Assignment-1

Geometric Transformation and Interpolation

Review: Geometric Transformation

	0	1
0	20	120
1	90	10

	0	1	2	3
0				
1				
2				
3				

Review: Geometric Transformation

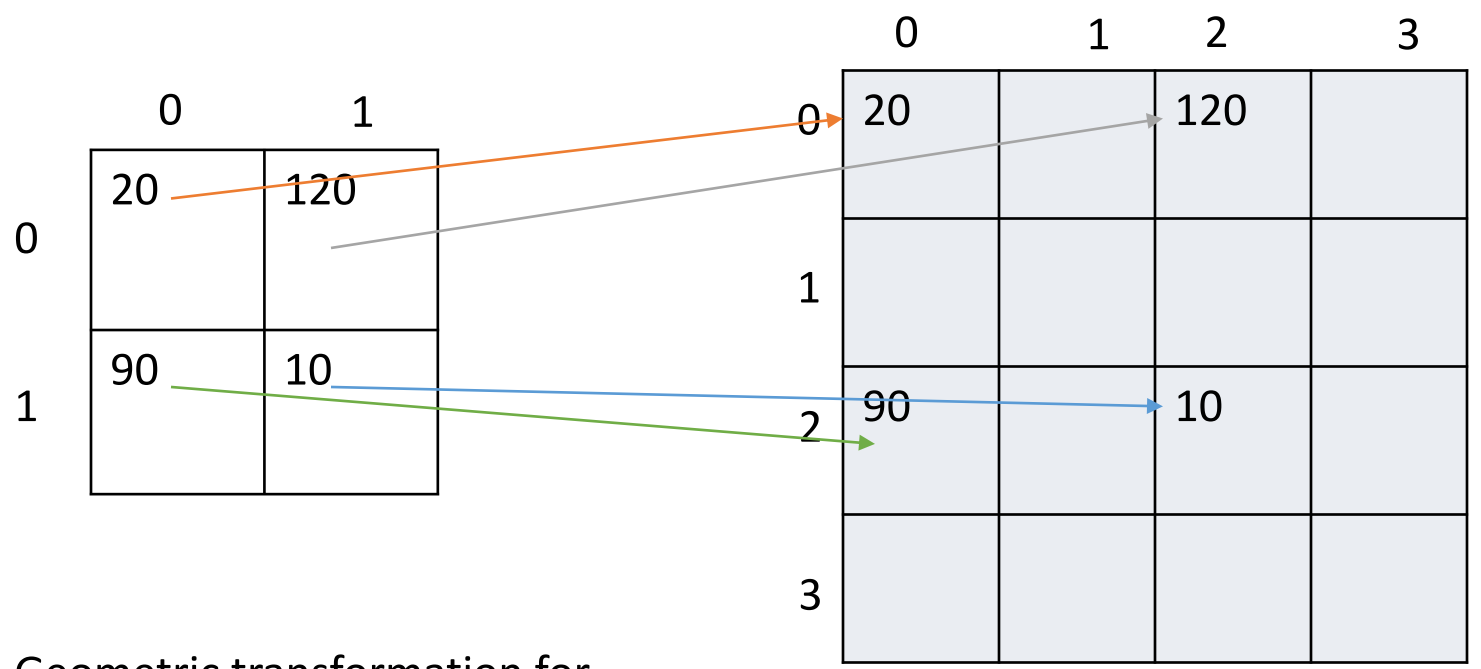
	0	1
0	20	120
1	90	10

	0	1	2	3
0				
1				
2				
3				

Geometric transformation for mapping pixels.

(0,0) X 2 → (0,0)
(0,1) X 2 → (0,2)
(1,0) X 2 → (2,0)
(1,1) X 2 → (2,2)

Review: Geometric Transformation



Geometric transformation for mapping pixels.
(0,0) X 2 → (0,0)
(0,1) X 2 → (0,2)
(1,0) X 2 → (2,0)
(1,1) X 2 → (2,2)

Forward Transformation Function
x’ = 2x
y’ = 2y

Review: Image Interpolation

- **Interpolation** — Process of using known data to estimate unknown values

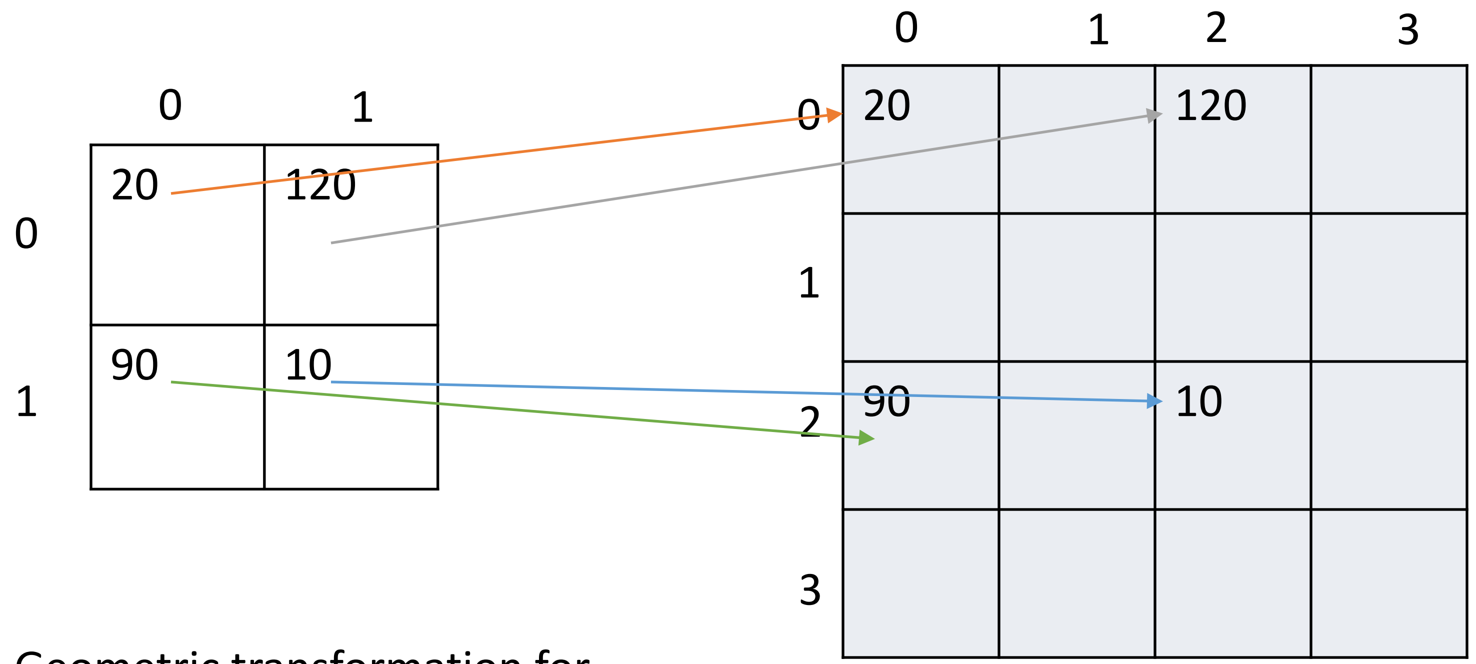
  e.g., zooming, shrinking, rotating, and geometric correction

- **Interpolation** (sometimes called ***resampling***) — an imaging method to increase (or decrease) the number of pixels in a digital image.

  Some digital cameras use interpolation to produce a larger image than the sensor captured or to create digital zoom

http://www.dpreview.com/learn/?/key=interpolation

Review: Geometric Transformation



Geometric transformation for mapping pixels.

(0,0) X 2 → (0,0)

(0,1) X 2 → (0,2)

(1,0) X 2 → (2,0)

(1,1) X 2 → (2,2)

It is difficult to interpolate and fill missing value when applying forward geometric transformation.

Review: Geometric Transformation: Inverse lookup

	0	1
0	20	120
1	90	10

	0	1	2	3
0				
1				
2				
3				

1. Create an image of desired size

Review: Geometric Transformation: Inverse lookup

	0	1
0	20	120
1	90	10

	0	1	2	3
0				
1				
2				
3				

1. Create an image of desired size
2. For each pixel in the new image calculate which pixel it corresponds to in the original image (Inverse transformation)

Geometric Transformation: Inverse lookup

	0	1
0	20	120
1	90	10

	0	1	2	3
0				
1				
2				
3				

Inverse mapping pixels.
(0,0) X 1/2 → (0,0)
(0,1) X 1/2 → (0,0.5)
(0,2) X 1/2 → (0,1)
(0,3) X 1/2 → (0,1.5)

Inverse Transformation Function
$x = 1/2\ x'$
$y = 1/2\ y'$

1. Create an image of desired size
2. For each pixel in the new image calculate which pixel it corresponds to in the original image (Inverse transformation).

Geometric Transformation: Inverse lookup

	0	1
0	20	120
1	90	10

	0	1	2	3
0				
1				
2				
3				

Inverse mapping pixels.
(0,0) X 1/2 → (0,0)
(0,1) X 1/2 → (0,0.5)
(0,2) X 1/2 → (0,1)
(0,3) X 1/2 → (0,1.5)

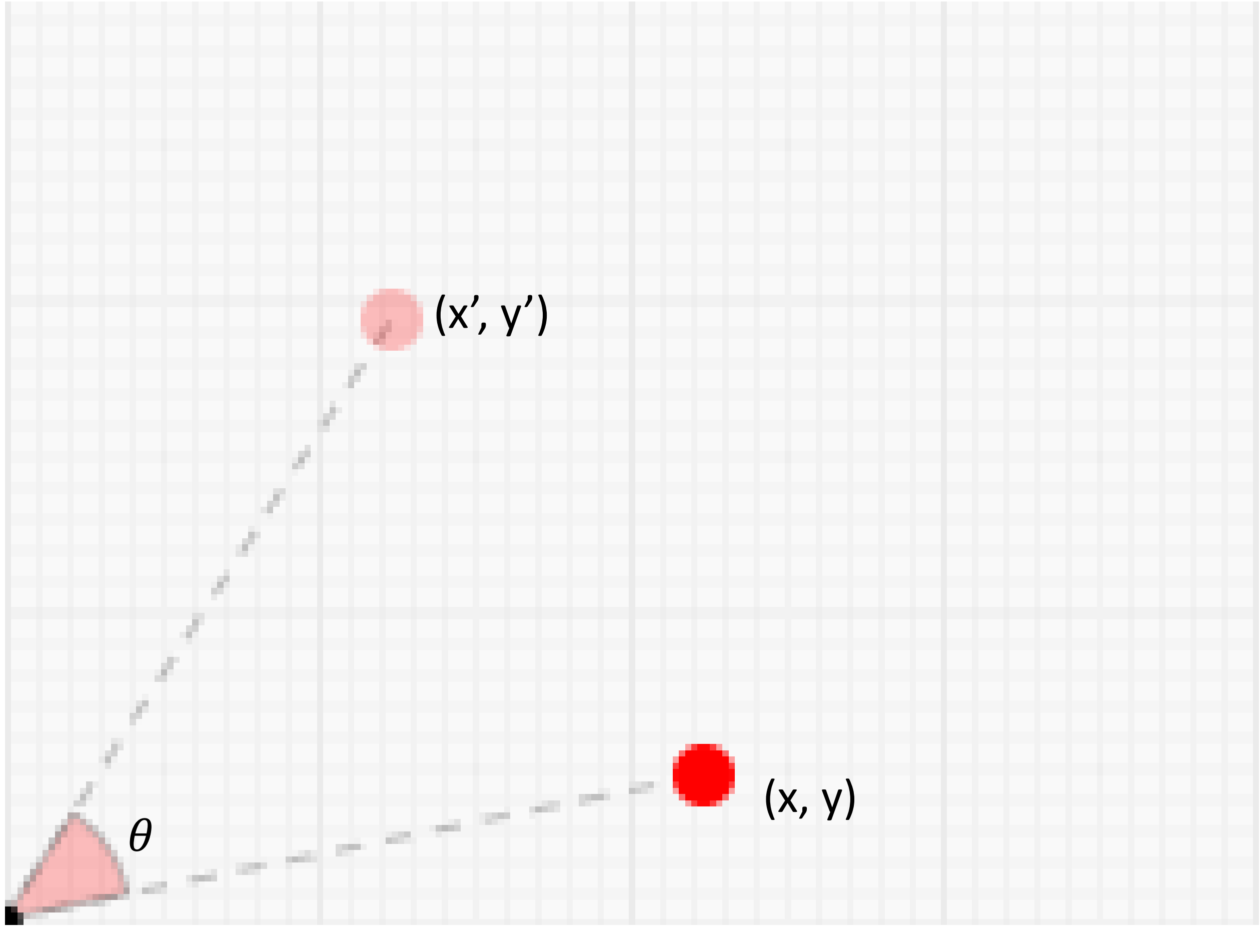
1. Create an image of desired size
2. For each pixel in the new image calculate which pixel it corresponds to in the original image.
3. Use values from nearby pixel to guess missing values

Rotate Image and Perform Interpolation

1. Forward rotate image
2. Inverse rotate image
3. Rotation with interpolation
    1. Nearest neighbour interpolation
    2. Bilinear interpolation

Rotation

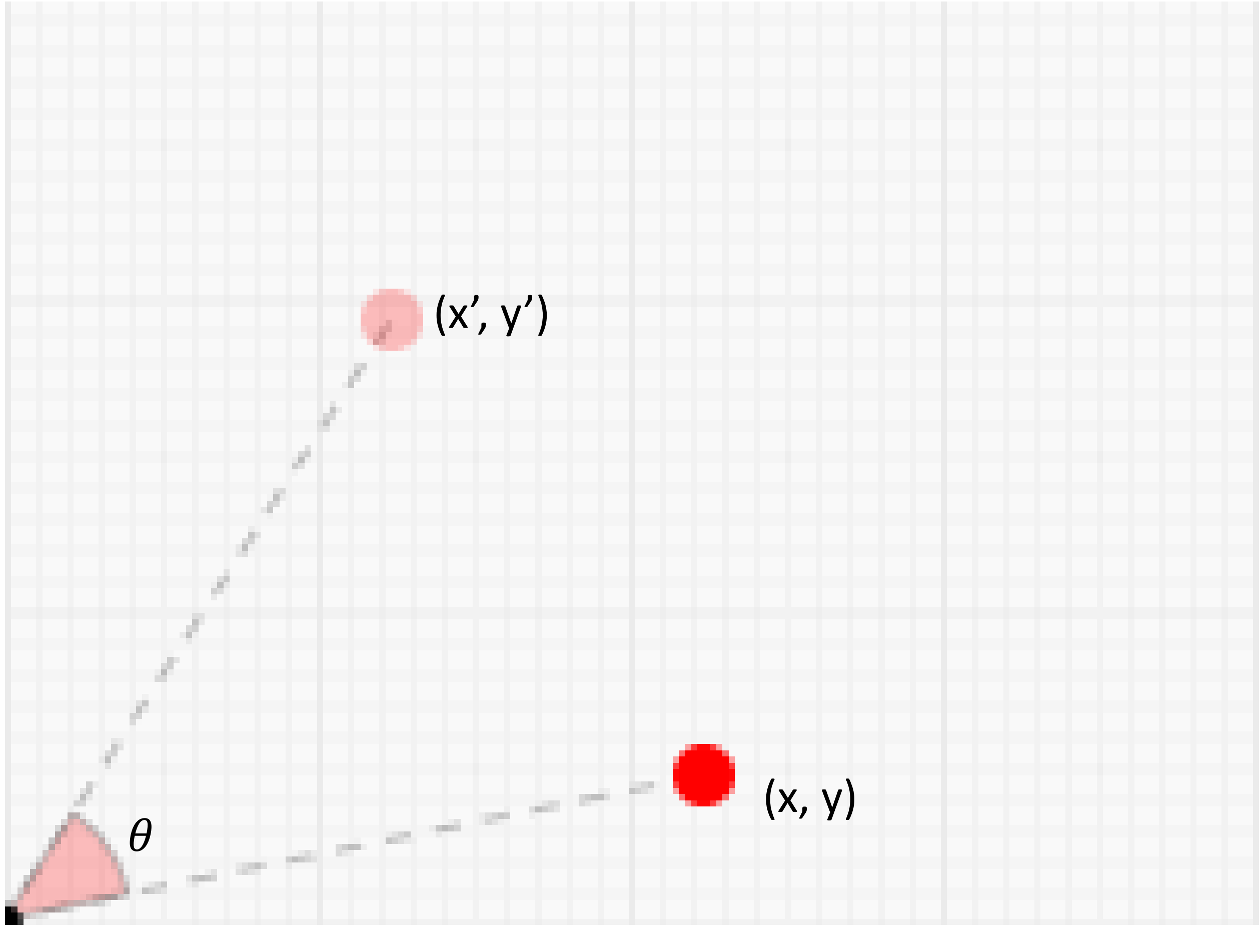
- Initial location = (x, y)
- After rotation
  - $(x', y') = ?$



