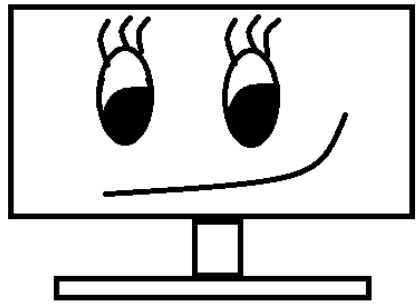
Seneca



CVI620/ DPS920
Introduction to Computer Vision

Image Morphology &
Geometric Transformations

Seneca College

Vida Movahedi

Overview

- Morphology
- Binary Masking
- Geometric Transformation

Morphology

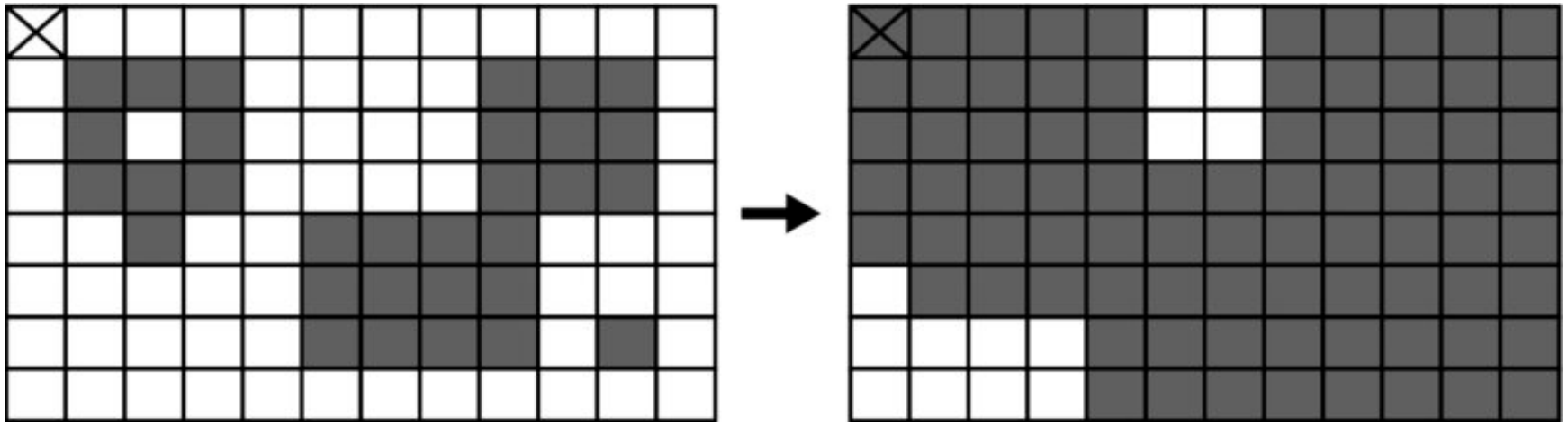
The study of structure or form [Merriam-Webster]

Morphology [2]

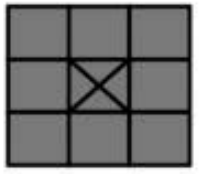
- Nonlinear filtering, often applied to binary images
- Kernels: often a square kernel or a disc, but can be more complicated
- Basic operations:
  - Dilate: Convolution of some image with a kernel in which any given pixel is replaced with the *local maximum* of all pixel values covered by the kernel
    - Effect: causes filled regions to grow
    - Good for finding connected components
  - Erode: Convolution of some image with a kernel in which any given pixel is replaced with the *local minimum* of all pixel values covered by the kernel
    - Good for removing smaller areas (noise)

Example: Dilate a binary image [3]

- Assuming white is zero in the following image

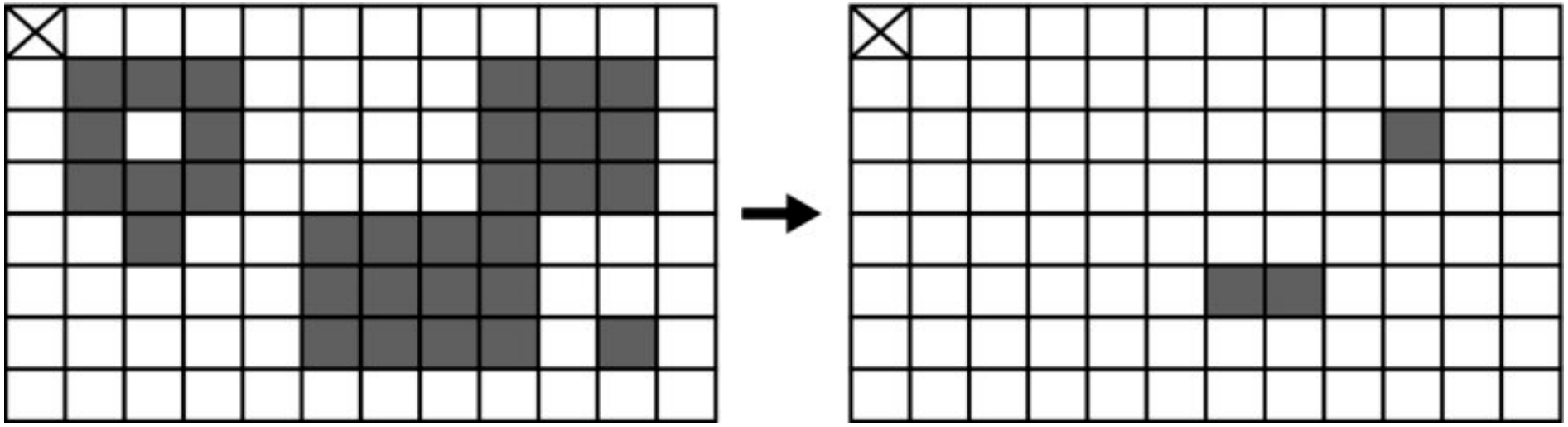


3x3 square kernel:

