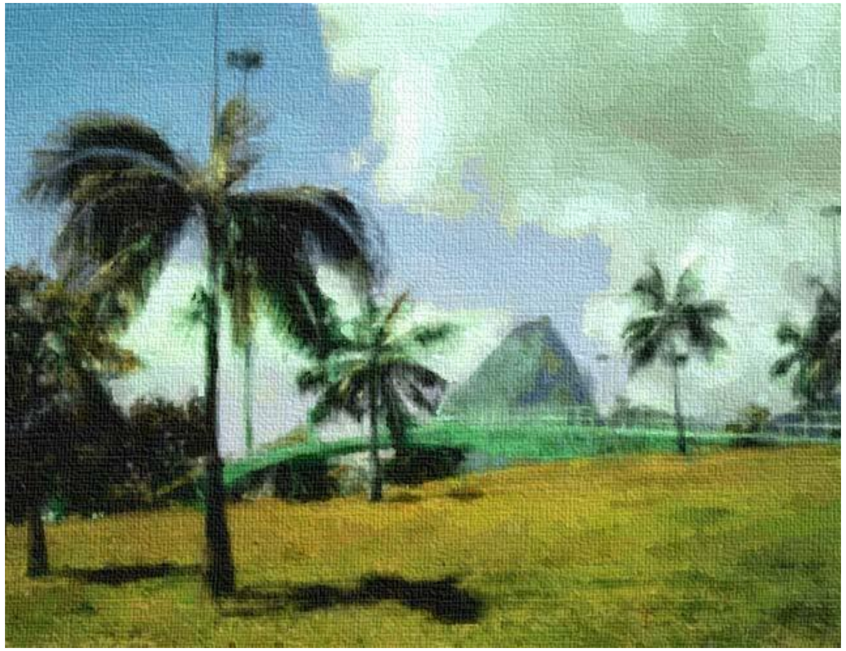


颜色迁移

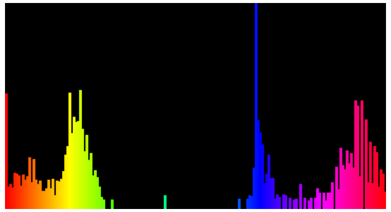
- **方法：** 在色相、色温等颜色特征空间按照图像内容的连续性进行迁移



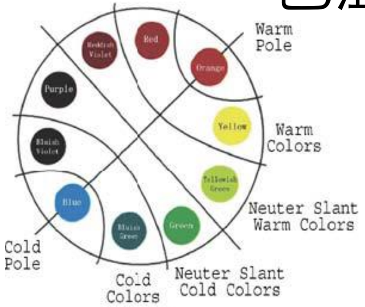


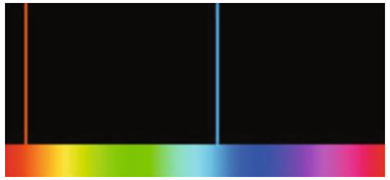


色相