Rotation

- Initial location = (x, y)
- After rotation
  - $(x', y') = ?$

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$



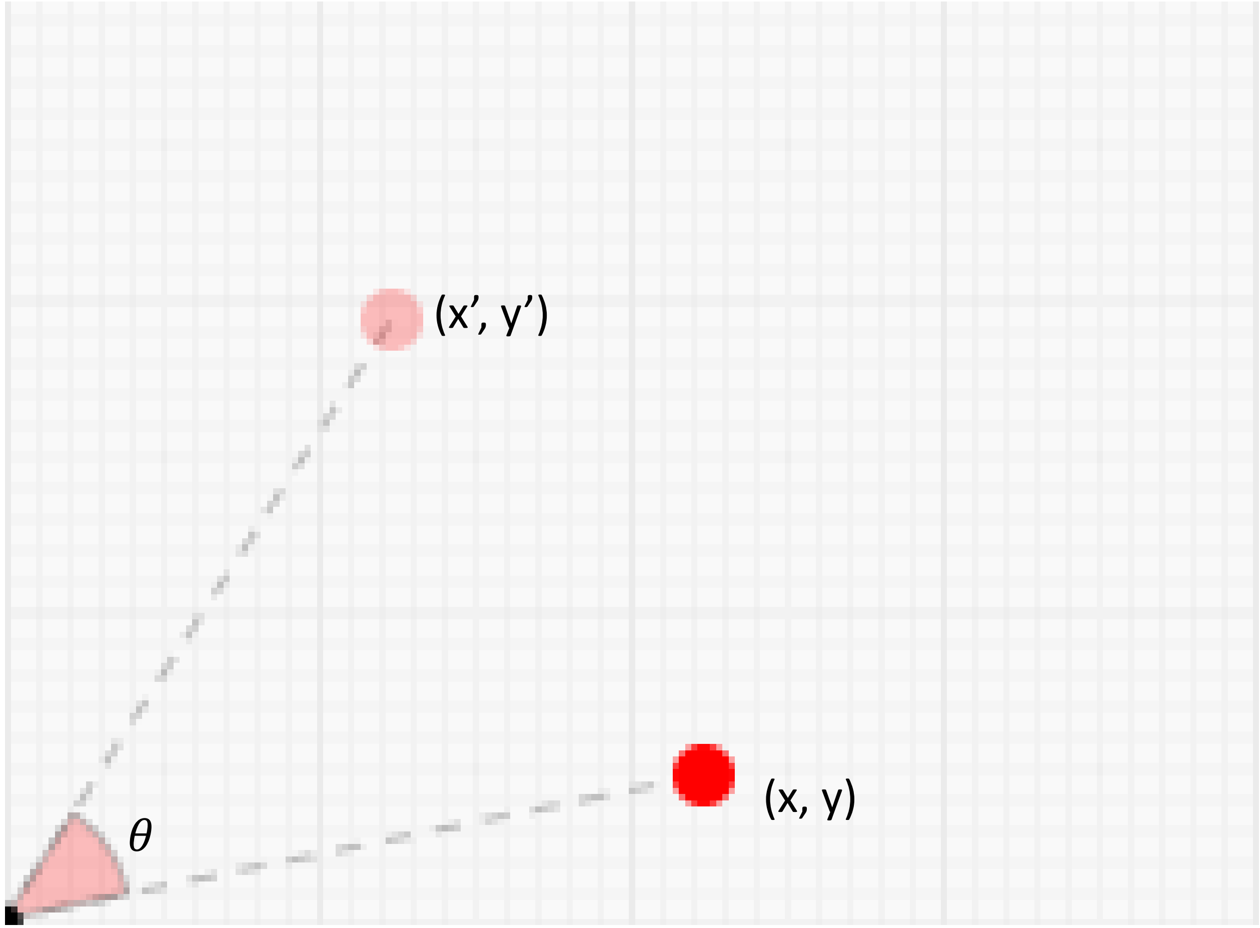
Rotation

- Initial location = (x, y)
- After rotation
  - $(x', y') = ?$

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

Matrix Notation

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



Rotation

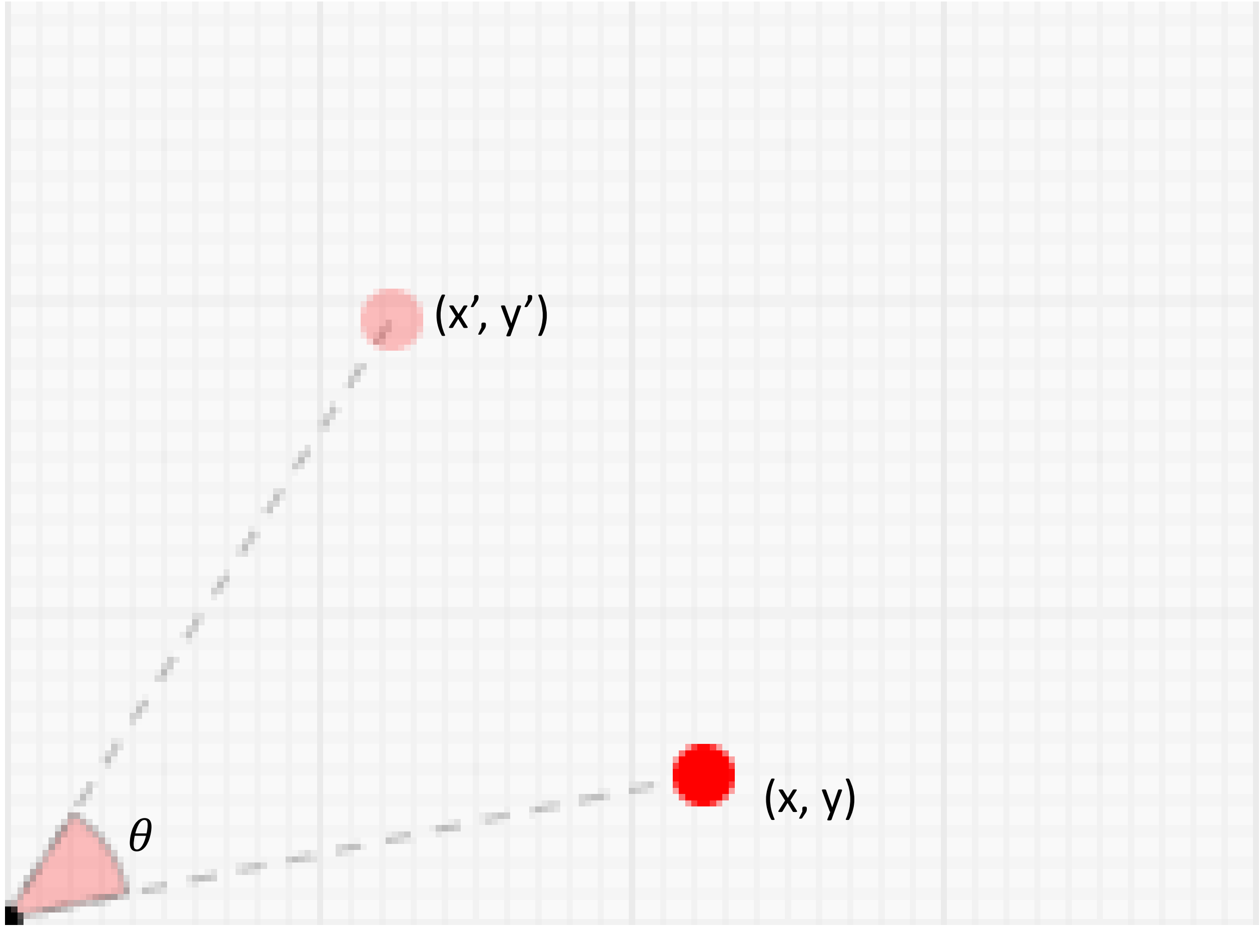
- Initial location = (x, y)
- After rotation
  - $(x', y') = ?$

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

Matrix Notation

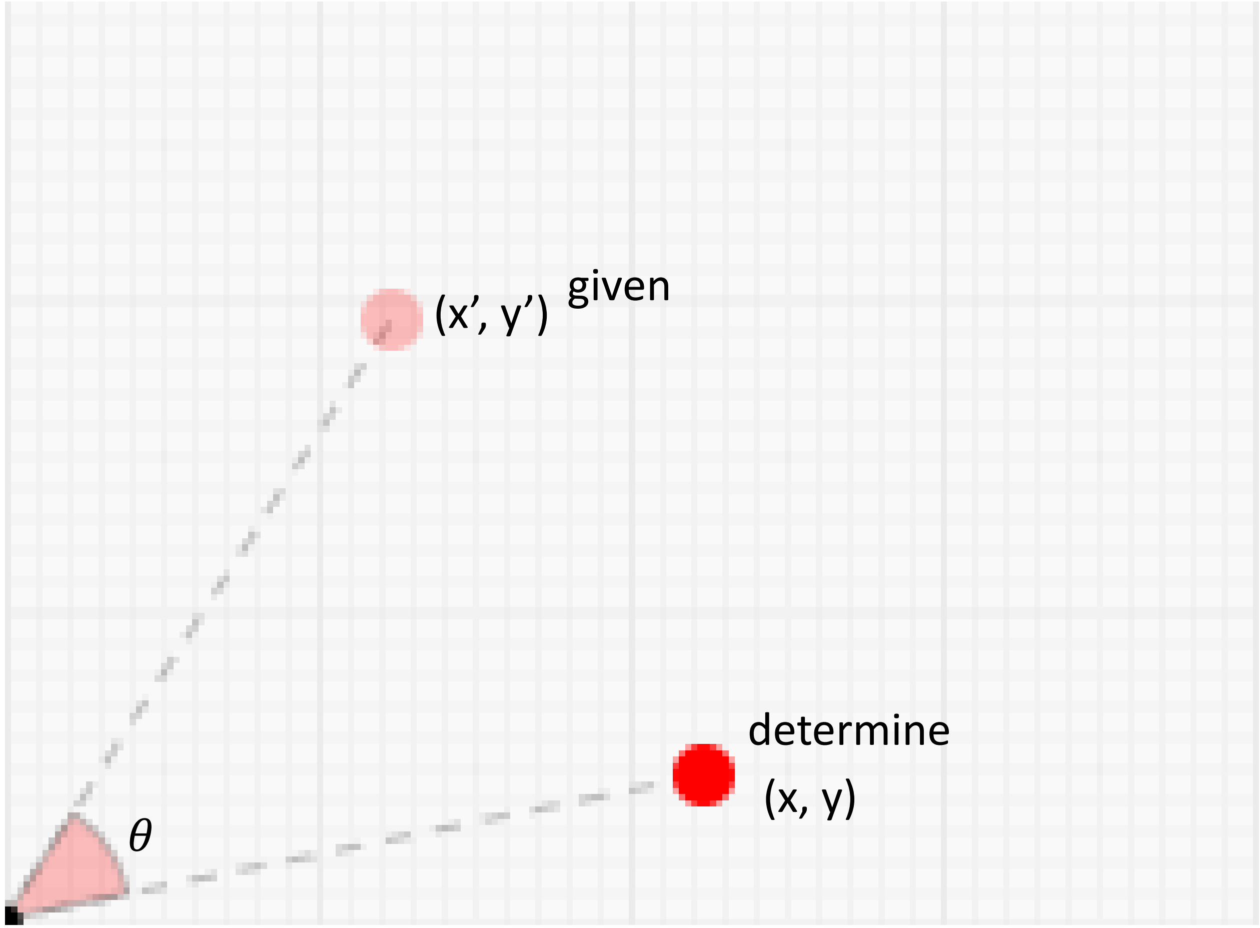
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Rotation Matrix