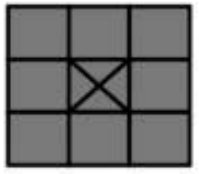


Example: Erode a binary image [3]

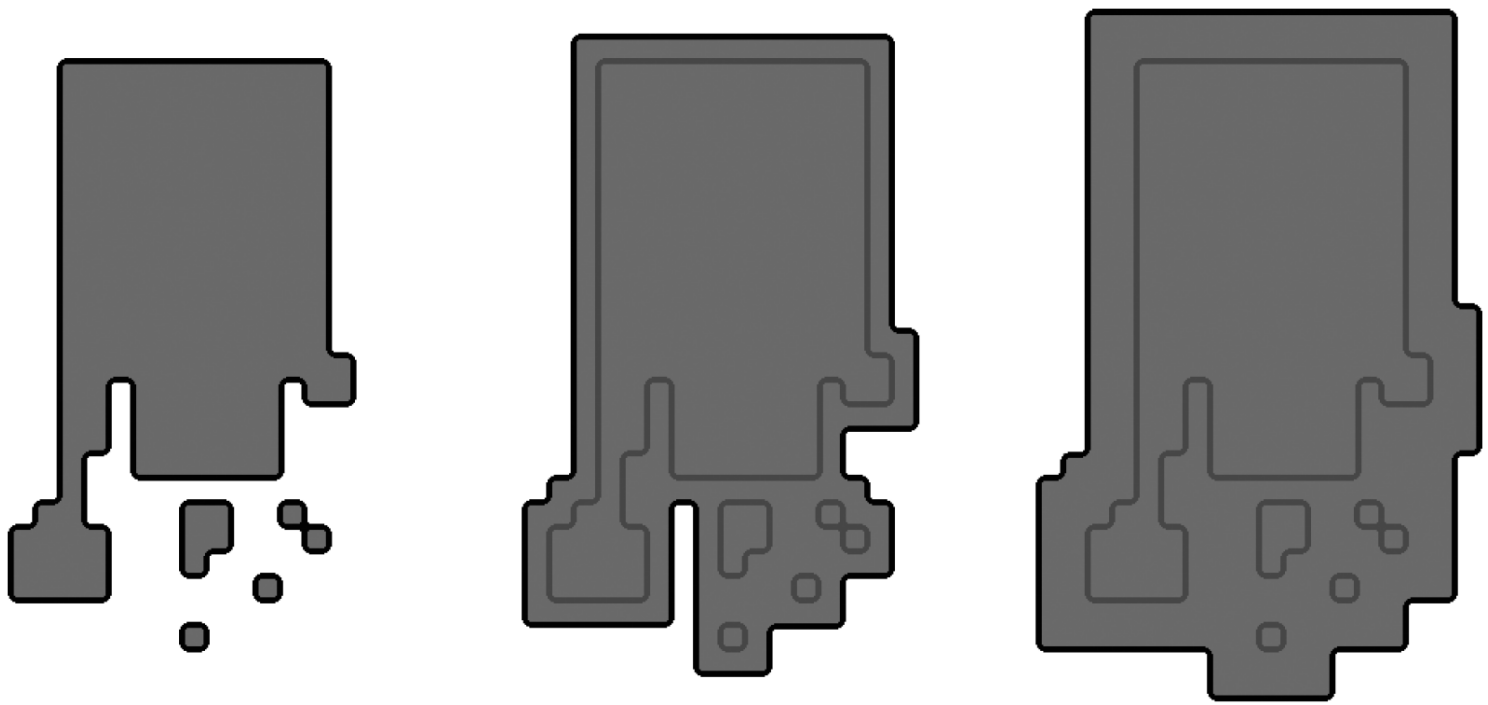
- Assuming white is zero in the following image



3x3 square kernel:

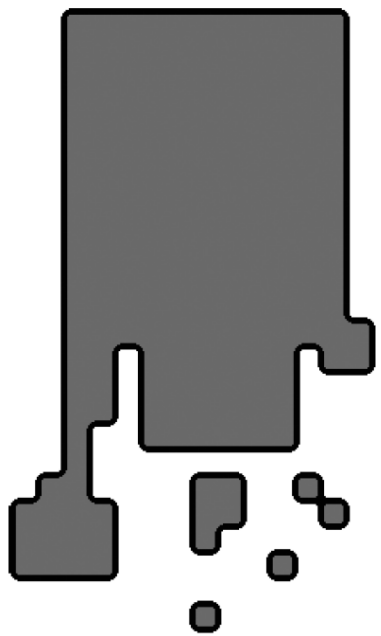


```
void cv::dilate(
  cv::InputArray    src,                    // Input image
  cv::OutputArray   dst,                    // Result image
  cv::InputArray    element,                                  // Structuring, a cv::Mat()
  cv::Point         anchor      = cv::Point(-1,-1),   // Location of anchor point
  int               iterations  = 1,                  // Number of times to apply
  int               borderType  = cv::BORDER_CONSTANT // Border extrapolation
  const cv::Scalar& borderValue = cv::morphologyDefaultBorderValue()
);
```