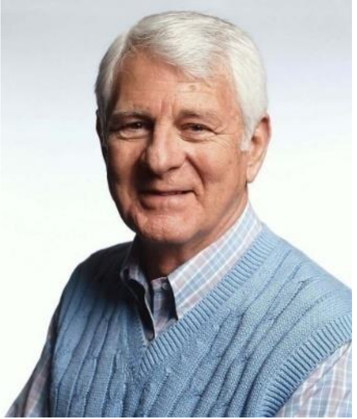
色温



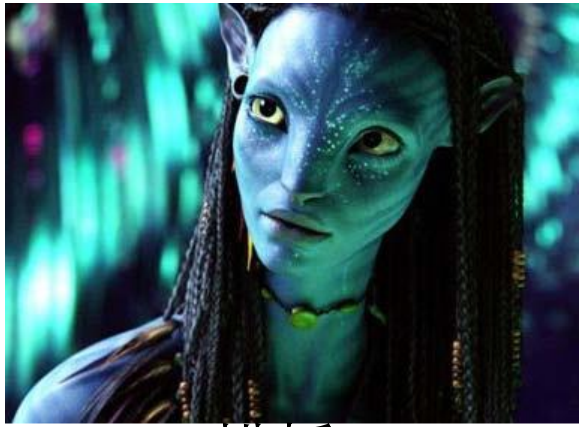


颜色迁移

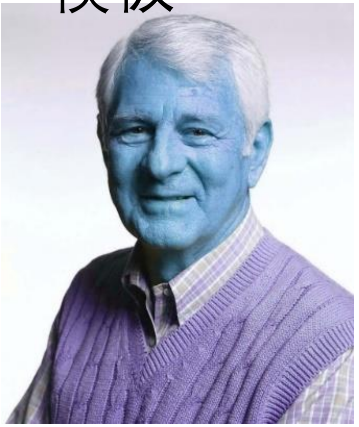
- **方法:** 基于特征匹配的颜色迁移



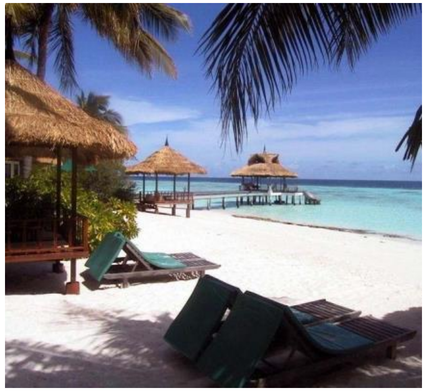
输入