Inverse Rotation

- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

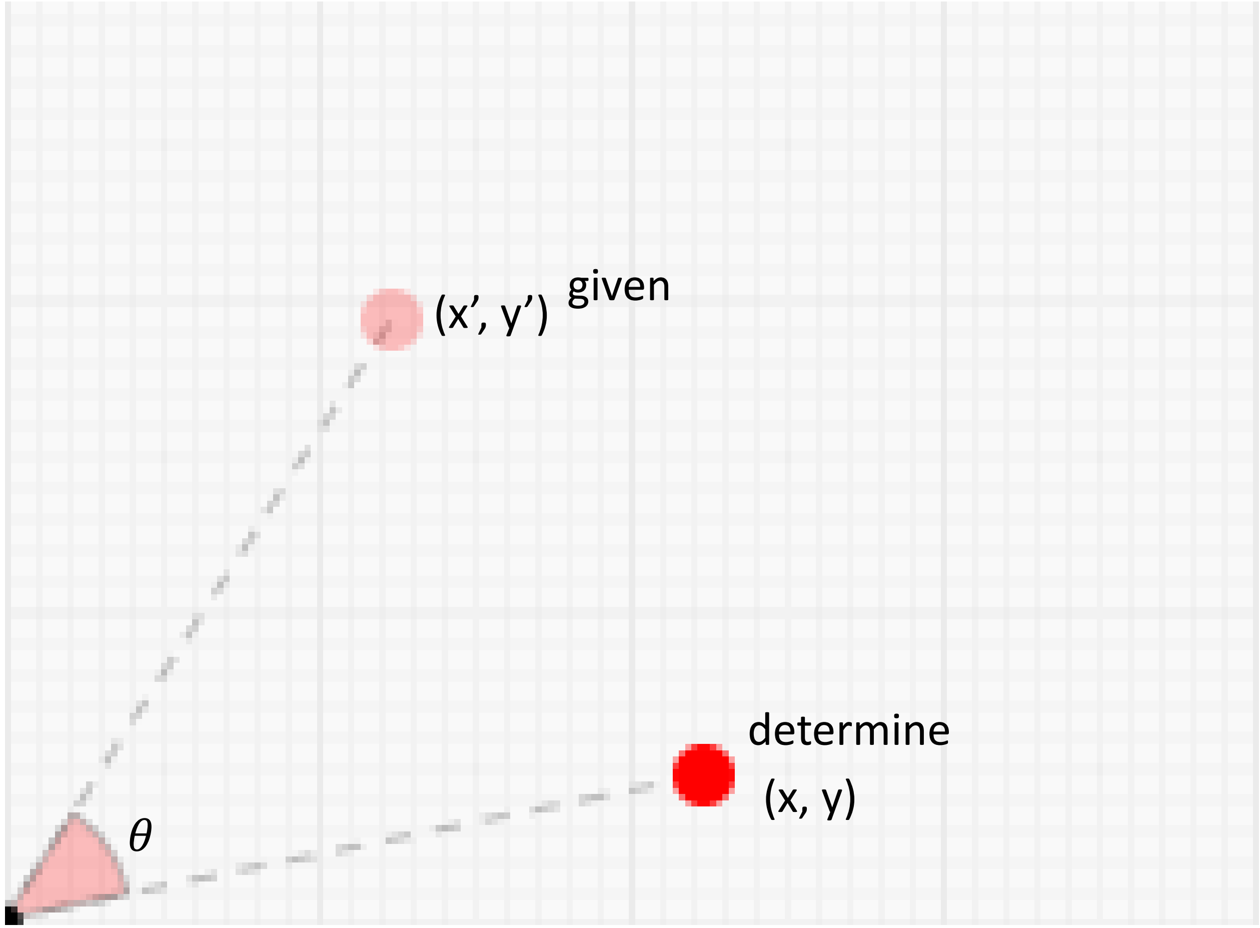


Inverse Rotation

- Given (x', y') that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

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



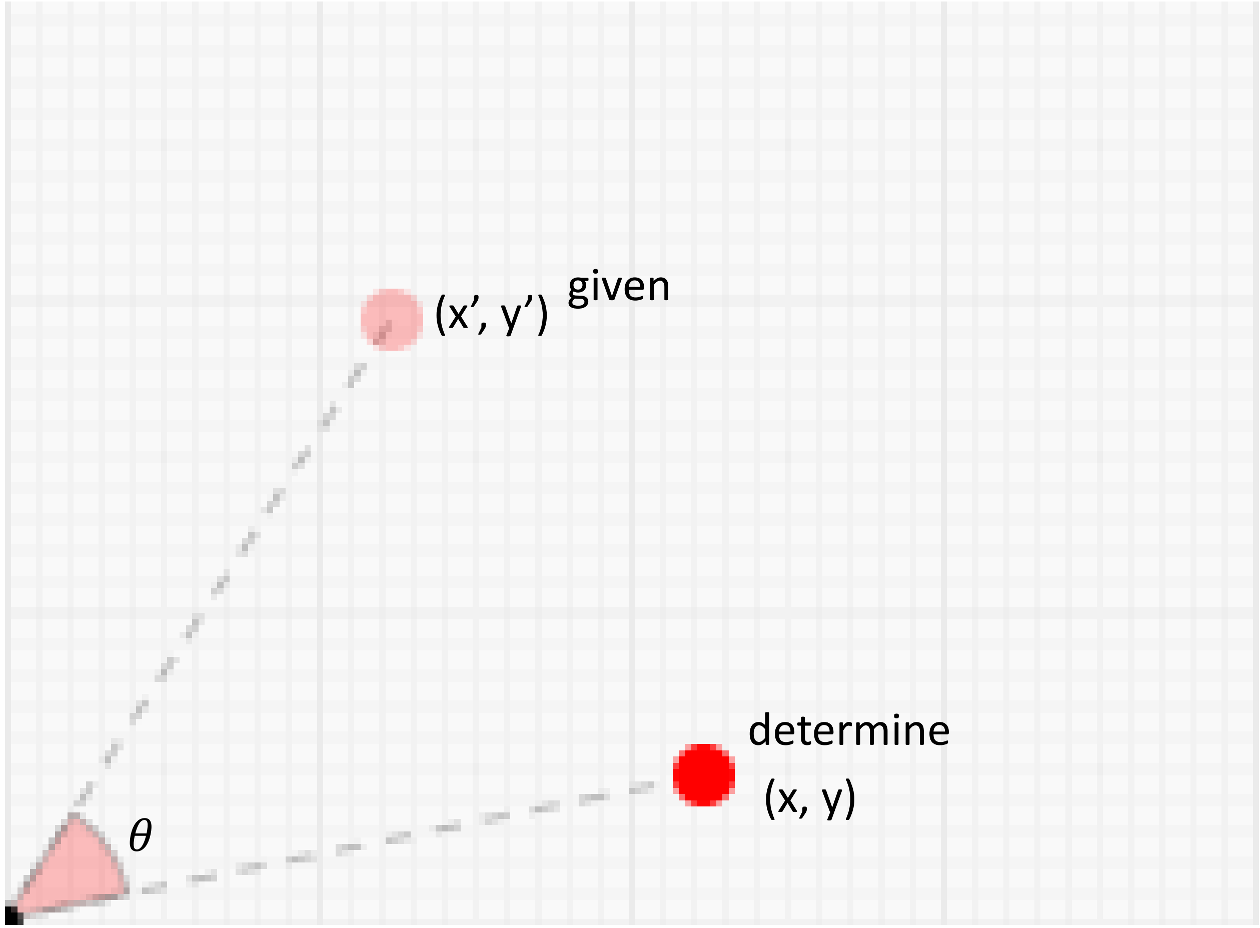
Inverse Rotation

- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

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

$$if\ A = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$



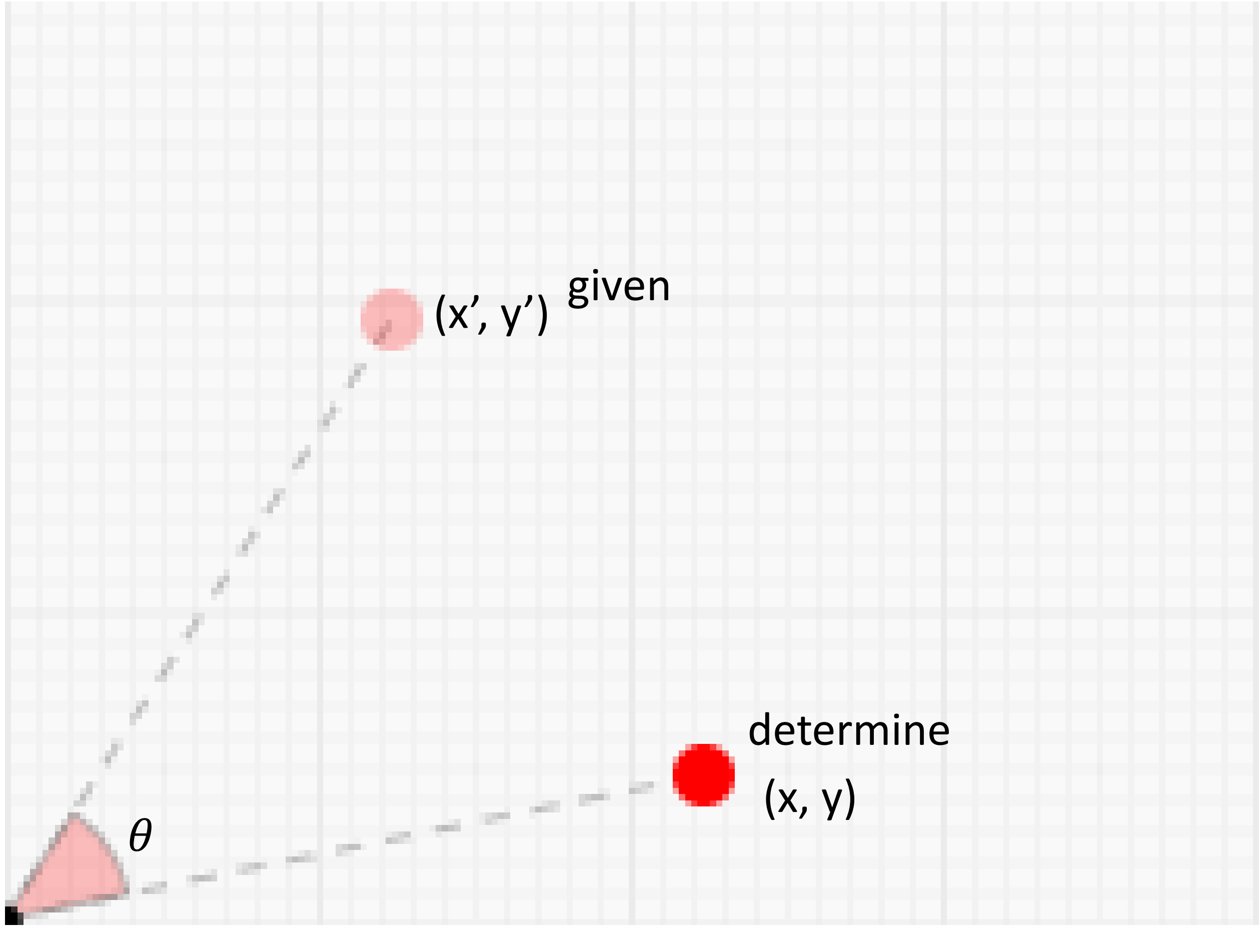
Inverse Rotation

- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

$$if\ A = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix}$$



Inverse Rotation

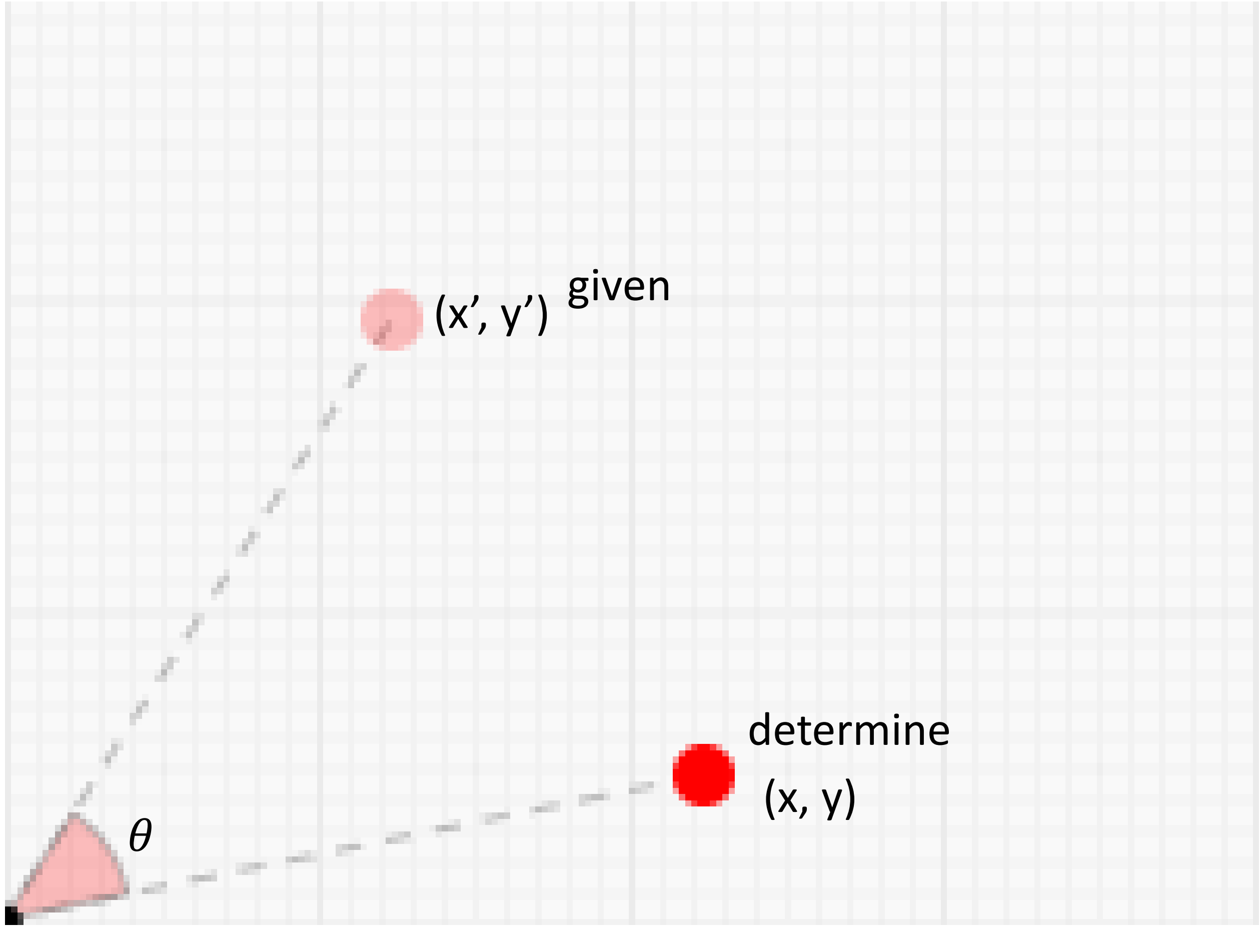
- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

$$if\ A = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix}$$

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$



Inverse Rotation

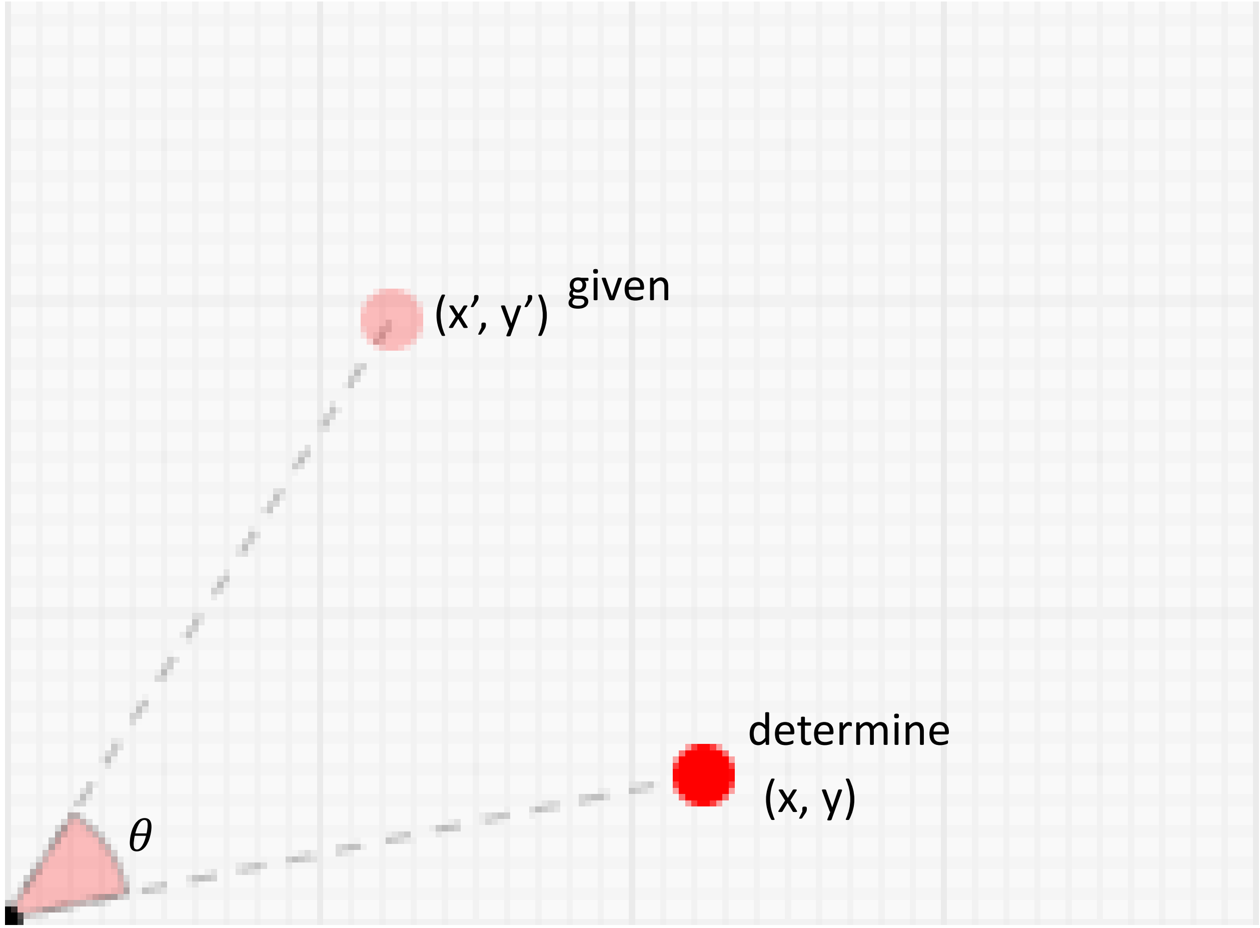
- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

$$if\ A = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix}$$

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$



Inverse Rotation

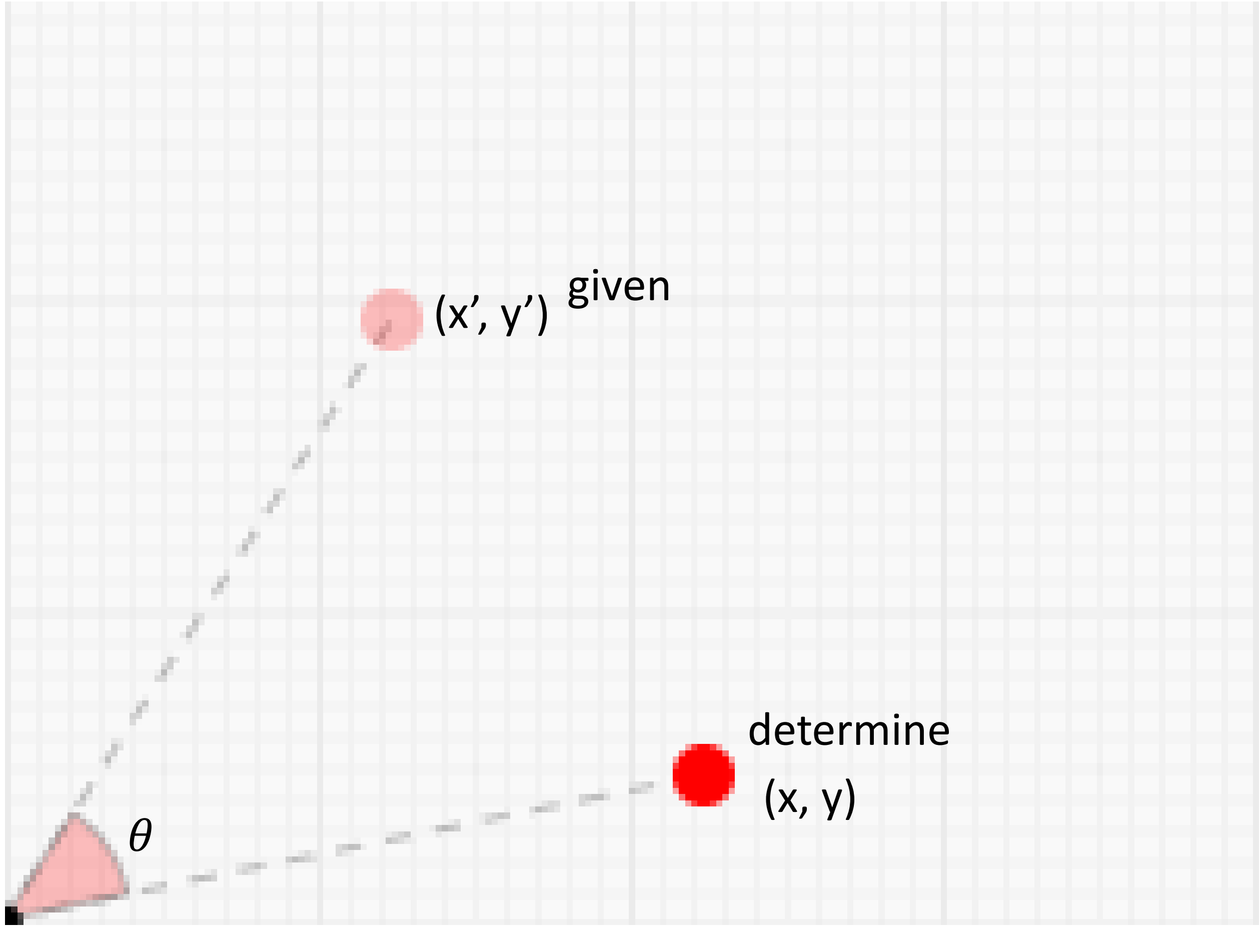
- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

$$if\ A = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = A^{-1} \begin{pmatrix} x' \\ y' \end{pmatrix}$$



Inverse of (2X2) matrix

$$if\ K = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Inverse of (2X2) matrix

$$if K = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$K \; X \; K^{-1} = I \; (identity \; matrix)$$

$$K^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Inverse of (2X2) matrix

$$if\ K = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$K\ X\ K^{-1} = I\ (identity\ matrix)$$

$$K^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

$$A = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

$$A^{-1} = \frac{1}{cos^2\theta + sin^2\theta}\begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix}, cos^2\theta + sin^2\theta = 1$$

$$A^{-1} = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix}$$

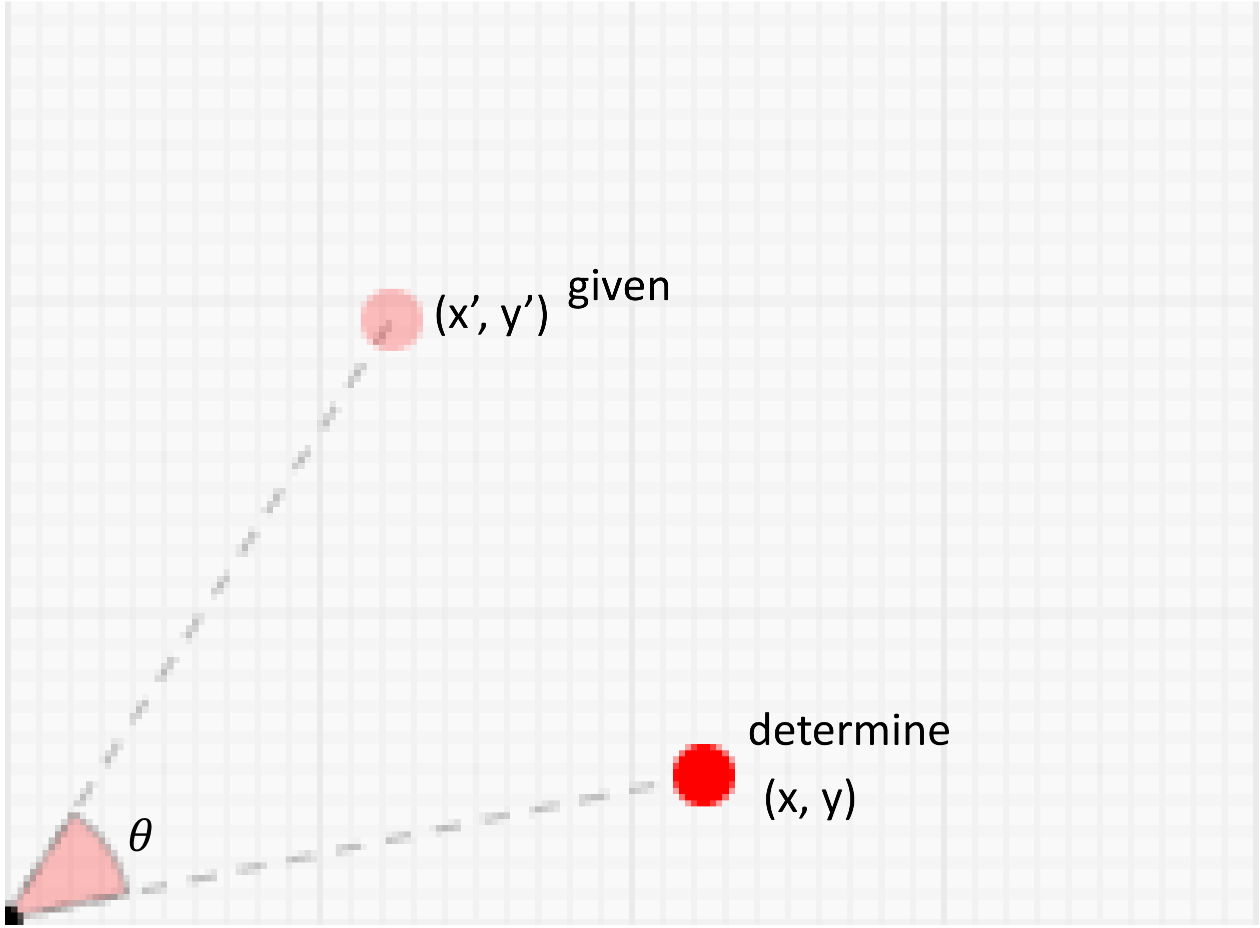
Inverse Rotation

- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

$$\begin{pmatrix} x \\ y \end{pmatrix} = A^{-1} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

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



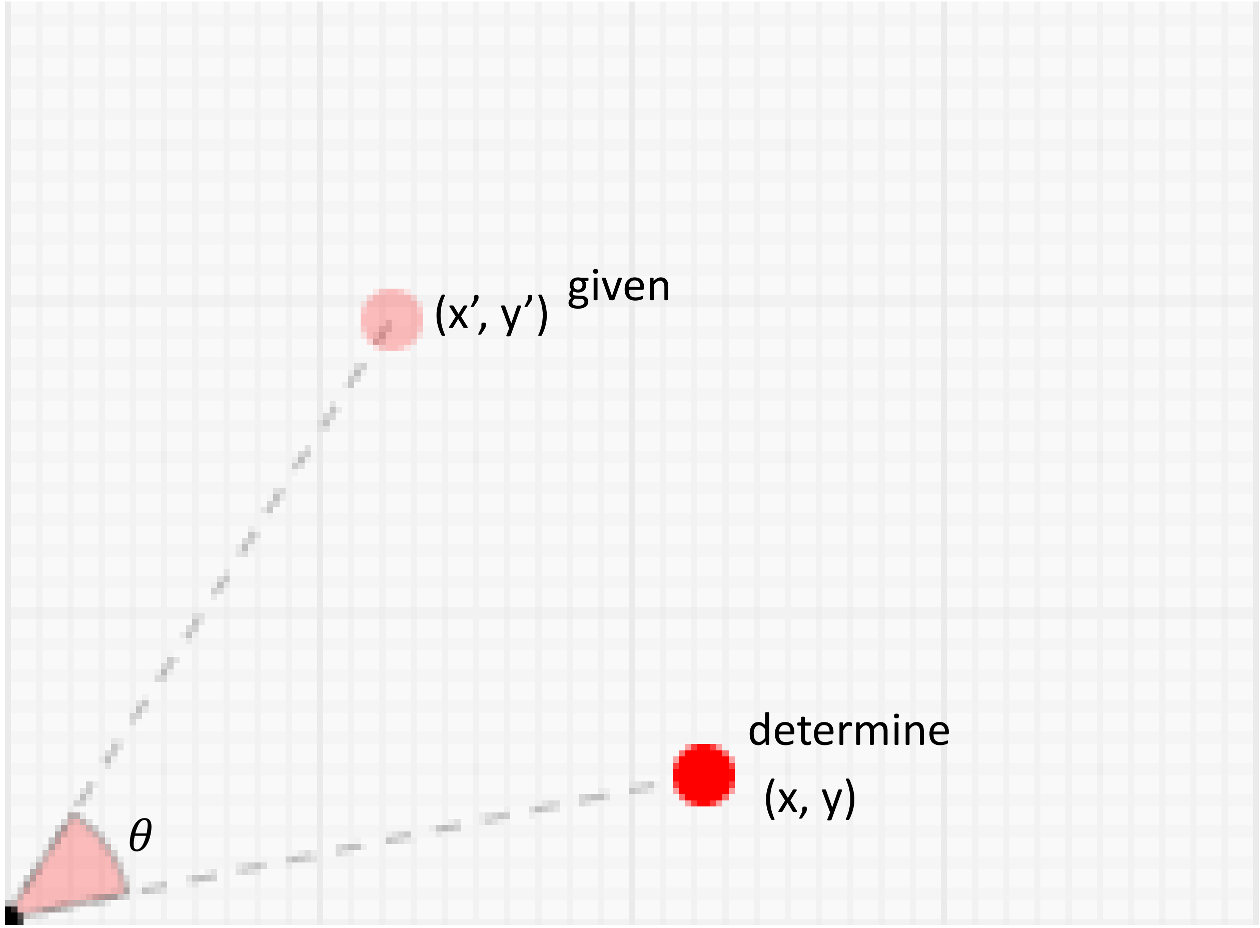
Inverse Rotation

- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Matrix Notation

$$\begin{pmatrix} x \\ y \end{pmatrix} = A^{-1} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix}}_{\text{Inverse rotation matrix}} \begin{pmatrix} x' \\ y' \end{pmatrix}$$