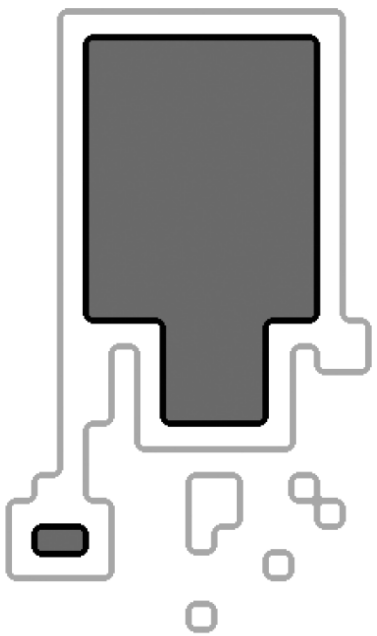


A) Original B) Dilate C) Twice

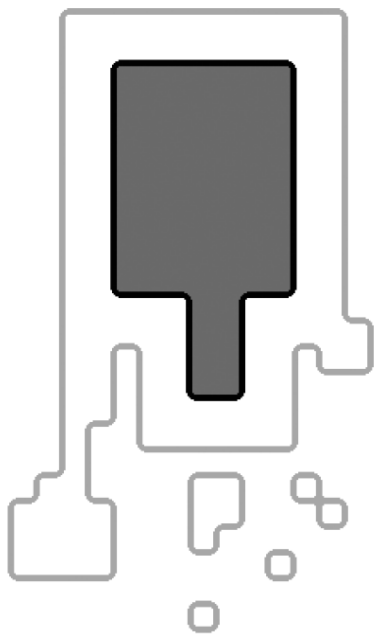
```
void cv::erode(
  cv::InputArray    src,                                  // Input image
  cv::OutputArray   dst,                                  // Result image
  cv::InputArray    element,                              // Structuring, a cv::Mat()
  cv::Point         anchor      = cv::Point(-1,-1),       // Location of anchor point
  int               iterations  = 1,                      // Number of times to apply
  int               borderType  = cv::BORDER_CONSTANT     // Border extrapolation
  const cv::Scalar& borderValue = cv::morphologyDefaultBorderValue()
);
```