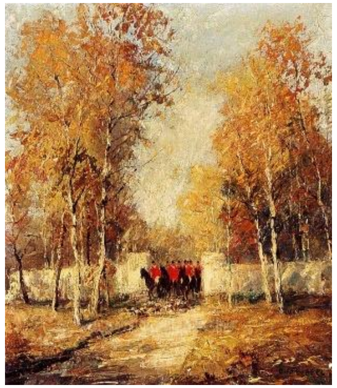
模板



结果

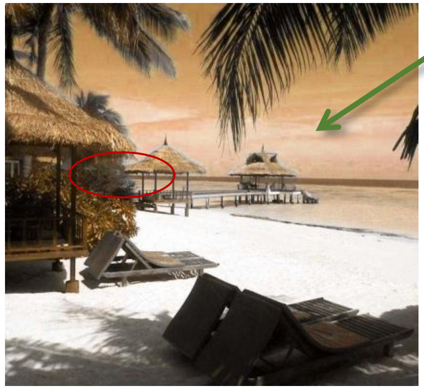


输入



模板

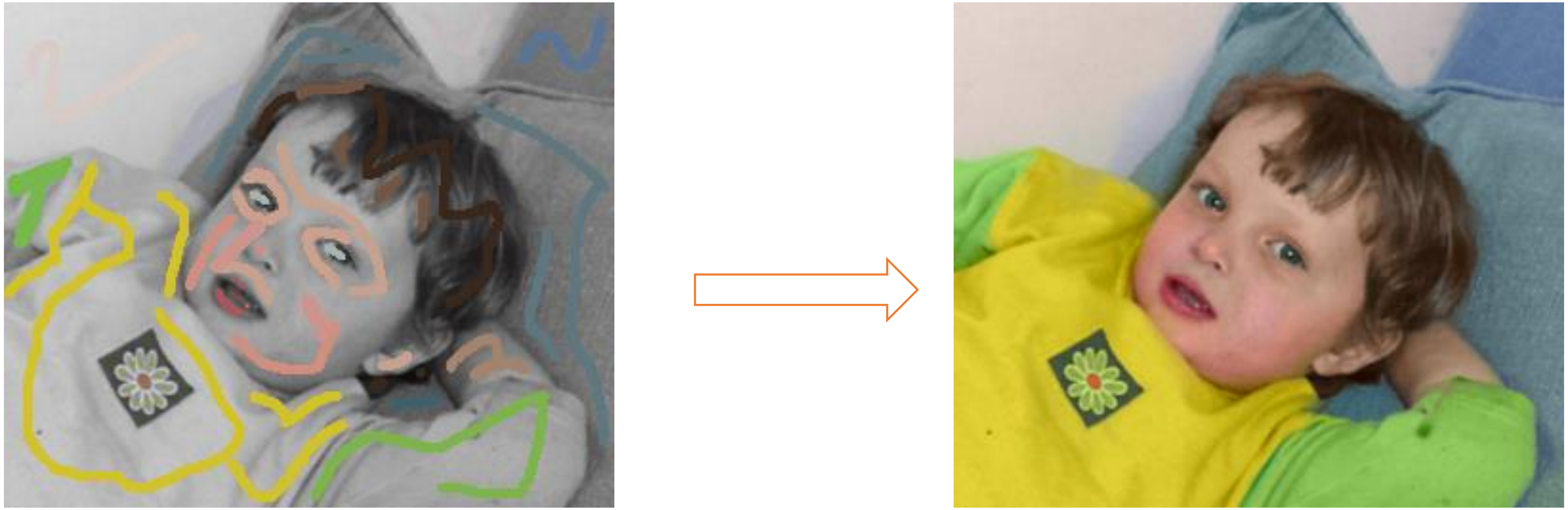
失败的结果



Yellow sky

颜色迁移

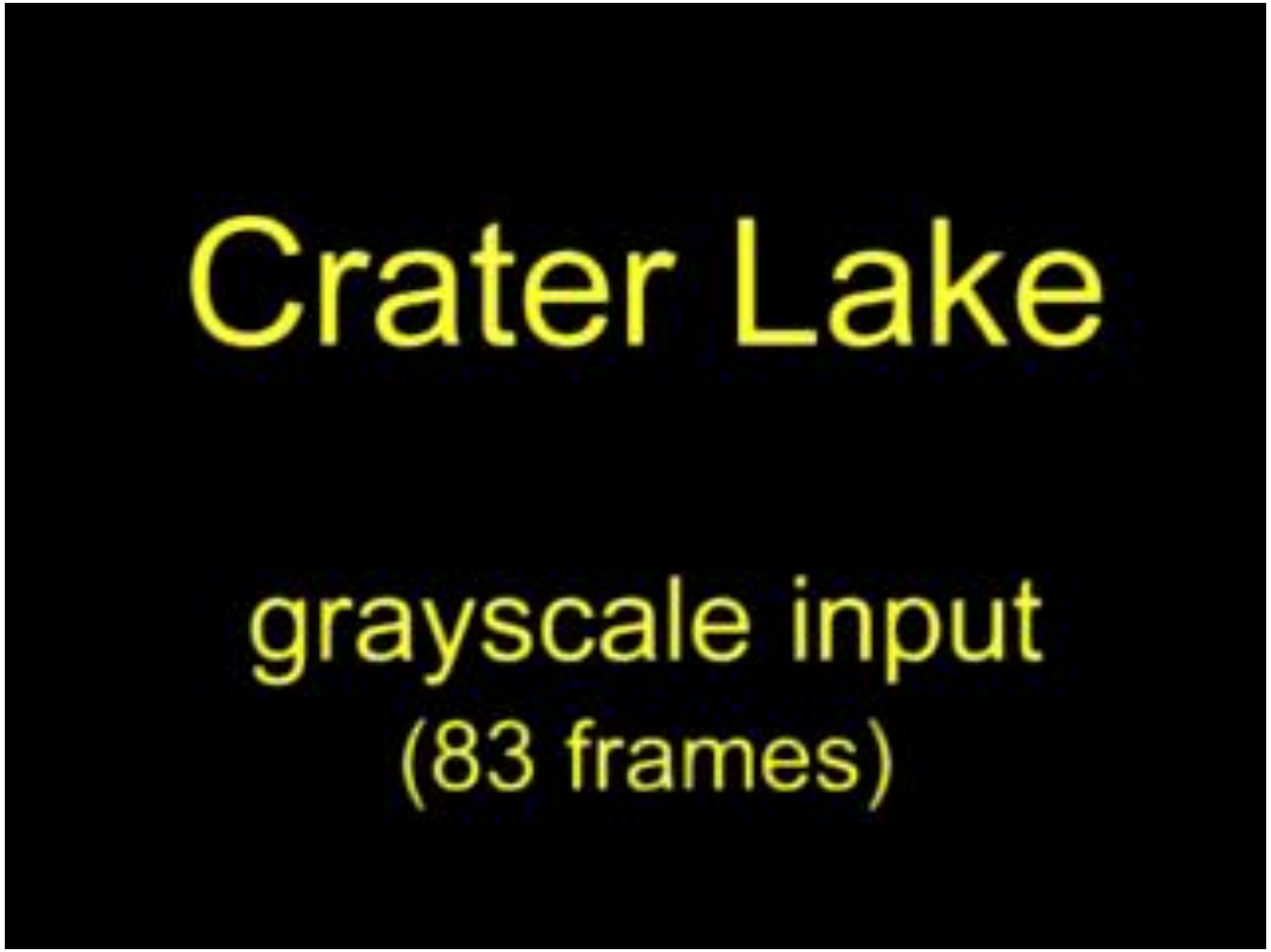
- **方法:** 基于笔画约束的颜色迁移