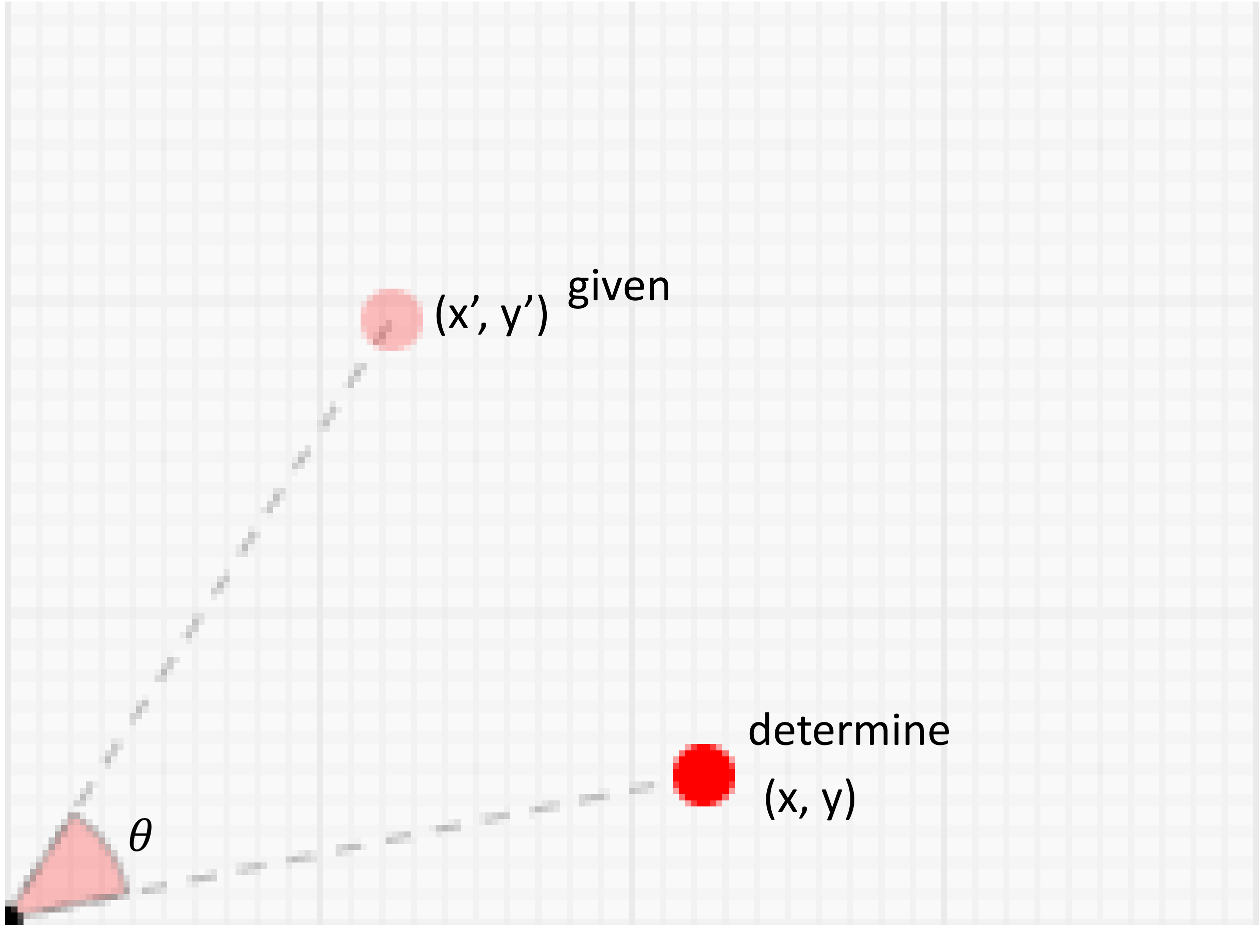
Inverse Rotation (Another way)

- Given(x', y')that has
  - Undergone rotation by θ
- Find original location (x, y)

Rotate (x’, y’) by (- θ)

Matrix Notation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} \begin{pmatrix} x\prime \\ y\prime \end{pmatrix}$$



Inverse Rotation (Another way)

- $\begin{pmatrix} x \\ y \end{pmatrix}$ = $\begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} \begin{pmatrix} x\prime \\ y\prime \end{pmatrix}$
- Cos is even function: $\cos(-\theta)$ = $\cos(\theta)$
- Sin is odd function: $\sin(-\theta) = -\sin(\theta)$
- $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x\prime \\ y\prime \end{pmatrix}$

  Inverse rotation matrix

Part 1: Forward Rotation

Input

Forward rotate by
$\theta = 0.5$ radians
(~28 deg)

Output

Forward Rotation

Forward rotate by
$\theta = 0.5$ radians
(~28 deg)

- Input:
  - Image
  - Theta (θ) - angle of rotation

Assumption

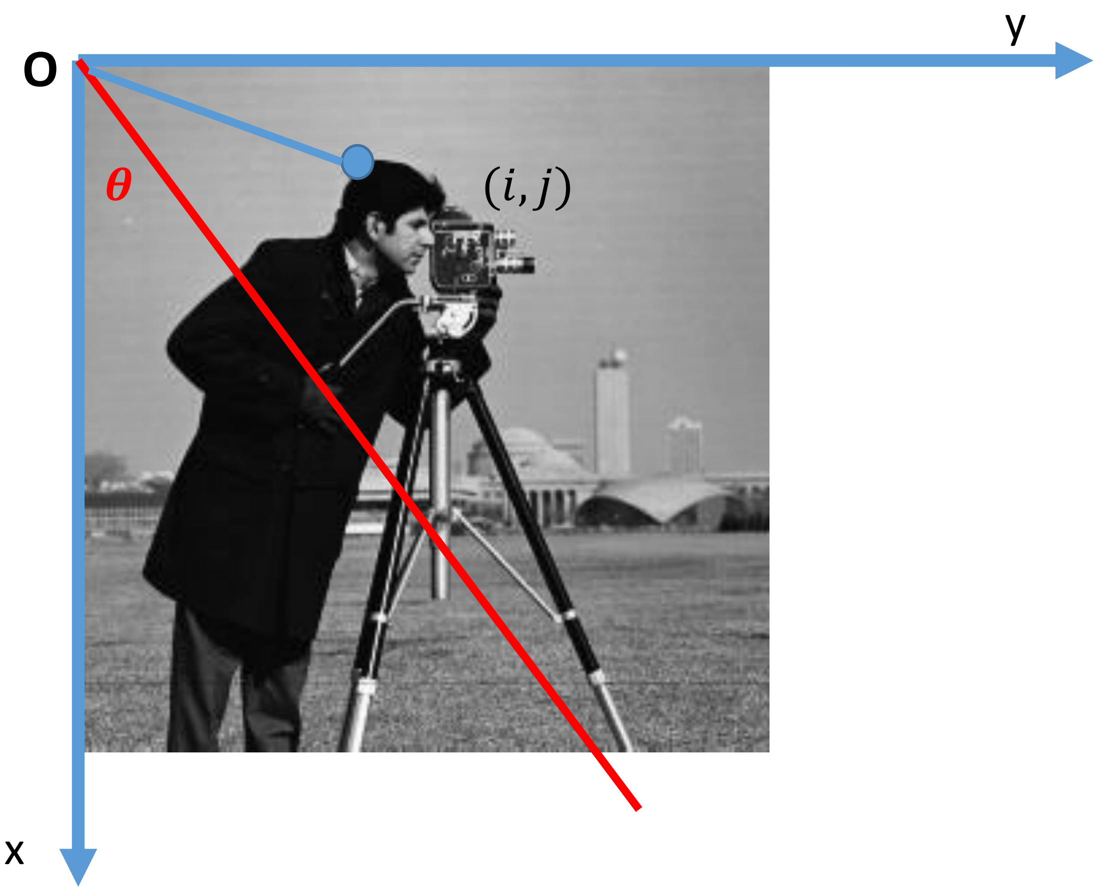
- To simplify,
  - We assume the following co-ordinate system
    - O-Origin

Assumption

- To simplify,
  - We assume the following co-ordinate system
  - Theta denote rotation from x-axis, towards y-axis

Rotate each pixel

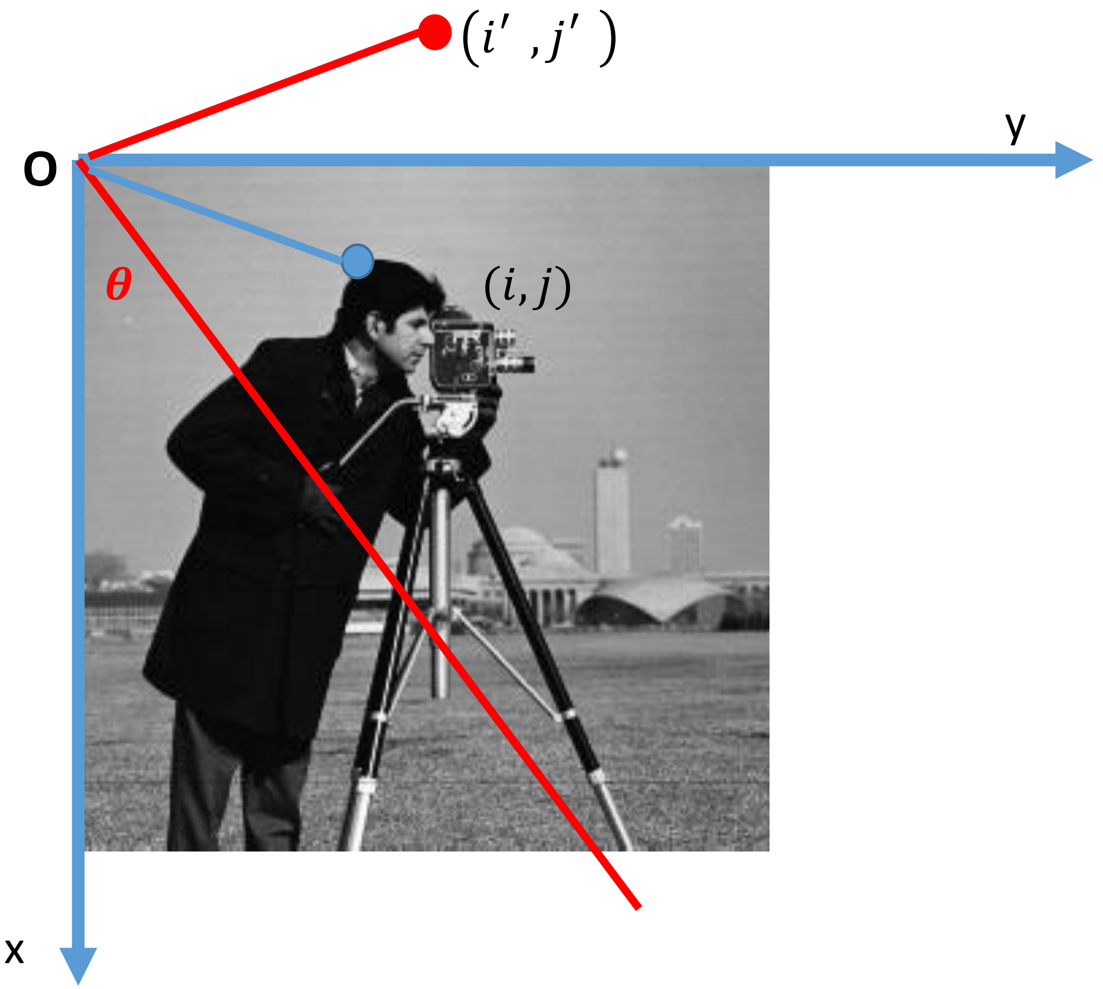
- Let R be the rotated image and I the original image.



Rotate each pixel

- Let R be the rotated image and I the original image.
- For every pixel (i, j)in I
  - apply rotation by theta to get (i', j') in new image.

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

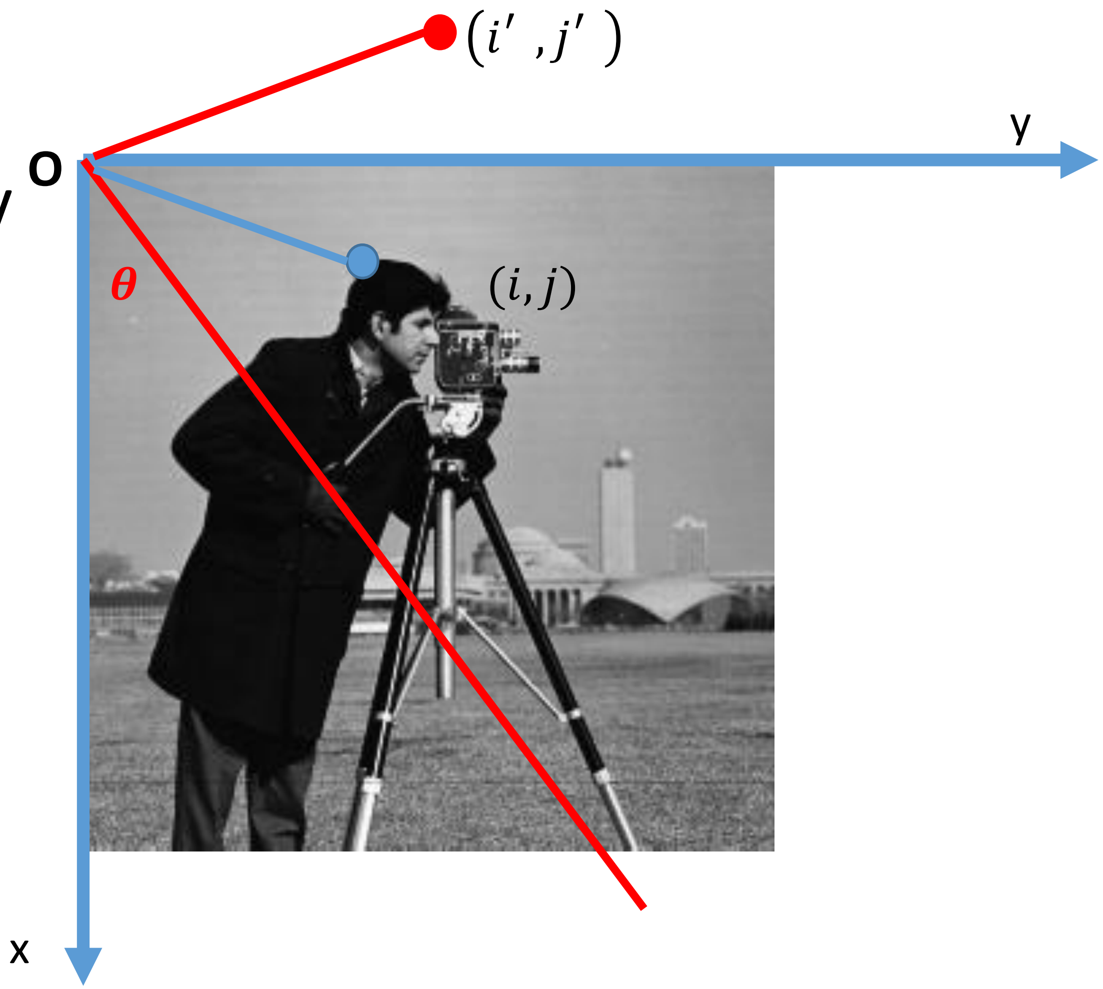


Rotate each pixel

- Let R be the rotated image and I the original image.
- For every pixel (i, j) apply rotation by theta to get $(i' \ , j' \ )$ in new image.