A) Original



B) Erode



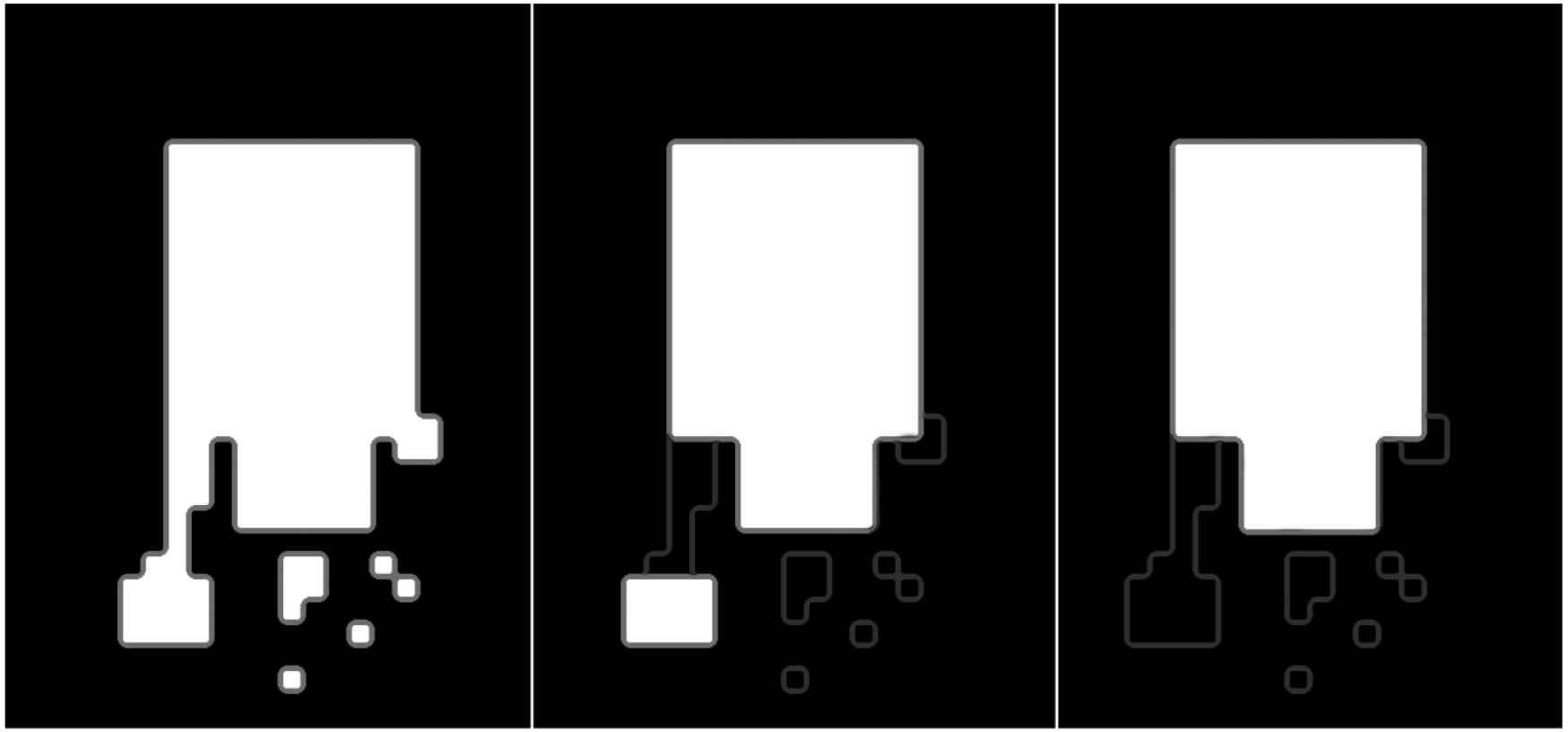
C) Twice

Opening

Example:

```
cv::Mat elem = 255 * cv::Mat::ones(s, s, CV_8UC1);
cv::morphologyEx(binI, morph, MORPH_OPEN, elem);
```

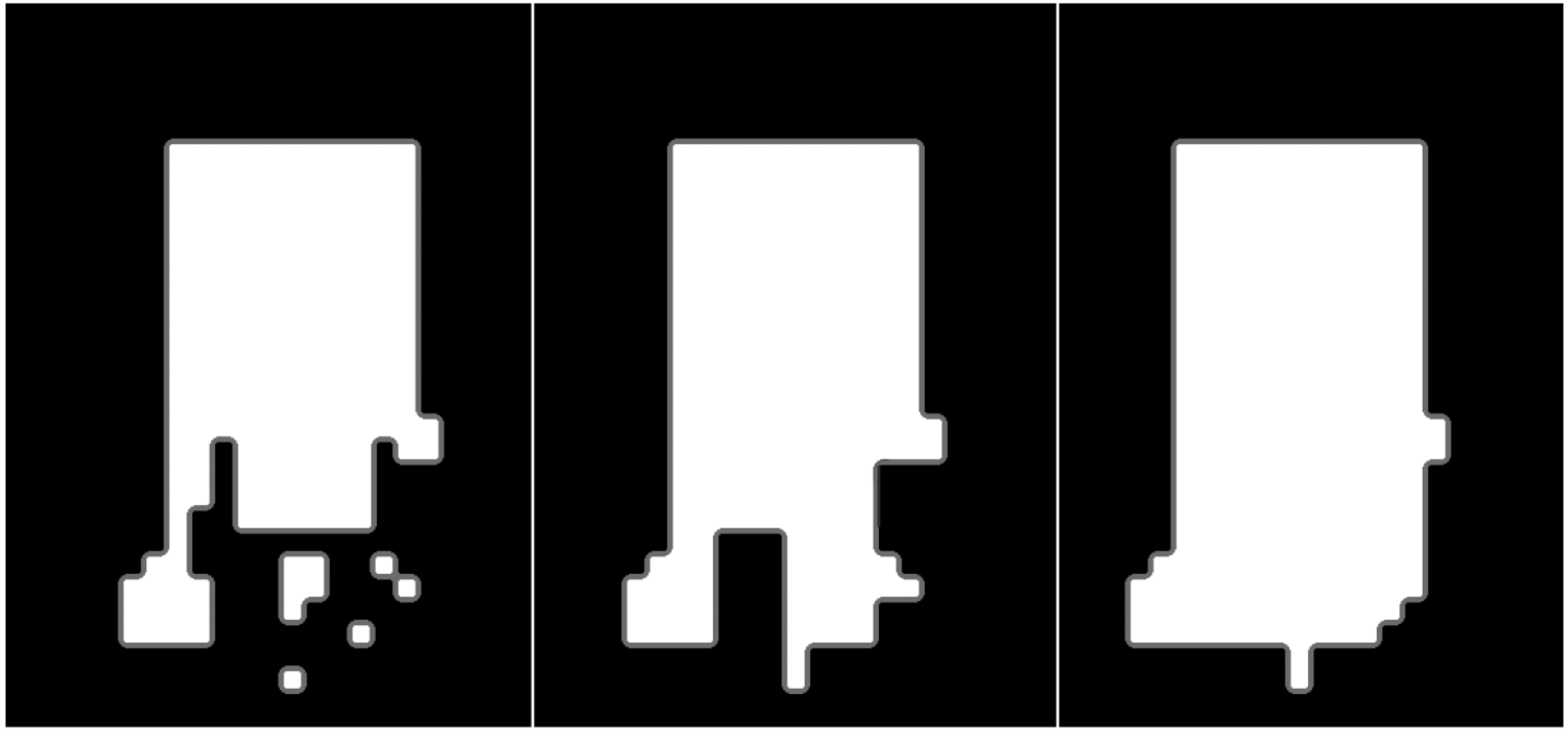
- Erode first, then dilate
- For example, for counting objects in a binary image



A) Original B) Open C) Twice

Closing

- Dilate first, then erode



A) Original B) Close C) Twice

```
void cv::morphologyEx(
  cv::InputArray    src,                          // Input image
  cv::OutputArray   dst,                          // Result image
  int               op,                           // Operator (e.g. cv::MOP_OPEN)
  cv::InputArray    element,                      // Structuring element, cv::Mat()
  cv::Point         anchor      = cv::Point(-1,-1),  // Location of anchor point
  int               iterations  = 1,                 // Number of times to apply
  int               borderType  = cv::BORDER_DEFAULT // Border extrapolation
  const cv::Scalar& borderValue = cv::morphologyDefaultBorderValue()

);
```