用户交互的笔画颜色值作为种子，迁移至对应的区域（灰度图着色）

颜色迁移

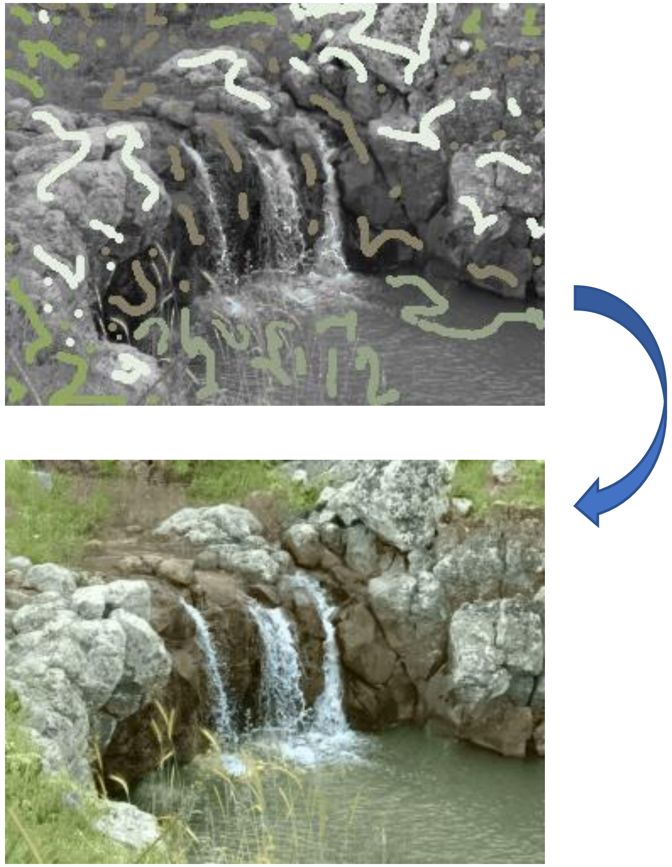
- **方法:** 基于笔画约束的颜色迁移



Crater Lake

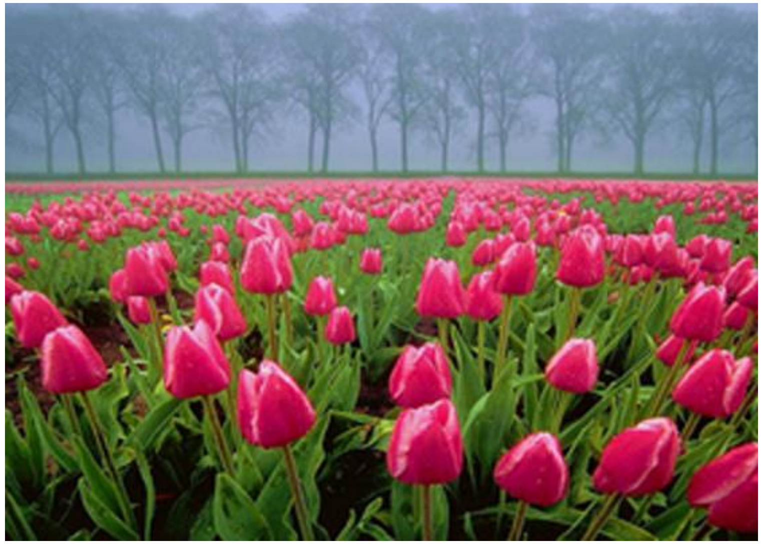
grayscale input

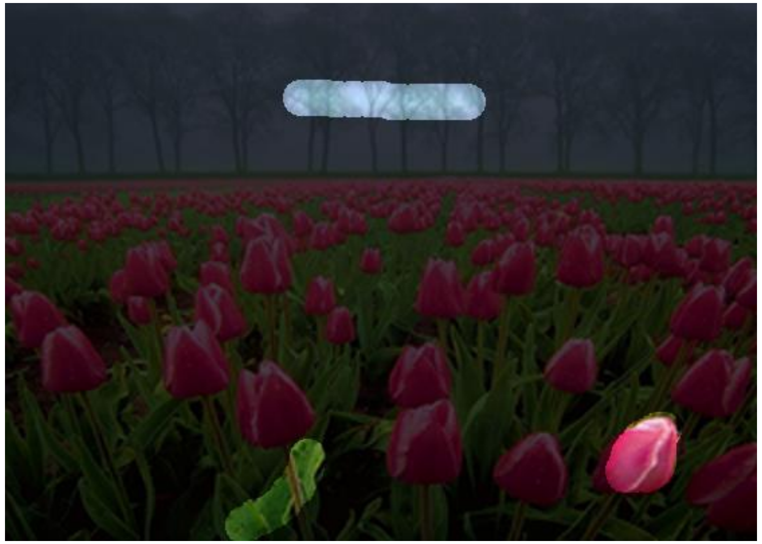
(83 frames)