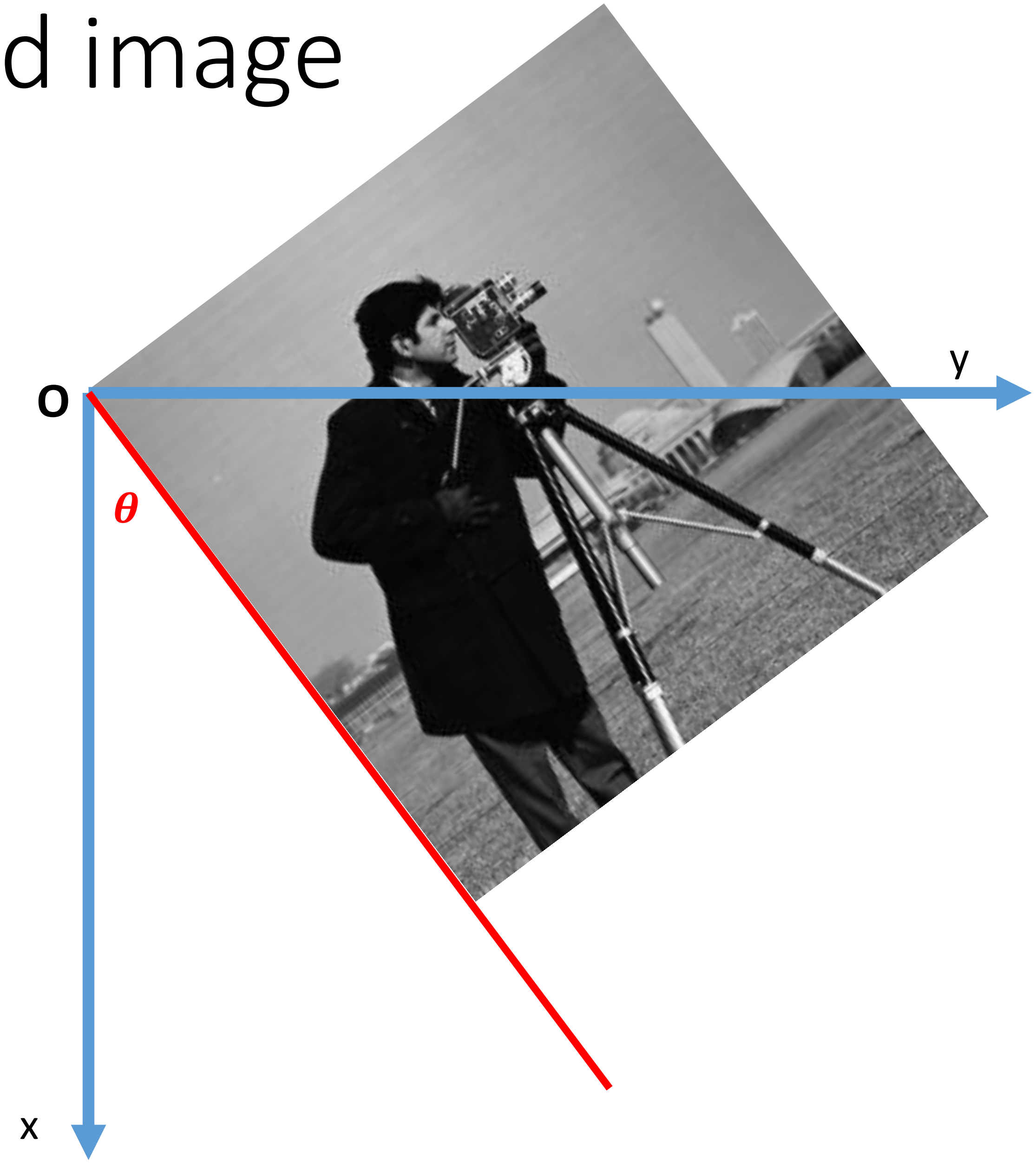
$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