Value of operation	Morphological operator	Requires temp image?
cv::MOP_OPEN	Opening	No
cv::MOP_CLOSE	Closing	No
cv::MOP_GRADIENT	Morphological gradient	Always
cv::MOP_TOPHAT	Top Hat	For in-place only (src = dst)
cv::MOP_BLACKHAT	Black Hat	For in-place only (src = dst)

type of morphological operation

Enumerator	
MORPH_ERODE	see **cv::erode**
MORPH_DILATE	see **cv::dilate**
MORPH_OPEN	an opening operation $$\texttt{dst} = \mathrm{open}(\texttt{src}, \texttt{element}) = \mathrm{dilate}(\mathrm{erode}(\texttt{src}, \texttt{element}))$$
MORPH_CLOSE	a closing operation $$\texttt{dst} = \mathrm{close}(\texttt{src}, \texttt{element}) = \mathrm{erode}(\mathrm{dilate}(\texttt{src}, \texttt{element}))$$
MORPH_GRADIENT	a morphological gradient $$\texttt{dst} = \mathrm{morph\_grad}(\texttt{src}, \texttt{element}) = \mathrm{dilate}(\texttt{src}, \texttt{element}) - \mathrm{erode}(\texttt{src}, \texttt{element})$$
MORPH_TOPHAT	"top hat" $$\texttt{dst} = \mathrm{tophat}(\texttt{src}, \texttt{element}) = \texttt{src} - \mathrm{open}(\texttt{src}, \texttt{element})$$
MORPH_BLACKHAT	"black hat" $$\texttt{dst} = \mathrm{blackhat}(\texttt{src}, \texttt{element}) = \mathrm{close}(\texttt{src}, \texttt{element}) - \texttt{src}$$

Binary Masking

Binary Masking

- Binary Mask:
  - A matrix of 0's and 1's, with the same size as the image
  - Only compute the function/ apply the changes for nonzero elements

Mask

Masking in OpenCV

- Examples in OpenCV:

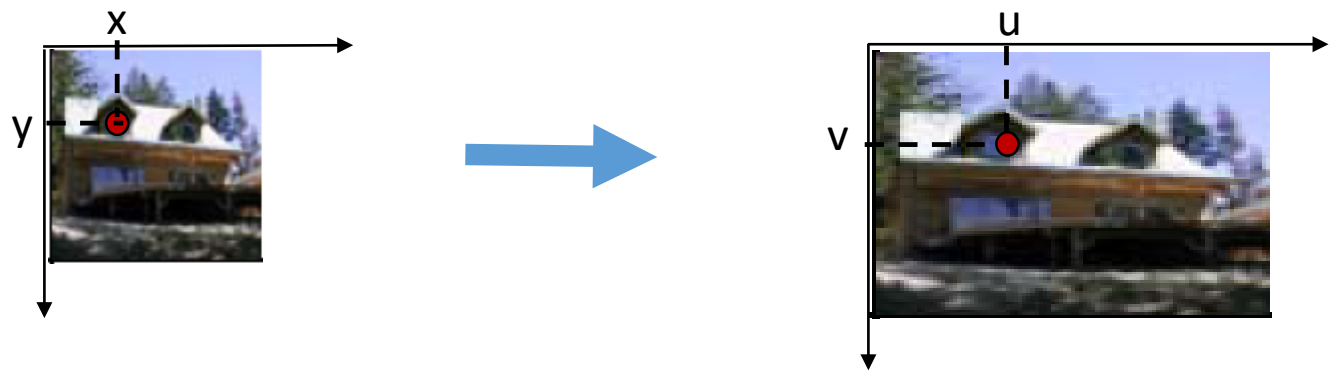
`m0.copyTo( m1, mask );`	Same as `m0.copyTo(m1)`, except only entries indicated in the array mask are copied
`m0.setTo( s, mask );`	Set all entries in `m0` to singleton value `s`; if mask is present, set only those values corresponding to nonzero elements in `mask`

```
void cv::add(
  cv::InputArray  src1,                  // First input array
  cv::InputArray  src2,                  // Second input array
  cv::OutputArray dst,                   // Result array
  cv::InputArray  mask  = cv::noArray(), // Optional, do only where nonzero
  int             dtype = -1             // Output type for result array
);
```

Geometric Transformation

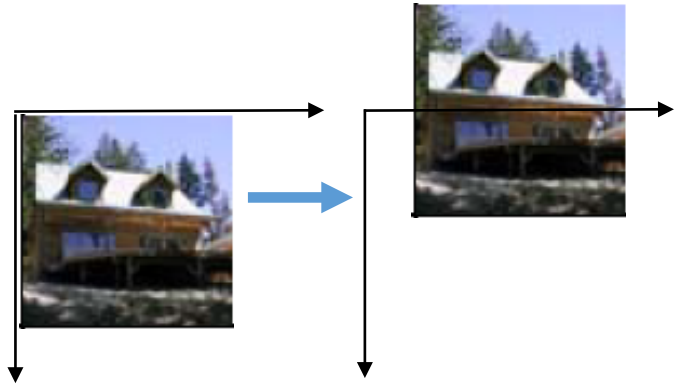
2D Transformations

- A pixel in the source image at location (x,y) is mapped to location (u,v) in the destination image