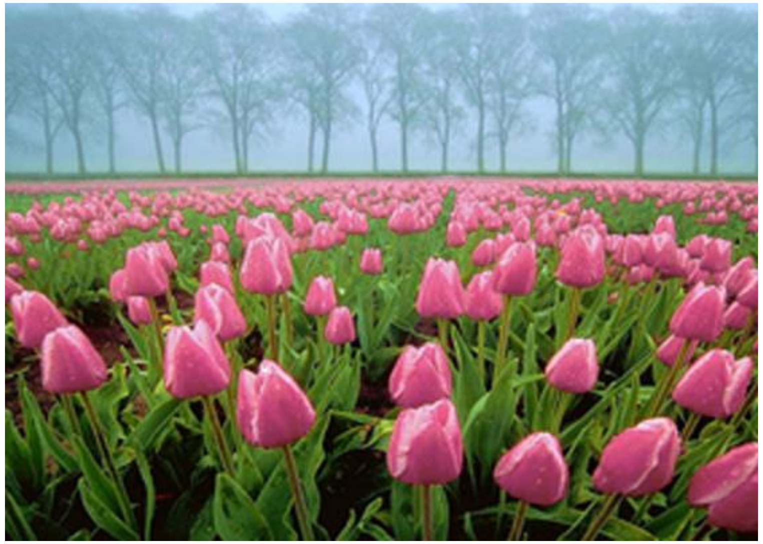


编辑传播

- **定义:** 将图像/视频局部编辑的结果传播到其余部分，从而只需要用户输入较小的信息集，便可实现图像/视频编辑

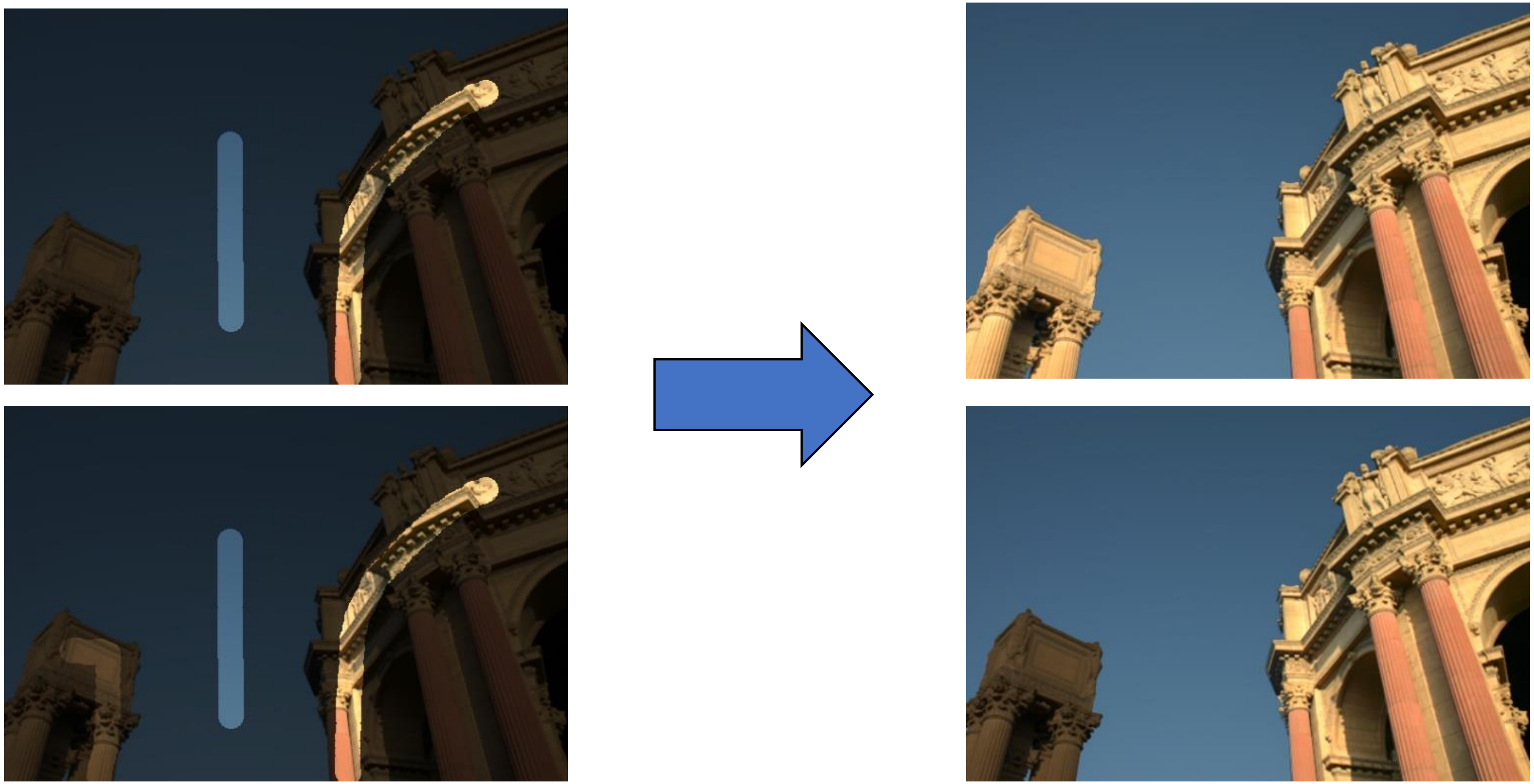






编辑传播

- **方法:** 基于区域相似性的编辑传播



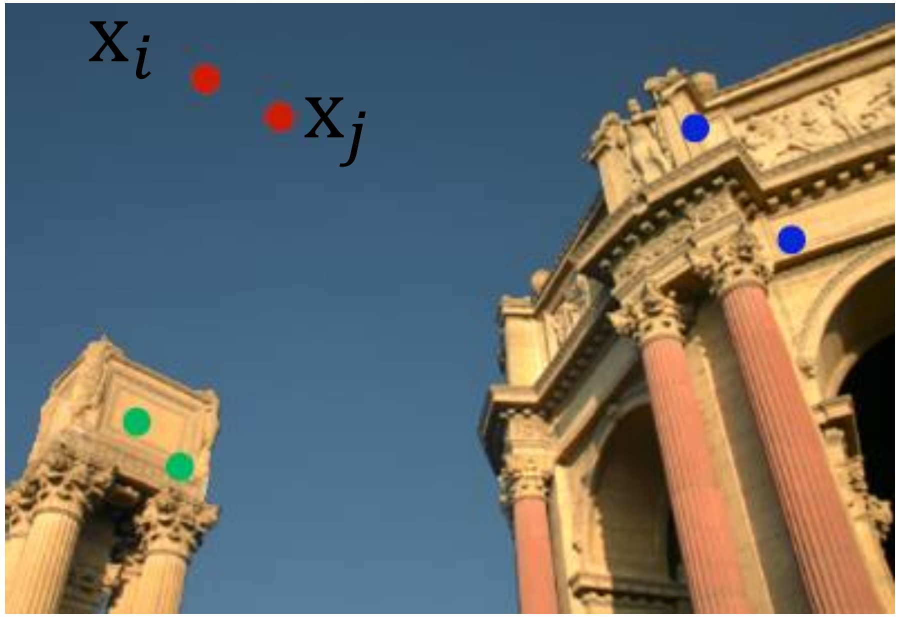