- $R(i', j') = I(i, j)$

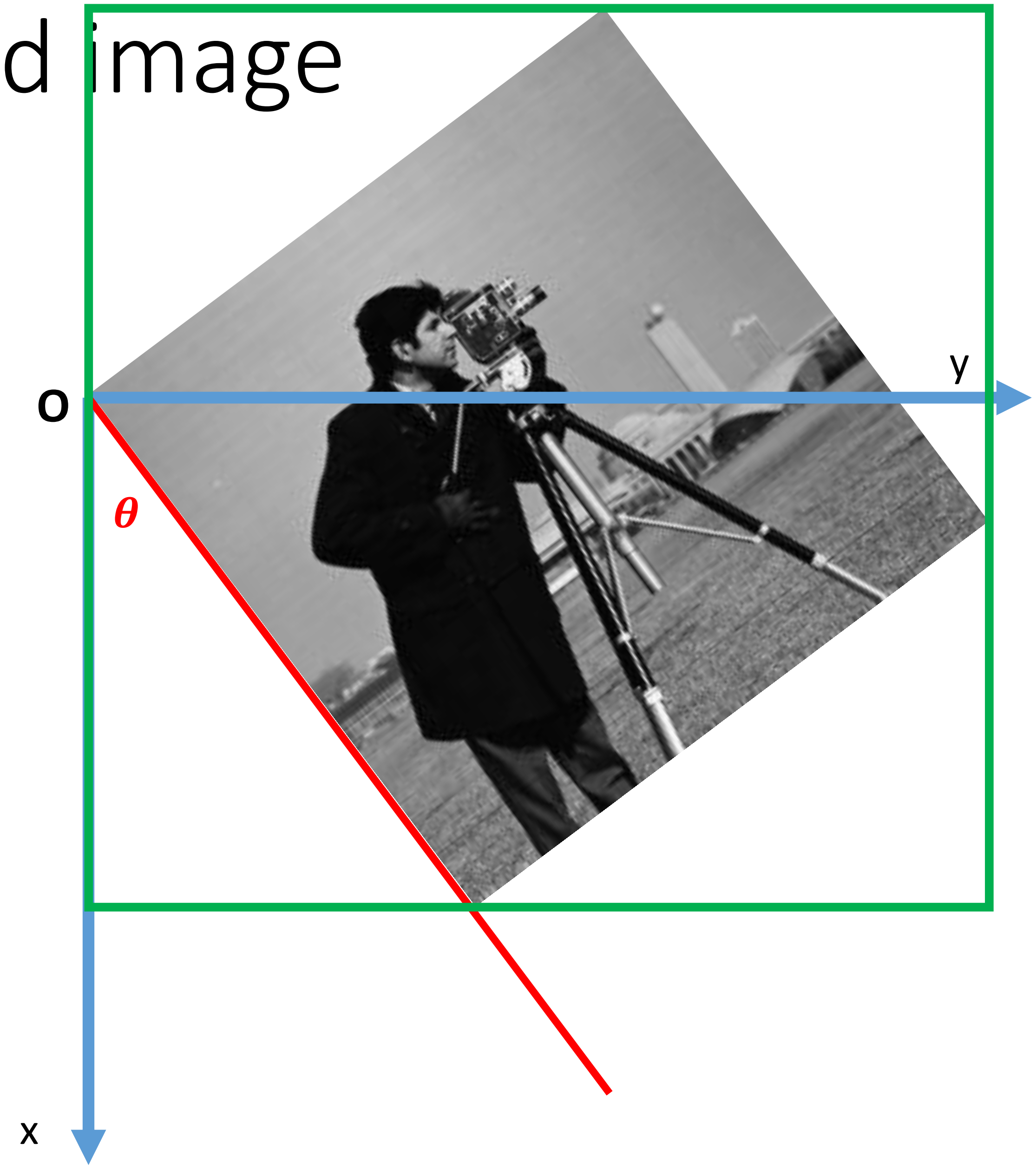


Determine size of rotated image

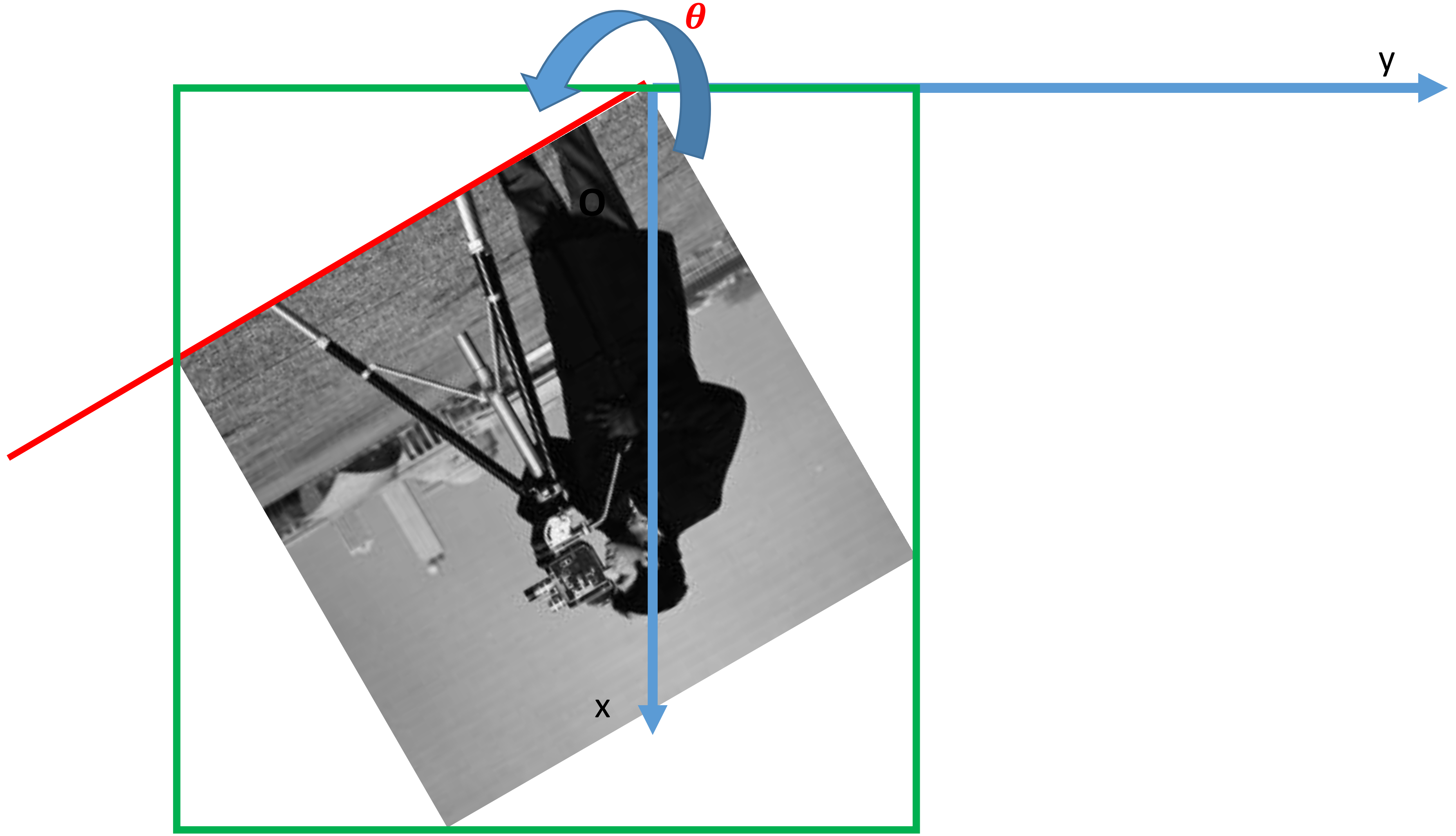
Determine size of rotated image



Determine size of rotated image

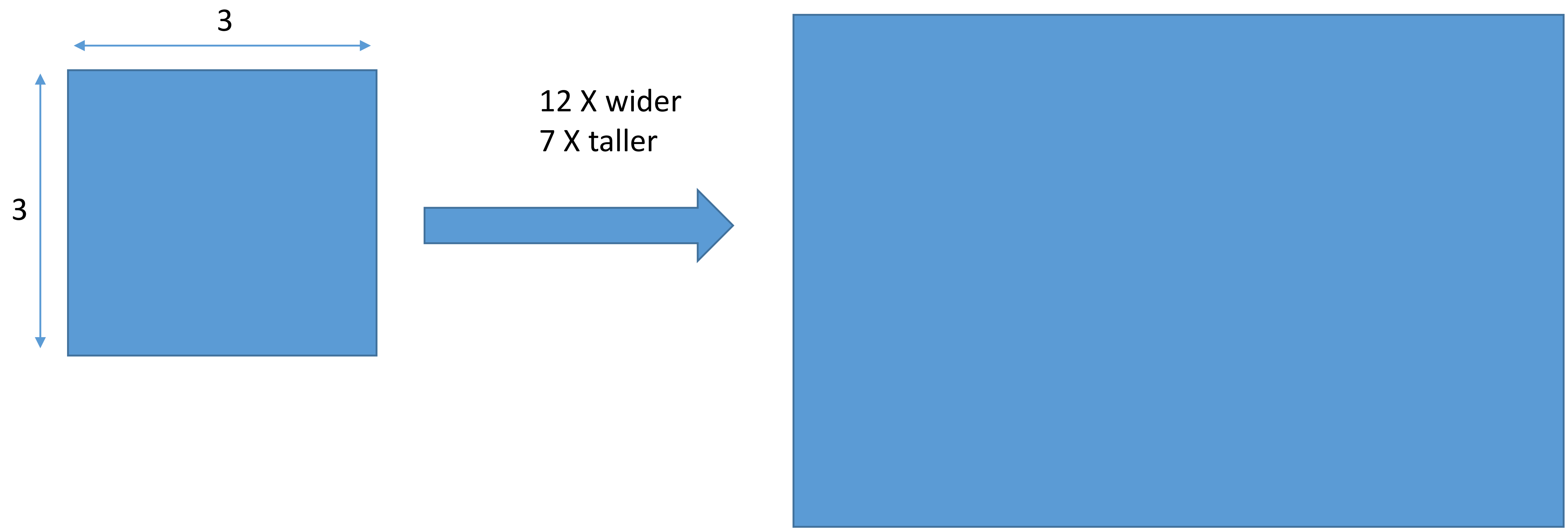


Determine size of rotated image



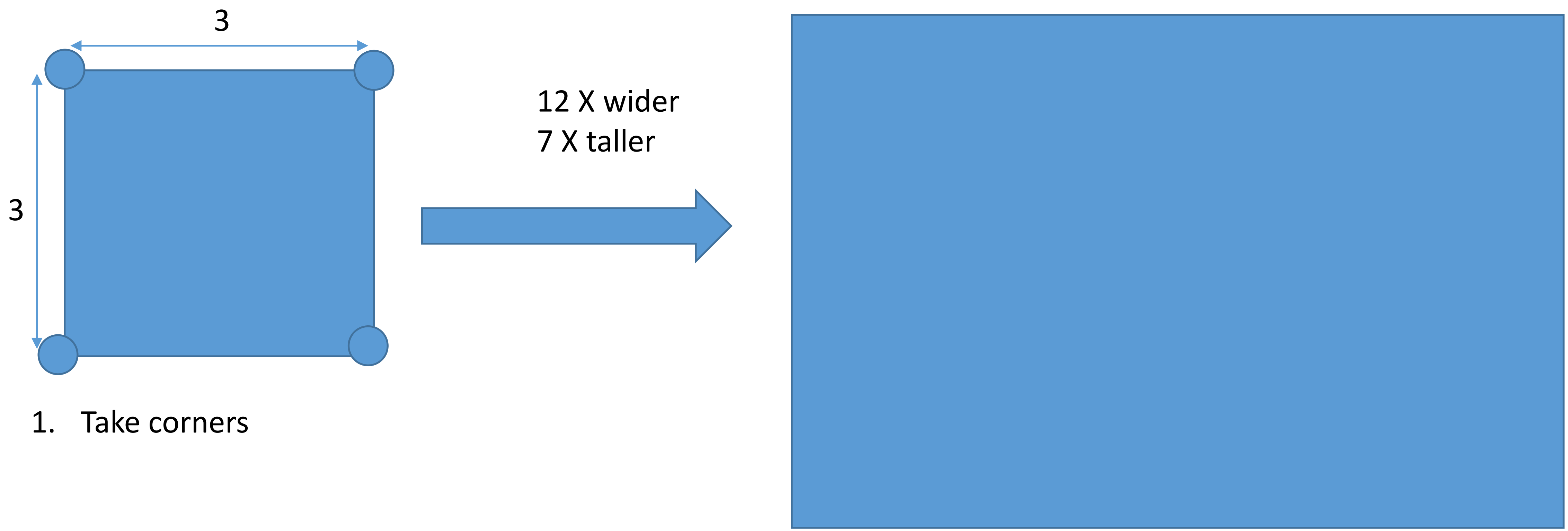
Example Resizing

What is the shape of the image I need to create



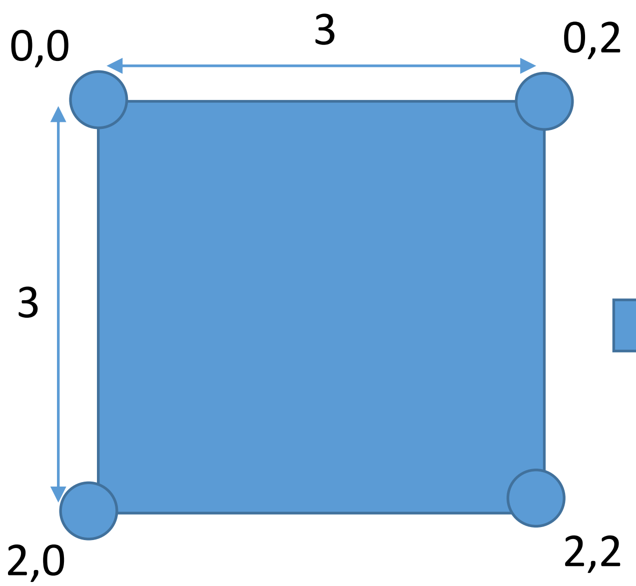
Example Resizing

What is the shape of the final image that I need to create



1. Take corners

Example Resizing



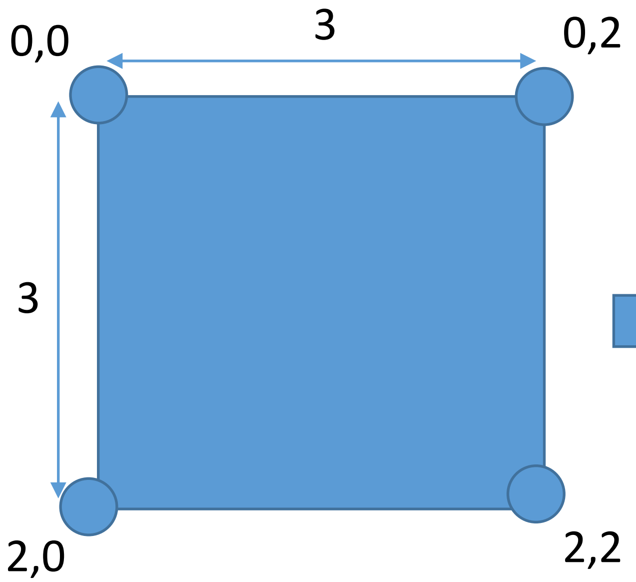
1. Take corners
2. Apply transformation

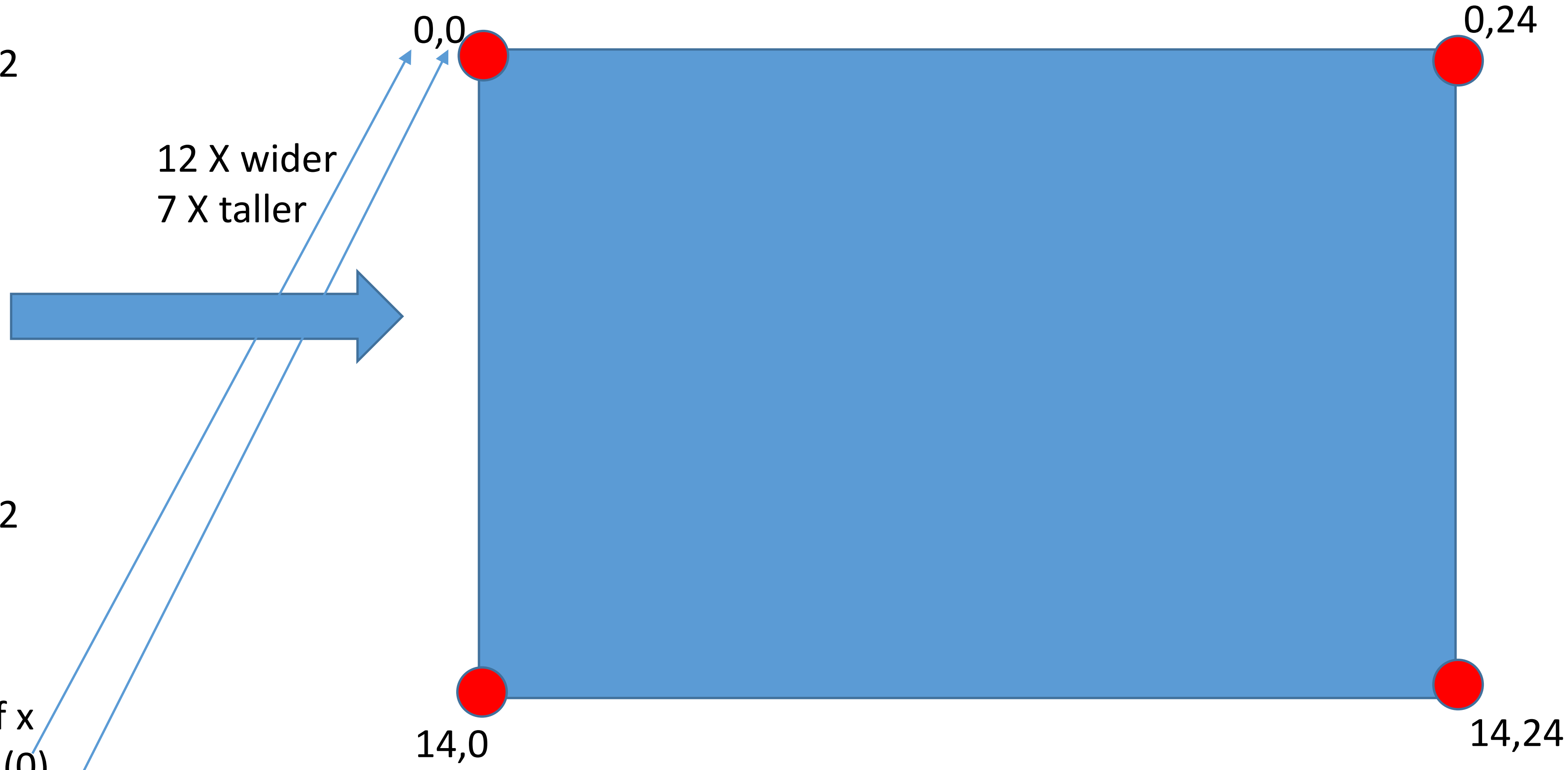
12 X wider
7 X taller

What is the shape of the final image that I need to create

Example Resizing

What is the shape of the final image that I need to create

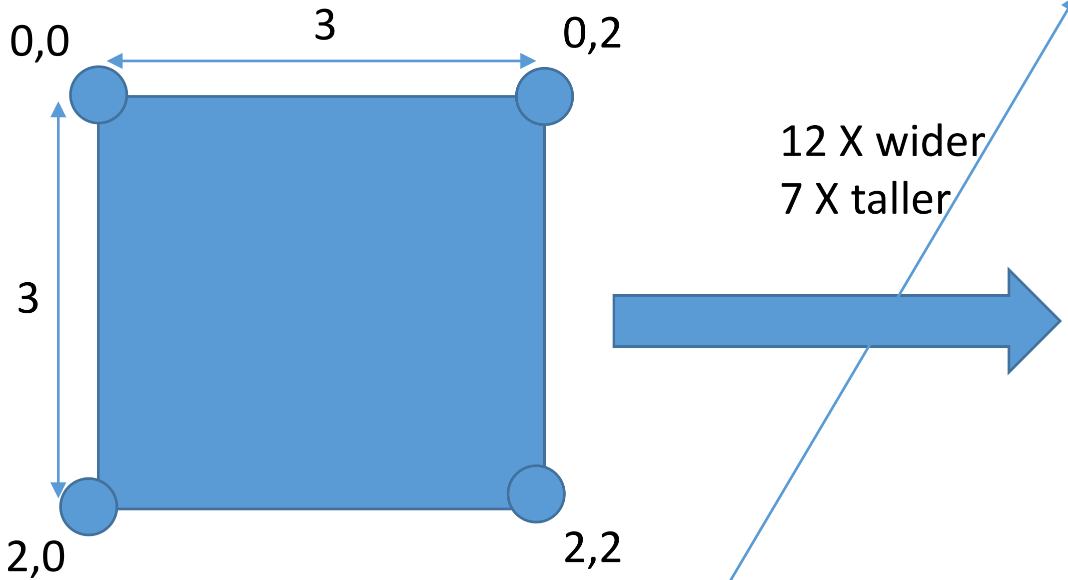




1. Take corners
2. Apply transformation
3. Image size
    1. min_x = minimum of x value amongst four (0)
    2. min_y = minimum of y value amongst the (0)

Example Resizing

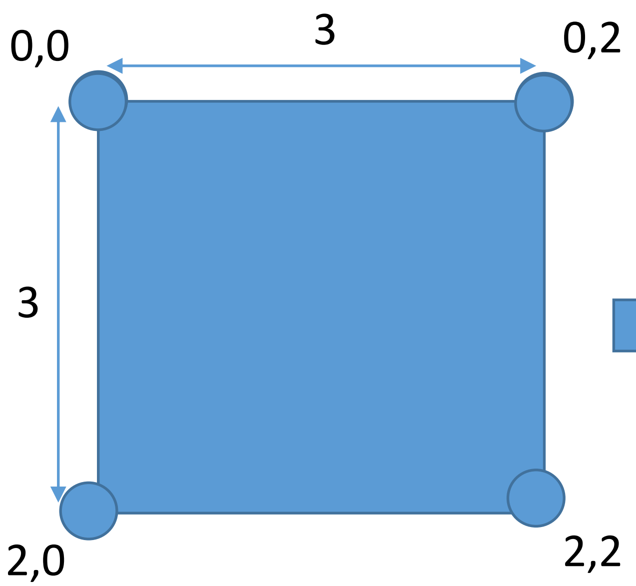
What is the shape of the final image that I need to create

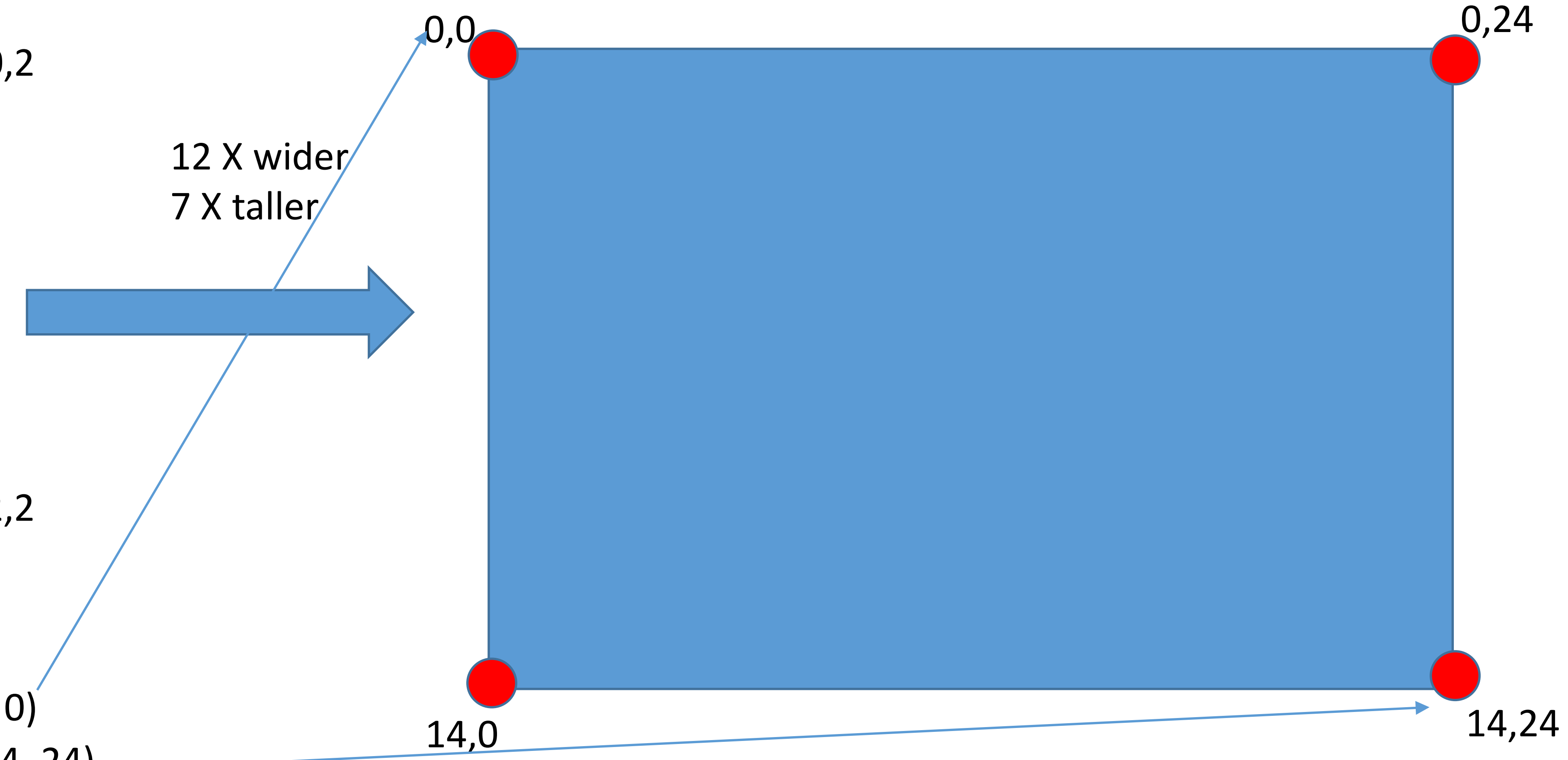


1. Take corners
2. Apply transformation
3. Image size
   1. (min_x, min_y)= (0, 0)
      (max_x, max_y)= (14, 24)

Example Resizing

What is the shape of the final image that I need to create

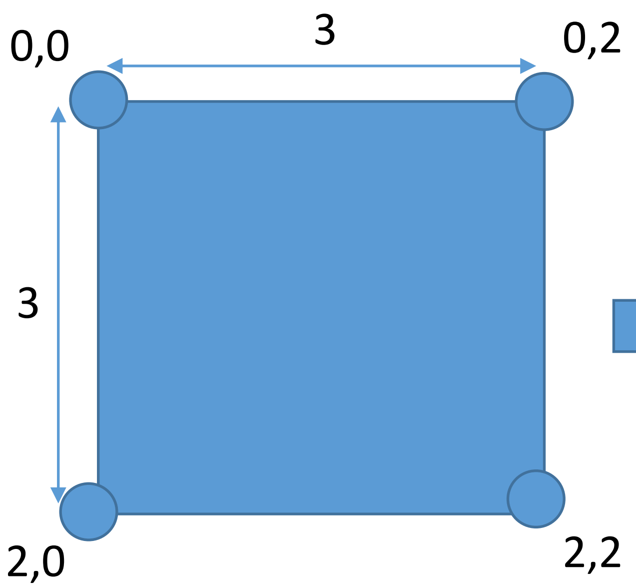


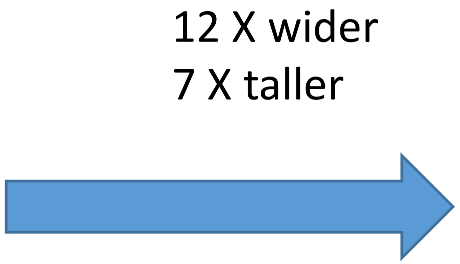


1. Take corners
2. Apply transformation
3. Image size
   1. (min_x, min_y)= (0, 0)
      (max_x, max_y)= (14, 24)
   2. Rows = max_x - min_x= 14 - 0
      cols= max_y - min_y= 24 - 0

Example Resizing

What is the shape of the final image that I need to create



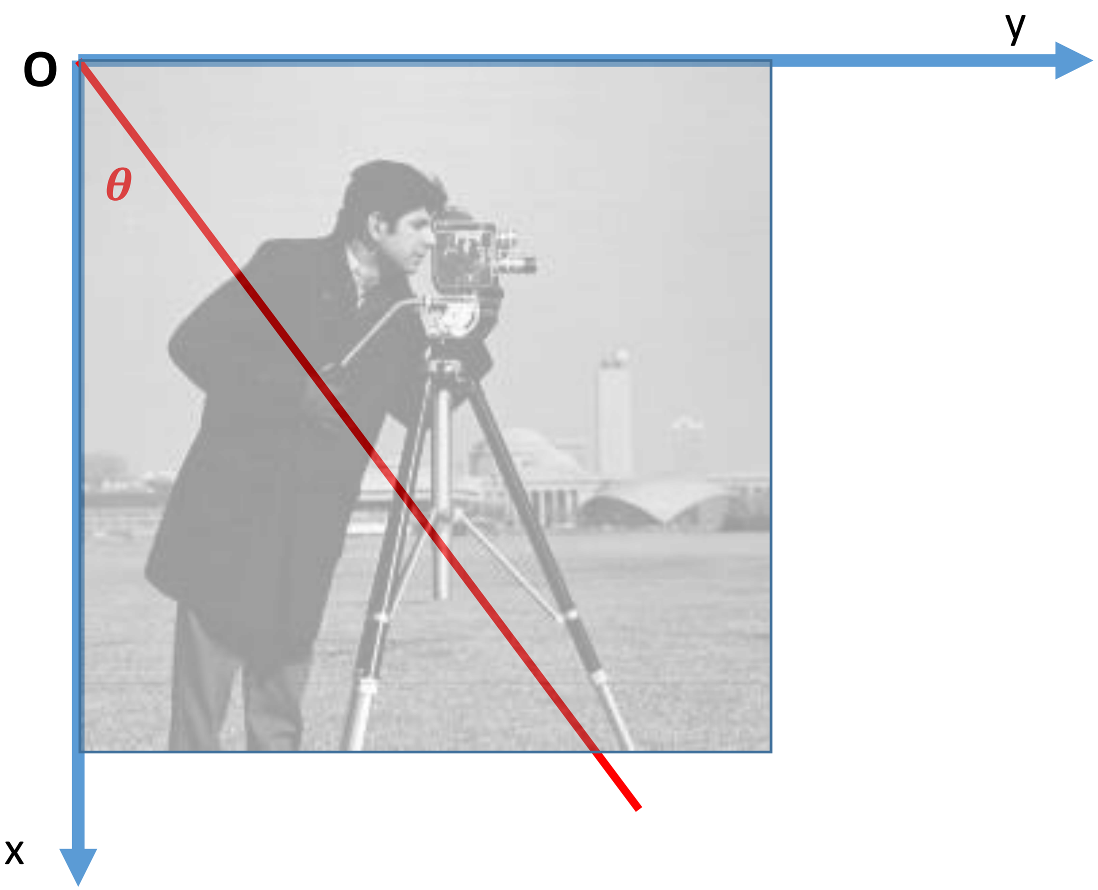


1. Take corners
2. Apply transformation
3. Image size
   1. Rows = max_x - min_x= 14 - 0

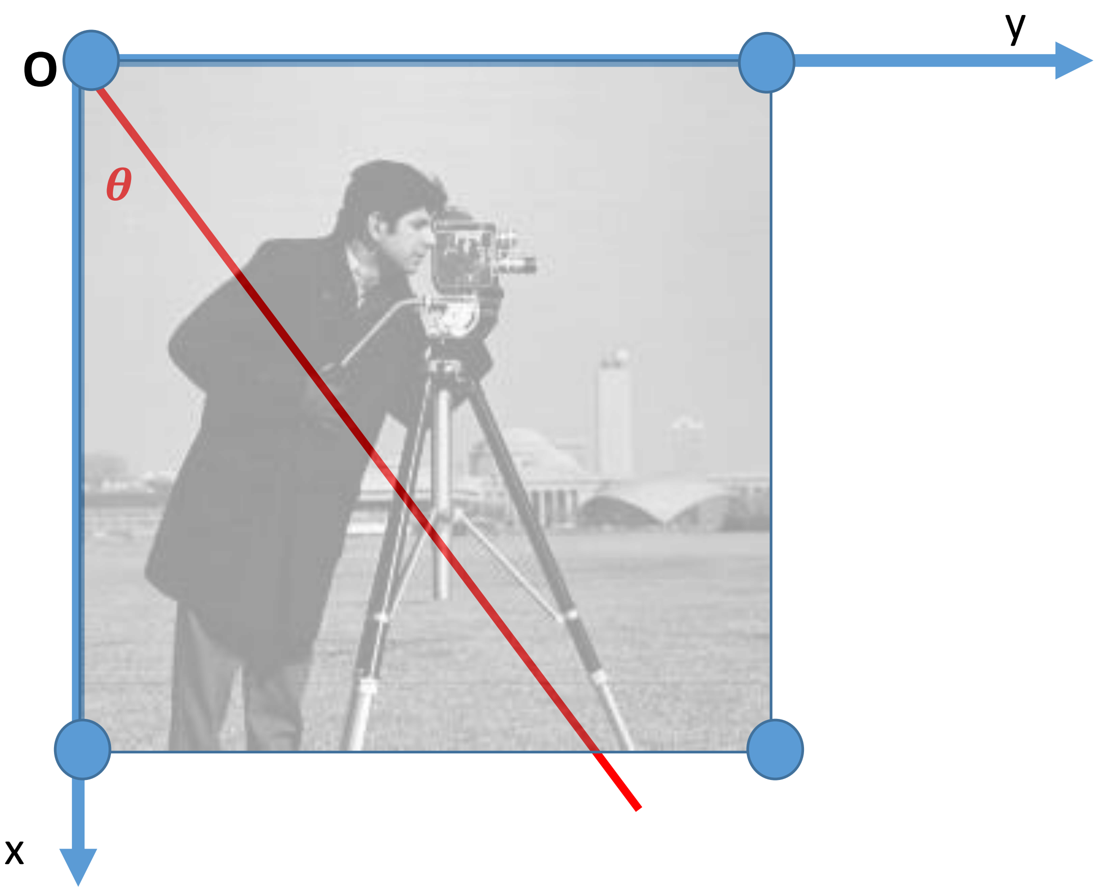
      cols= max_y - min_y= 24 – 0

   Size of rotated image = (rows, cols) = (14, 24)