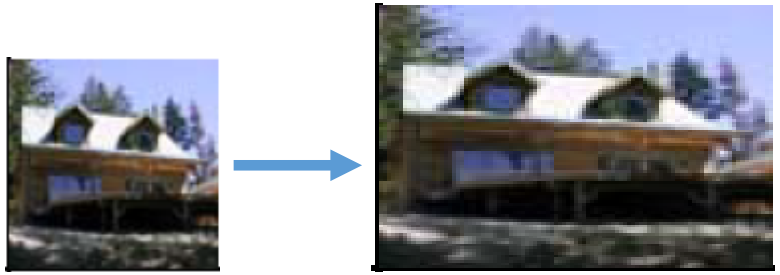


Types of 2D Transformation

- Translation – pixels move in the same direction
  - $u = x + t_x$
  - $v = y + t_y$



- Scale or resize
  - $u = x * s_x$
  - $v = y * s_y$

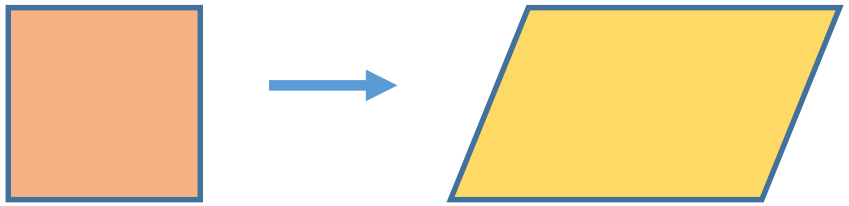


- Rotation
  - $u = x * \cos\theta - y * \sin\theta$
  - $v = y * \sin\theta + x * \cos\theta$

Types of 2D Transformation (cont.)

- Shear
  - $u = x + y * sh_x$
  - $u = y + x * sh_y$

  - If $sh_y = 0$



Matrix Notation for Affine Transformations

- ***Affine:***

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Translation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Scale/ Resize:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

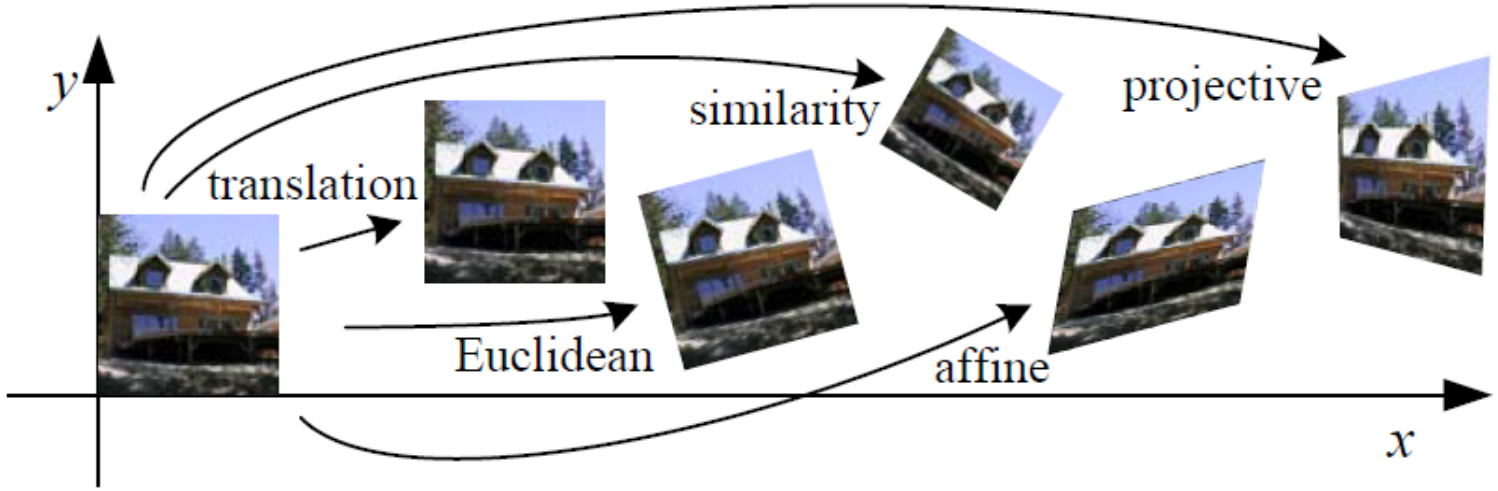
- Rotation:

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

- Shear:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Geometric Image Transformations



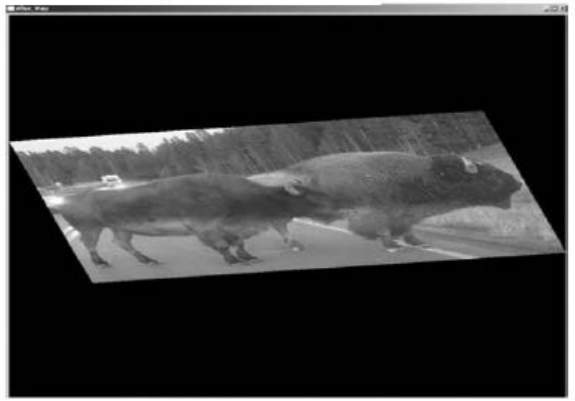
Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Original