编辑传播

- **方法:** 基于区域相似性的编辑传播
  - 优化图像空间的相似性函数

$$z_{ij}=\exp\left(-\|\mathrm{f}_i-\mathrm{f}_j\|^2/\sigma_a\right)\exp\left(-\|\mathrm{x}_i-\mathrm{x}_j\|^2/\sigma_s\right)$$

像素表观:
局部颜色均
值/方差等

空间位置



编辑传播

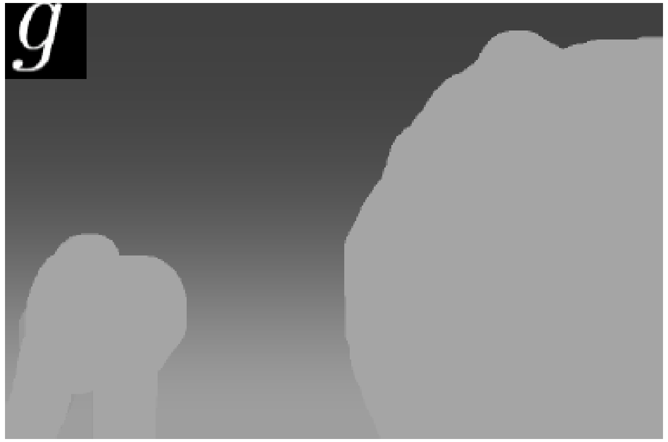
- **方法:** 基于区域相似性的编辑传播
    - 优化图像空间的相似性函数

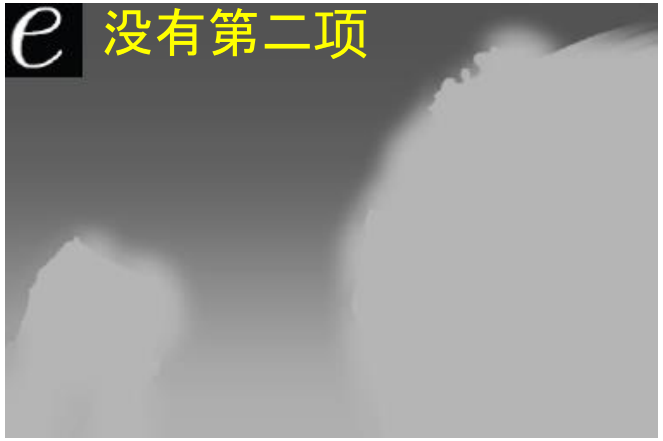
$$\sum_i \sum_j w_j z_{ij}(e_i - g_j)^2 + \lambda \sum_i \sum_j z_{ij}(e_i - e_j)^2$$