Determine size of rotated image

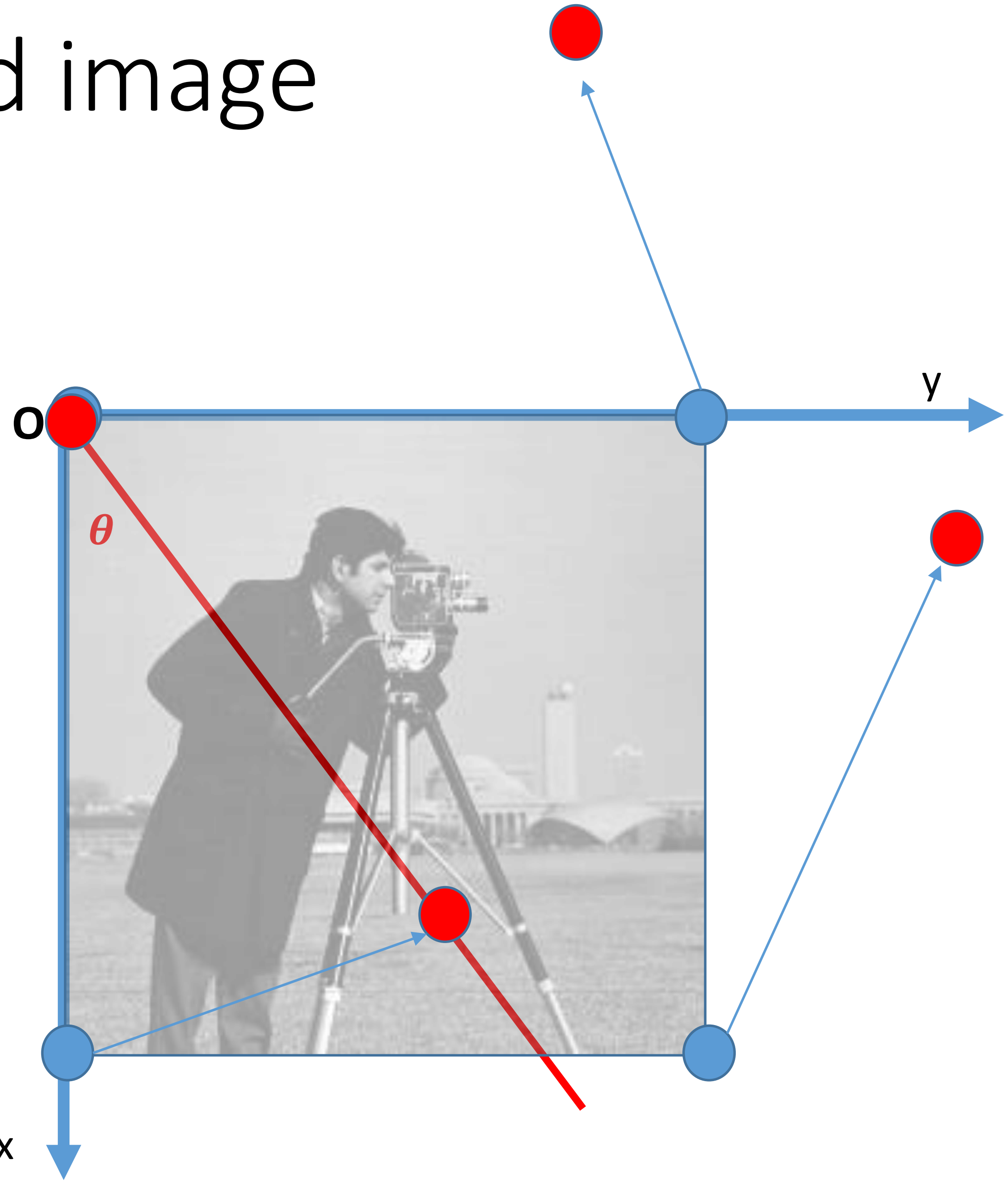


Determine size of rotated image



Determine size of rotated image

1. Compute Rotation matrix
2. Rotate corners



Determine size of rotated image

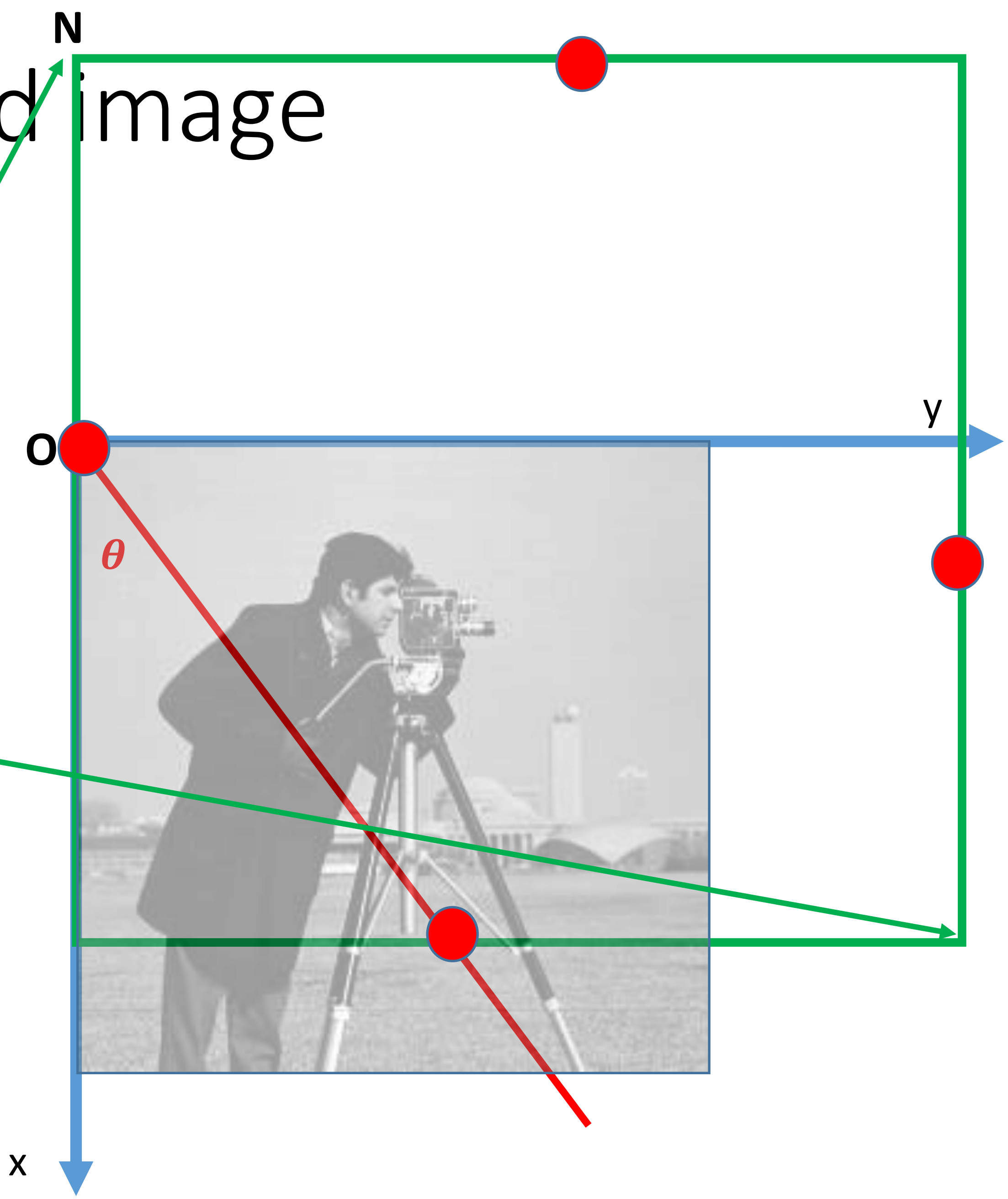
Let I be the original image

1. Compute Rotation matrix
2. Rotate corners

   min_x = minimum of x value amongst the rotated corners

   min_y, max_x …

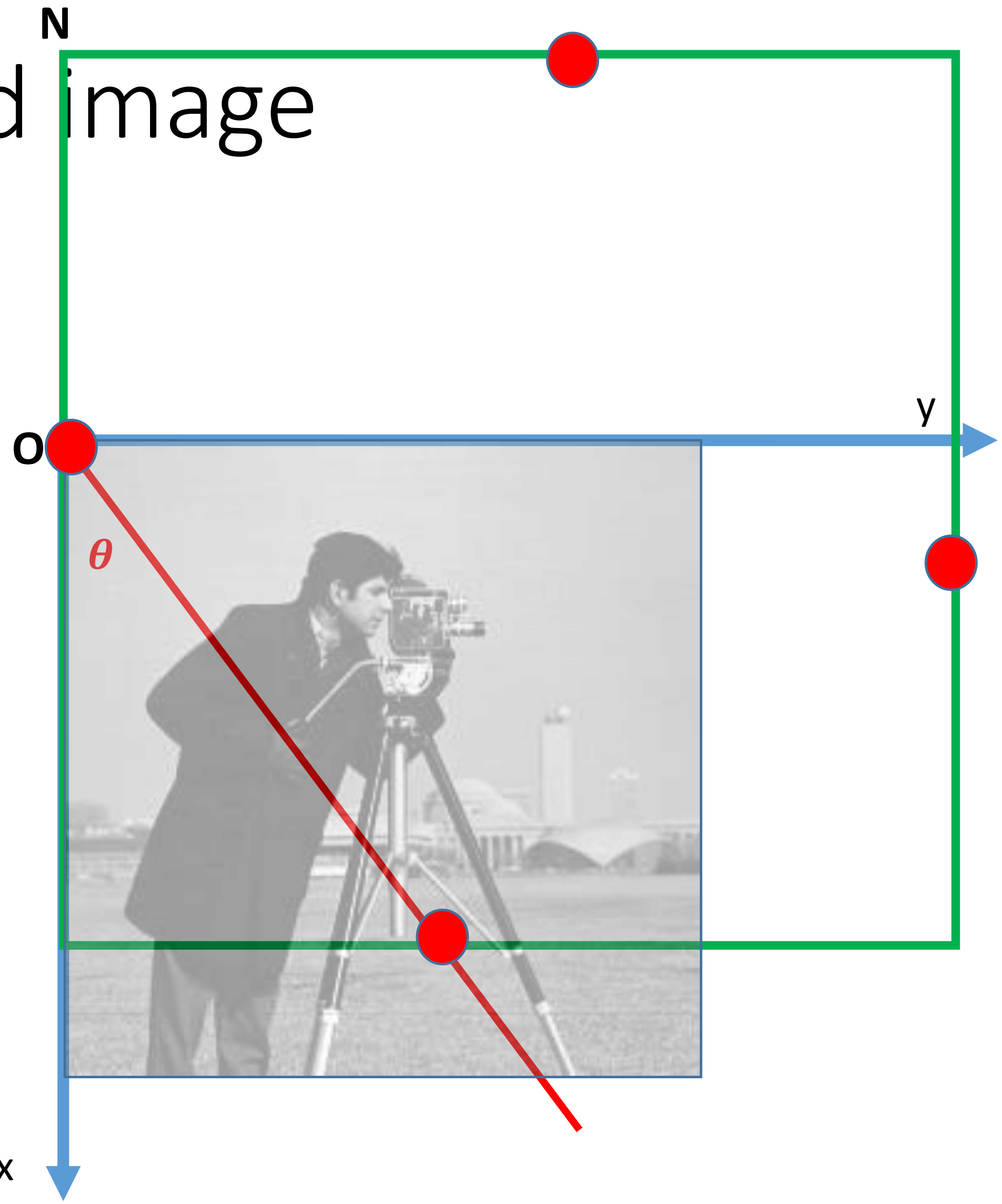
   1. Get (min_x, min_y) (top left of the image)
      1. Is also the new origin (N)
   2. Get (max_x, max_y) (bottom right)



Determine size of rotated image

Let I be the original image

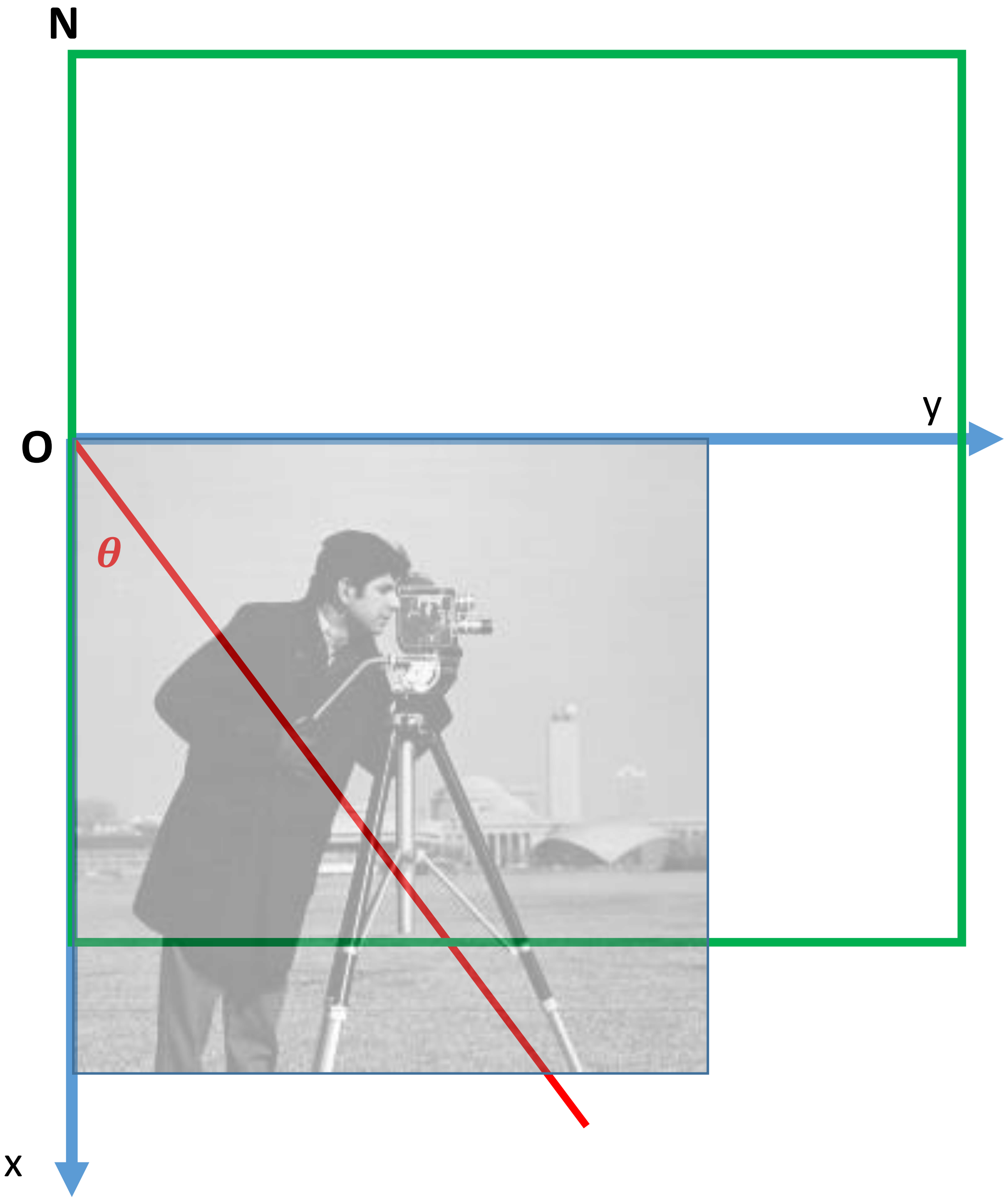
1. Compute Rotation matrix
2. Rotate corners
    1. Get (min_x, min_y) (top left of the image)
    2. Get (max_x, max_y) (bottom right)
        1. rows = $max_x - min_x$
        2. cols = max_y - min_y
    3. Size of rotated image = (rows, cols)



Create empty image

Let I be the original image

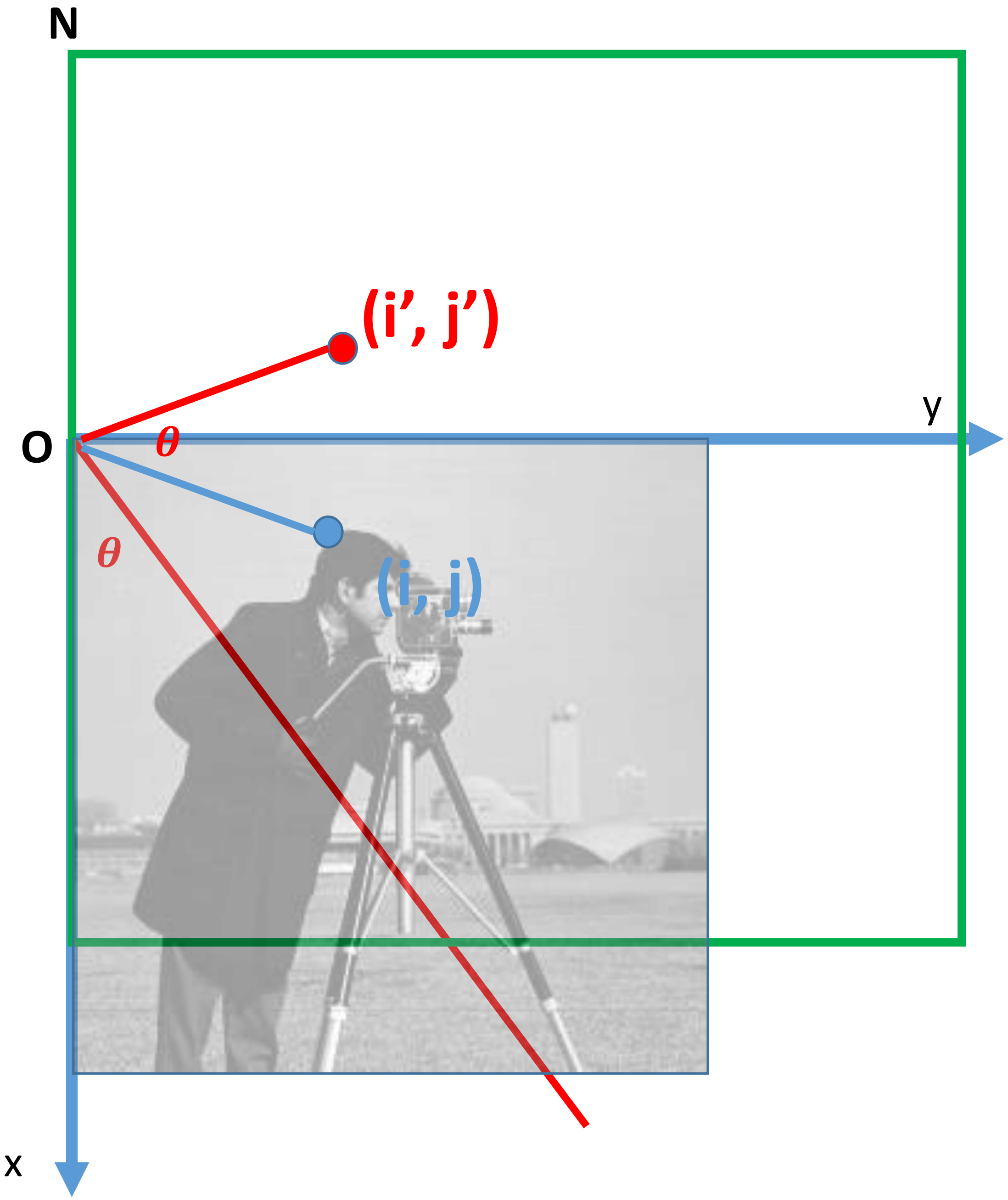
1. Compute Rotation matrix
2. Rotate corners
    1. Get (min_x, min_y) (top left of the image)
    2. Get (max_x, max_y) (bottom right)
        1. rows = $max_x - min_x$
        2. cols = max_y - min_y
    3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)