Perspective

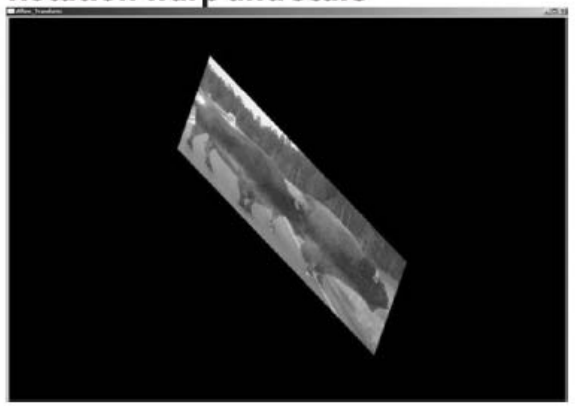
Rotation and scale

Affine warp



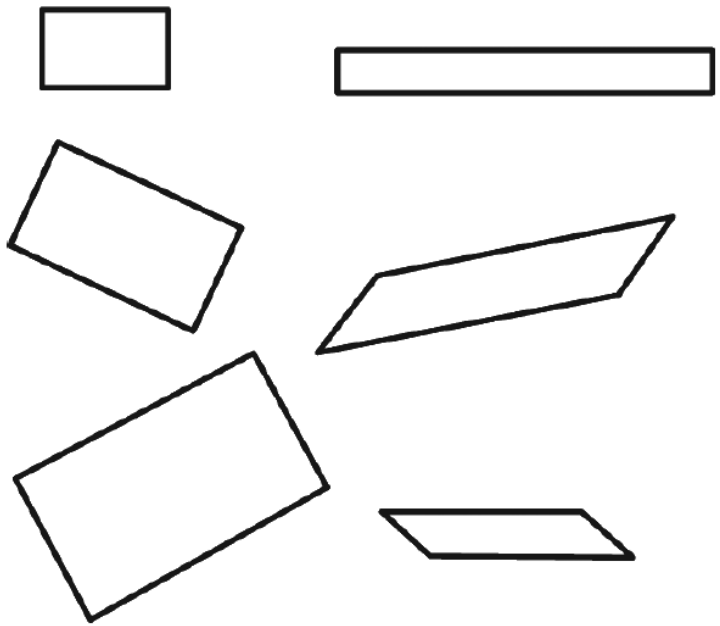
Affine scale

Rotation warp and scale



Affine (2x2)

Parallelograms

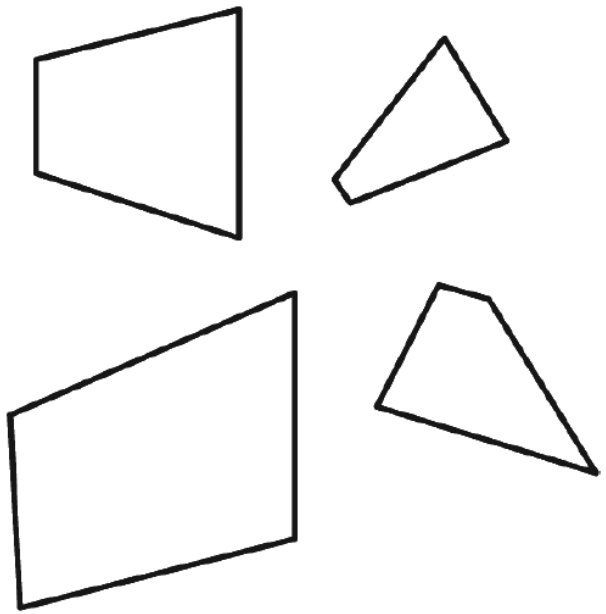


Perspective (3x3)

(or "Homography")

Trapezoids

(Includes all of Affine)



Implementations:

- Forward implementation:
  - For every pixel in the source image at location (x,y)
    - Calculate (u,v)
    - Copy the pixel: $I_{dst}(u,v) = I_{src}(x,y)$

- Backward implementation
  - For every pixel in the destination image at location (u,v)
    - Calculate (x,y) – inverse transformation
    - Estimate the value of pixel at (x,y)
    - Copy the pixel: $I_{dst}(u,v) = I_{src}(x,y)$

Affine Transform using OpenCV

- Given the 2x3 transform matrix M, find the result *dst*

```
void cv::warpAffine(
  cv::InputArray    src,                                  // Input image
  cv::OutputArray   dst,                                  // Result image
  cv::InputArray    M,                                    // 2-by-3 transform mtx
  cv::Size          dsize,                                // Destination image size
  int               flags       = cv::INTER_LINEAR,     // Interpolation, inverse
  int               borderMode  = cv::BORDER_CONSTANT,  // Pixel extrapolation
  const cv::Scalar& borderValue = cv::Scalar()          // For constant borders
);
```

Find the transform

- Given the resulting image, find the transformation matrix

```
cv::Mat cv::getAffineTransform(                    // Return 2-by-3 matrix
  const cv::Point2f* src,                          // Coordinates *three* of vertices
  const cv::Point2f* dst                           // Target coords, three vertices
);
```