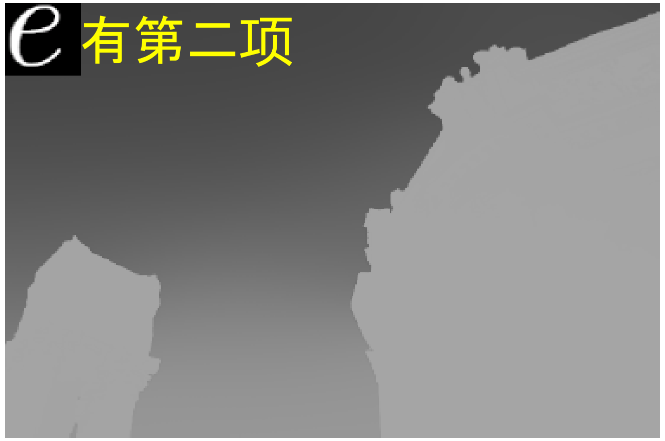
满足用户编辑g的约束

编辑区域有相似的表现

– w_j: 指定满足约束条件像素的权重

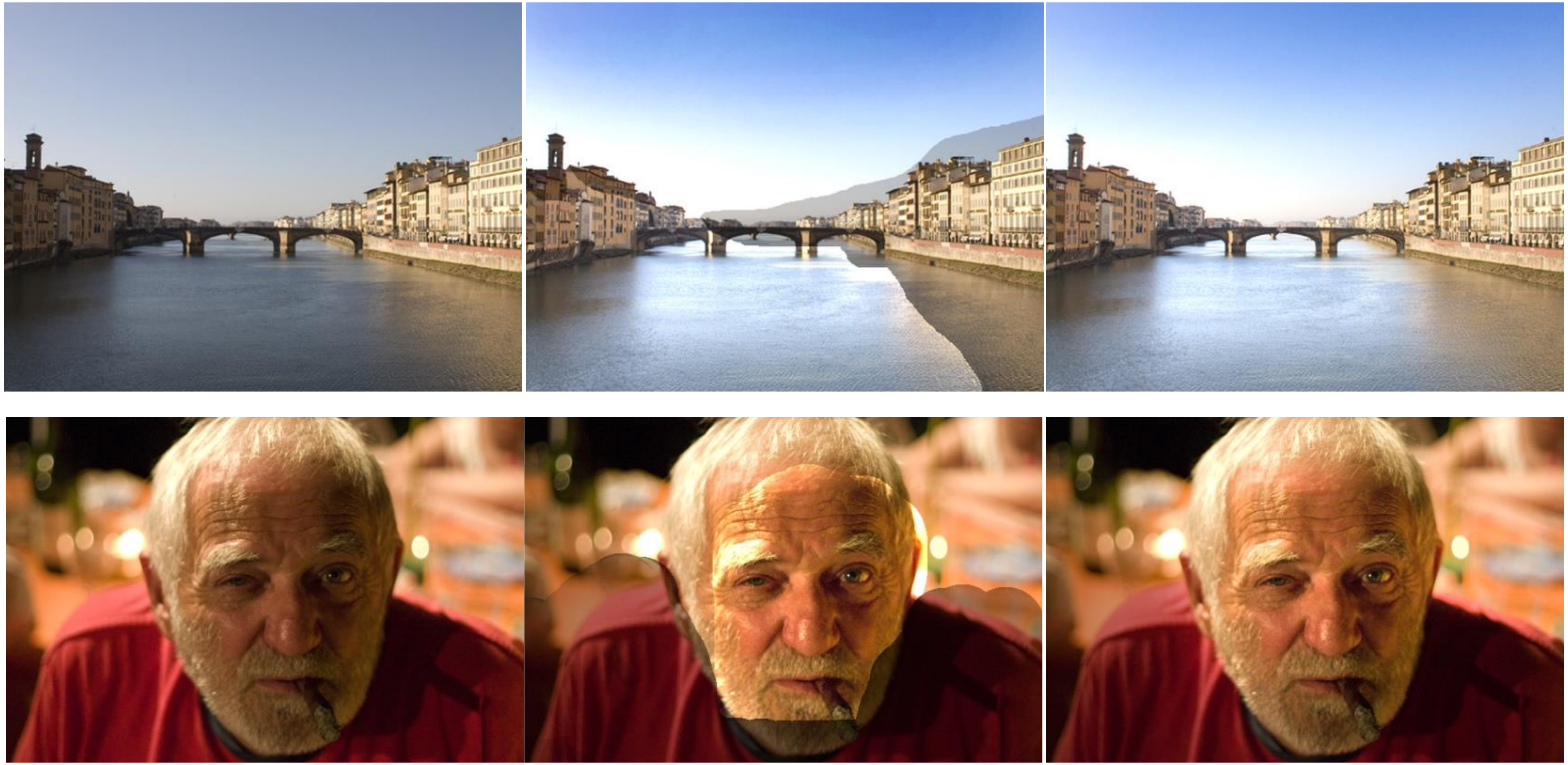






编辑传播

- **方法**: 基于区域相似性的编辑传播



Thank you!

Questions?