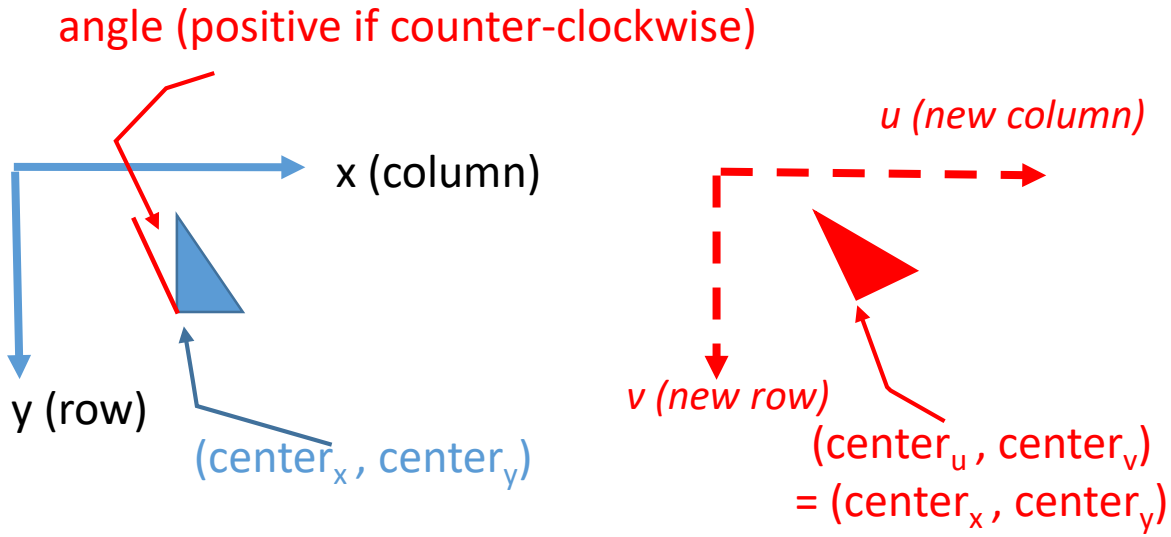
Get the similarity transform matrix

```
cv::Mat cv::getRotationMatrix2D(                    // Return 2-by-3 matrix
  cv::Point2f  center                               // Center of rotation
  double       angle,                               // Angle of rotation
  double       scale                                // Rescale after rotation
);
```

If we define α = scale * cos(angle) and β = scale * sin(angle), then this function computes the matrix M to be:

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center_x - \beta \cdot center_y \\ -\beta & \alpha & \beta \cdot center_x + (1-\alpha)\cdot center_y \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Rotate an image

```
angle = 30;
M = getRotationMatrix2D(cv::Point2f(img.cols / 2.0, img.rows /2.0), angle, 1);
warpAffine(img, dst, M, img.size());
```

Resize an image

- Also the resize function designed specifically for scaling the image

```
void cv::resize(
  cv::InputArray  src,                                  // Input image
  cv::OutputArray dst,                                  // Result image
  cv::Size        dsize,                                // New size
  double          fx            = 0,                    // x-rescale
  double          fy            = 0,                    // y-rescale
  int             interpolation = CV::INTER_LINEAR // interpolation method
);
```

Table 11-1. cv::resize() interpolation options

Interpolation	Meaning
cv::INTER_NEAREST	Nearest neighbor
cv::INTER_LINEAR	Bilinear
cv::INTER_AREA	Pixel area resampling
cv::INTER_CUBIC	Bicubic interpolation
cv::INTER_LANCZOS4	Lanczos interpolation over 8 × 8 neighborhood.

It is important to notice the difference between `cv::resize()` and the similarly named `cv::Mat::resize()` member function of the `cv::Mat` class. `cv::resize()` creates a new image of a different size, over which the original pixels are mapped. The `cv::Mat::resize()` member function resizes the image whose member you are calling, and crops that image to the new size. Pixels are not interpolated (or extrapolated) in the case of `cv::Mat::resize()`.

Resize an image

```
fx = 1;
fy = 2;
resize(img, dst, Size(), fx, fy);
```

Find the inverse transform

- Given the transform matrix, find the inverse

```
void cv::invertAffineTransform(
  cv::InputArray  M,                    // Input 2-by-3 matrix
  cv::OutputArray iM                    // Output also a 2-by-3 matrix
);
```

Perspective Transform

- Given the 3x3 transform matrix M, find the result *dst*

```
void cv::warpPerspective(
  cv::InputArray    src,                                  // Input image
  cv::OutputArray   dst,                                  // Result image
  cv::InputArray    M,                                    // 3-by-3 transform mtx
  cv::Size          dsize,                                // Destination image size
  int               flags       = cv::INTER_LINEAR,     // Interpolation, inverse
  int               borderMode  = cv::BORDER_CONSTANT,  // Extrapolation method
  const cv::Scalar& borderValue = cv::Scalar()          // For constant borders
);
```

Find the perspective transform

- Given the resulting image, find the transformation matrix

```
cv::Mat cv::getPerspectiveTransform(            // Return 3-by-3 matrix
  const cv::Point2f* src,                       // Coordinates of *four* vertices
  const cv::Point2f* dst                        // Target coords, four vertices
);
```

References

[1] Computer Vision: Algorithms and Applications, R. Szeliski (http://szeliski.org/Book)

[2] Learning OpenCV 3, A. Kaehler & G. Bradski

- Available online through Safari Books, Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA51532449200003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US

[3] Practical introduction to Computer Vision with OpenCV, Kenneth Dawson-Howe

- Available through Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA51428109500003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US