Rotate each pixel

Let I be the original image

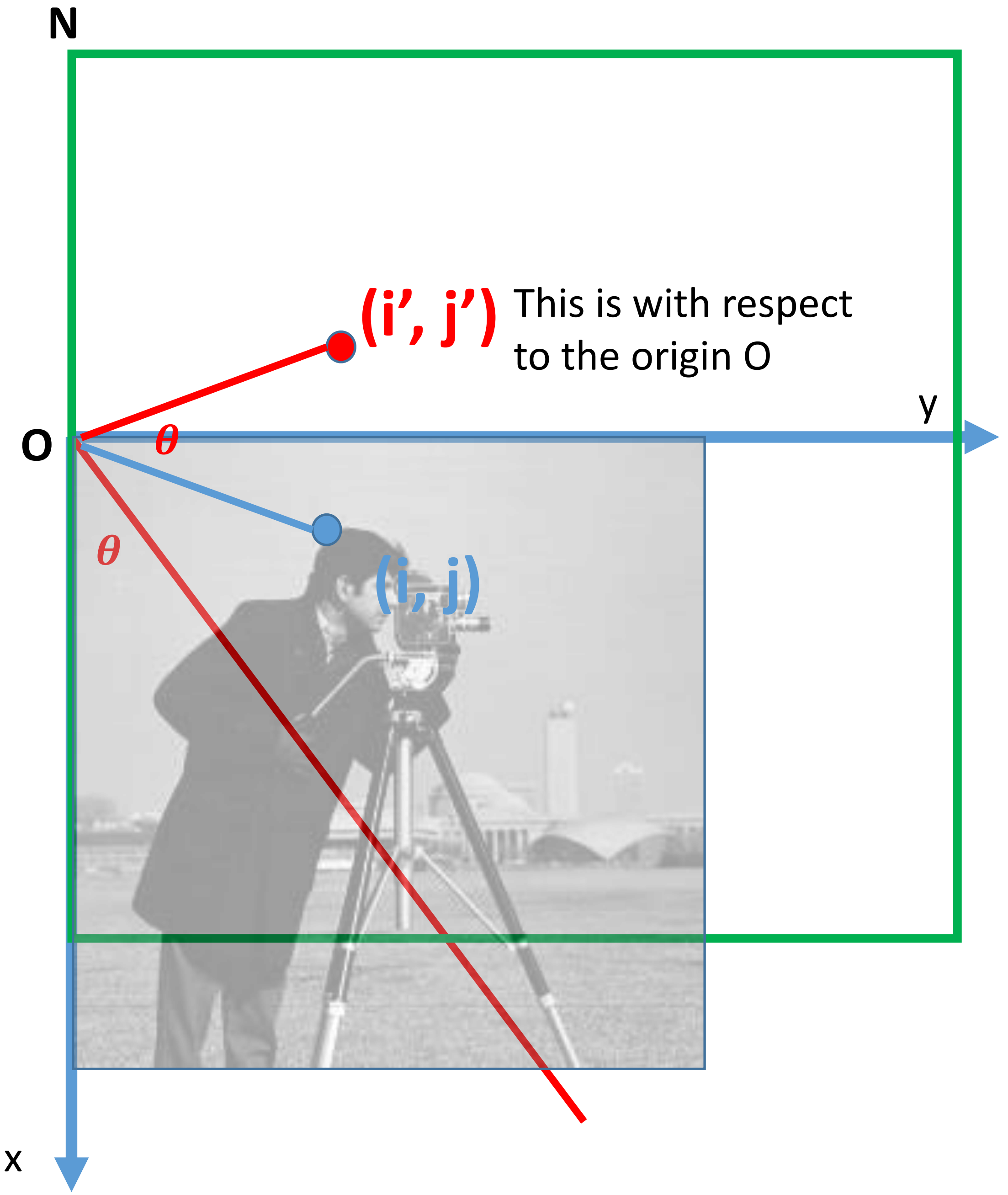
1. Compute Rotation matrix
2. Rotate corners
   1. Get (min_x, min_y) (top left of the image)
   2. Get (max_x, max_y) (bottom right)
      1. Rows = $max_x - min_x$
      2. Cols = max_y - min_y
   3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)
4. For each (i, j) in original image:
   1. Compute rotated location (I', j')



Rotate each pixel

Let I be the original image

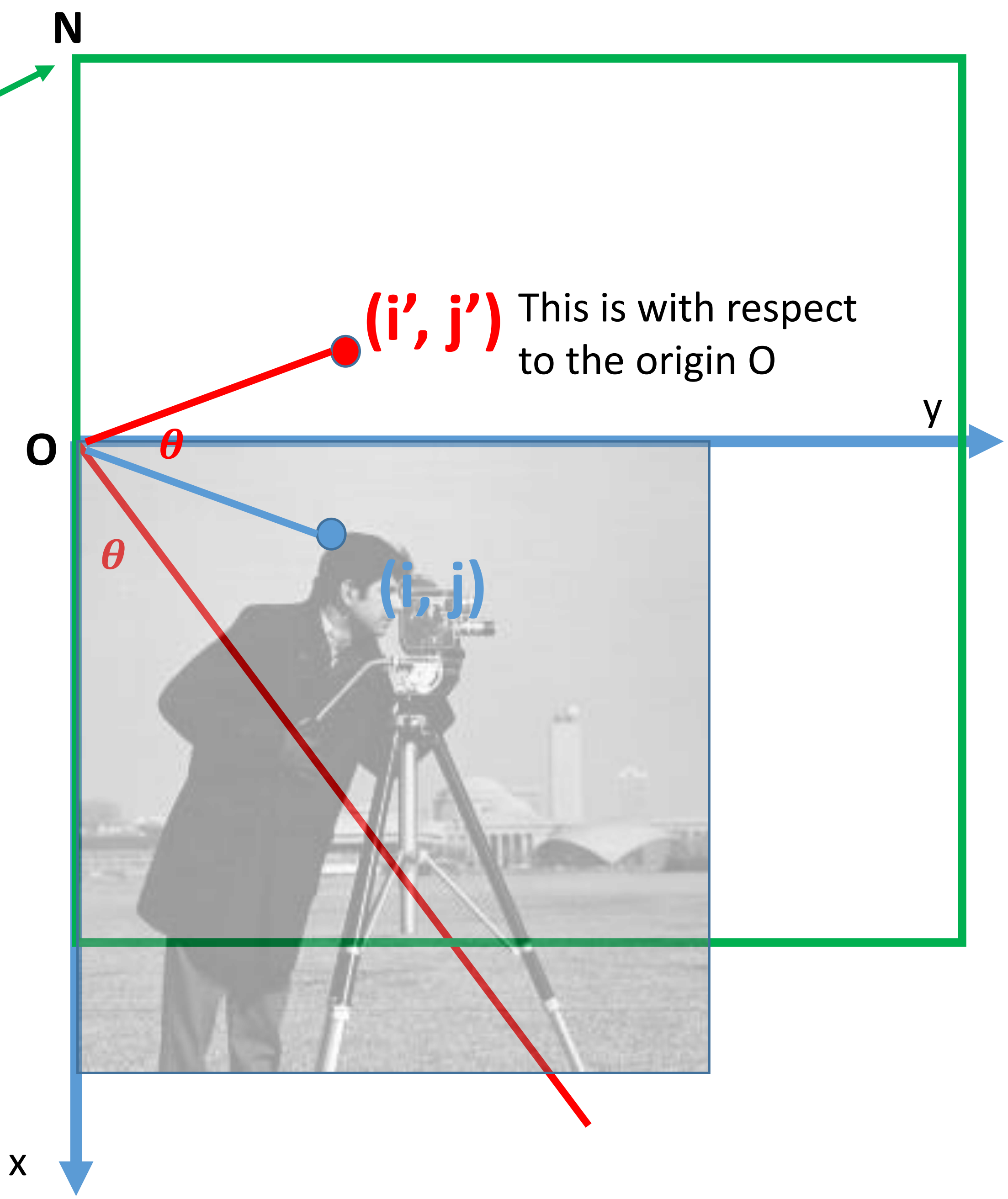
1. Compute Rotation matrix
2. Rotate corners
    1. Get (min_x, min_y) (top left of the image)
    2. Get (max_x, max_y) (bottom right)
        1. Rows = $max_x - min_x$
        2. Cols = max_y - min_y
    3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)
4. For each (i, j) in original image:
    1. Compute rotated location (I’, j’)



Rotate each pixel

Let I be the original image

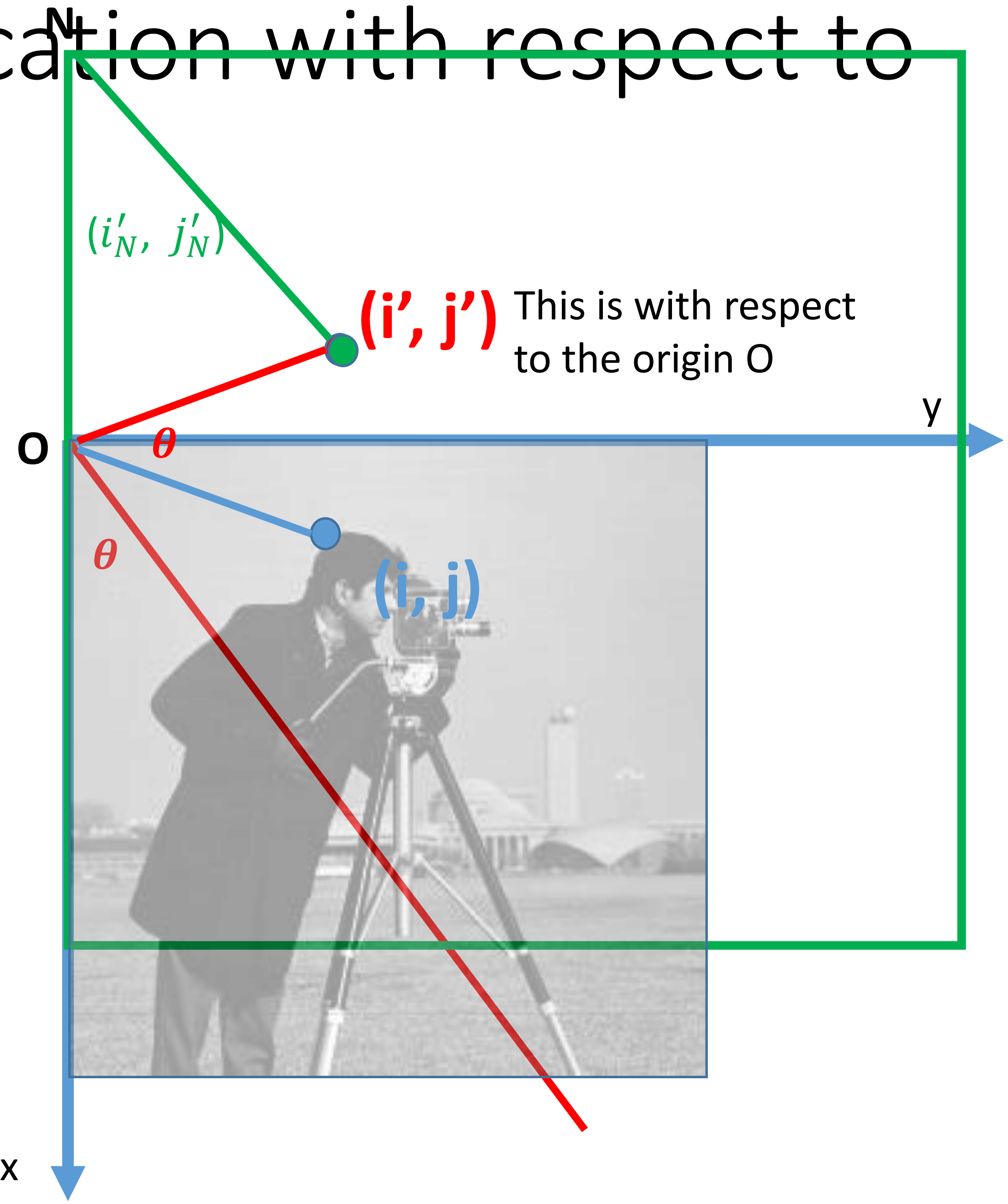
1. Compute Rotation matrix
2. Rotate corners
    1. Get (min_x, min_y) (top left of the image)
    2. Get (max_x, max_y) (bottom right)
        1. Rows = $max_x - min_x$
        2. Cols = max_y - min_y
    3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)
4. For each (i, j) in original image:
    1. Compute rotated location (I’, j’)



Step 3: Compute new location with respect to N

Let I be the original image

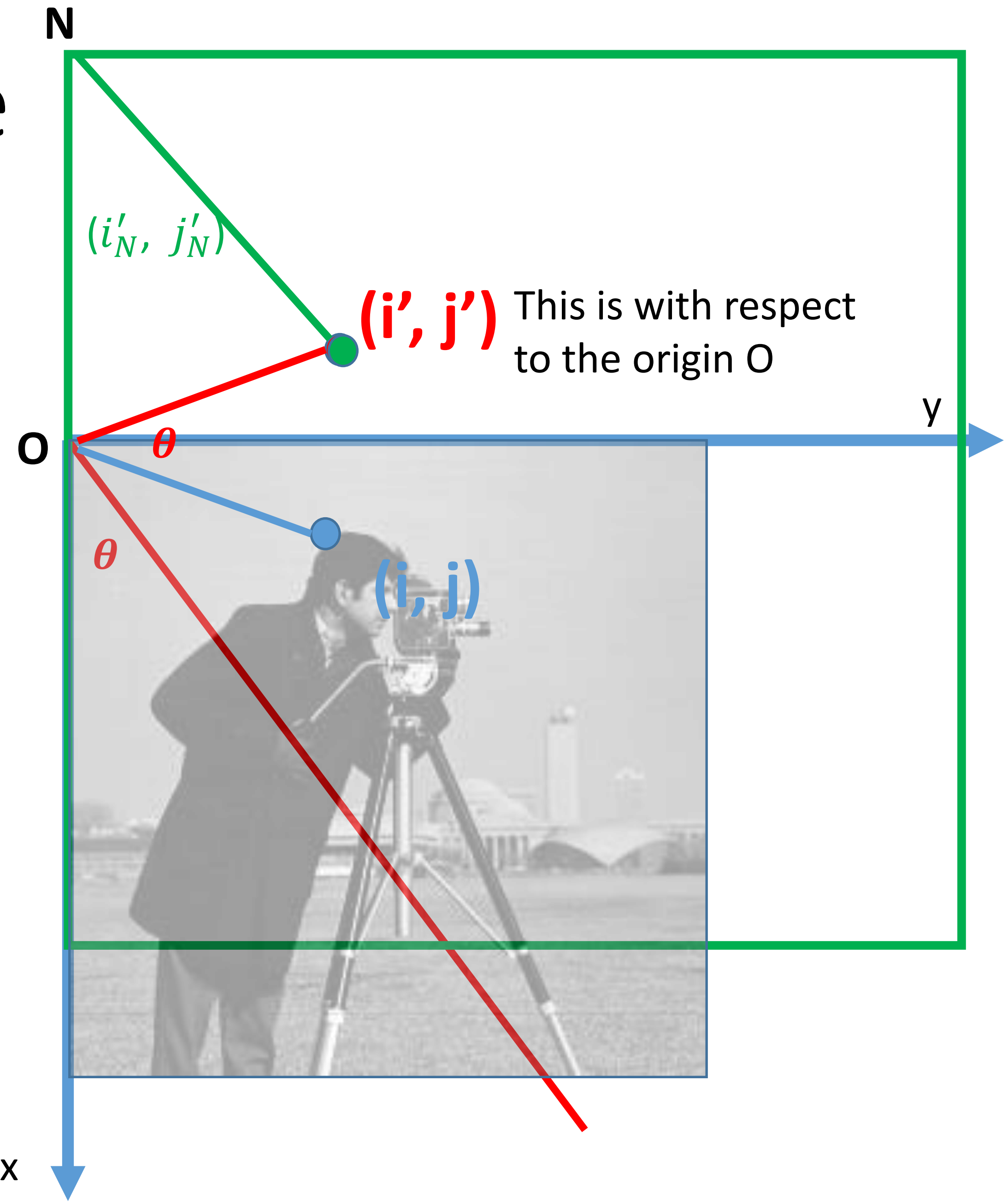
1. Compute Rotation matrix
2. Rotate corners
    1. Get (min_x, min_y) (top left of the image)
    2. Get (max_x, max_y) (bottom right)
        1. Rows = $max_x - min_x$
        2. Cols = max_y - min_y
    3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)
4. For each (i, j) in original image:
    1. Compute rotated location (i’, j’)
    2. $i'_N = i' - min_x, j'_N = j' - min_y$



Step 3: Assign pixel value

Let I be the original image

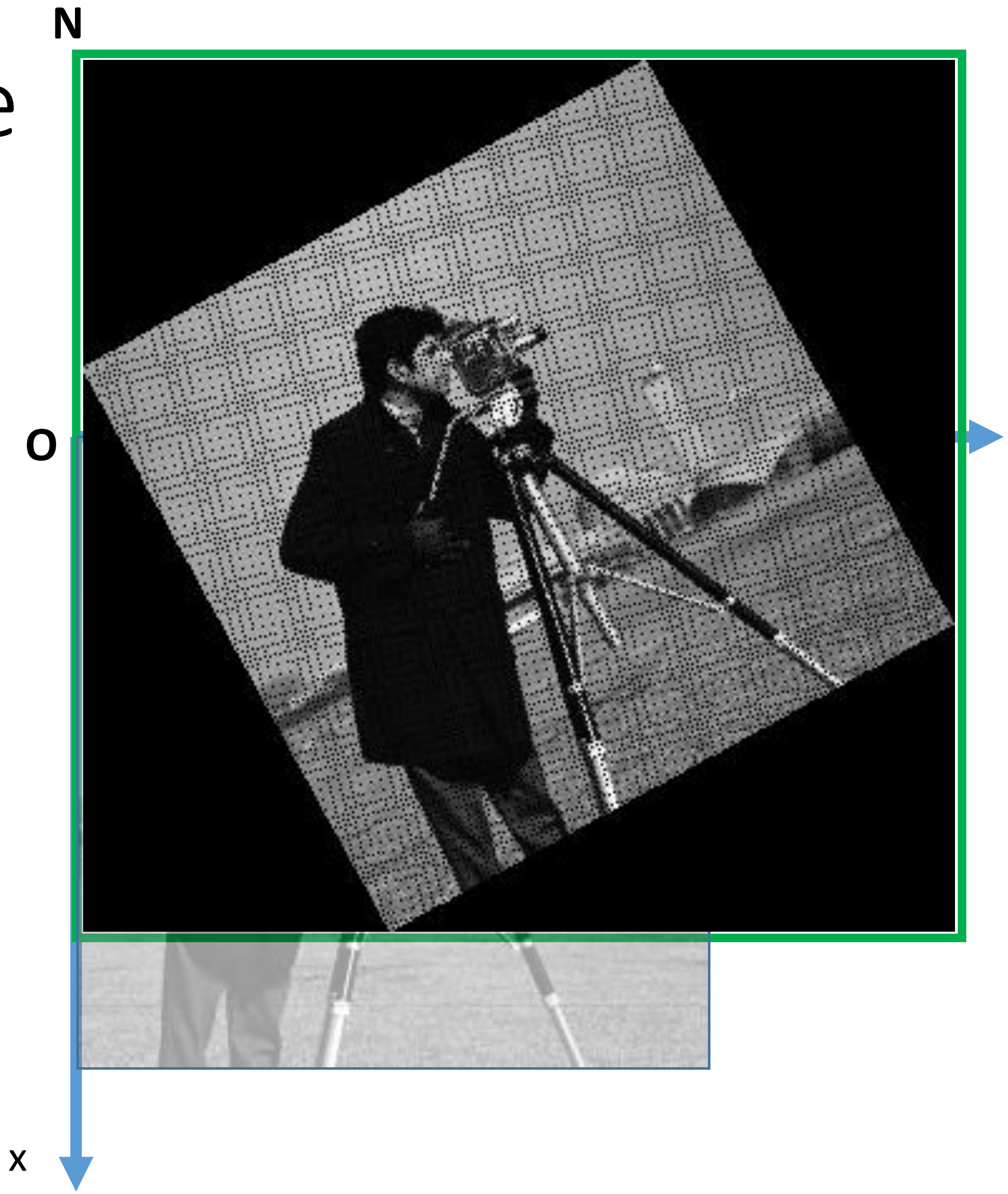
1. Compute Rotation matrix
2. Rotate corners
   1. Get (min_x, min_y) (top left of the image)
   2. Get (max_x, max_y) (bottom right)
      1. Rows = $max_x - min_x$
      2. Cols = max_y - min_y
   3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)
4. For each (i, j) in original image:
   1. Compute rotated location (i', j')
   2. $i'_N = i' - min_x,\ j'_N = j' - min_y$
   3. R($i'_N,\ j'_N$) = I(i, j)



Step 3: Assign pixel value

Let I be the original image

1. Compute Rotation matrix
2. Rotate corners
    1. Get (min_x, min_y) (top left of the image)
    2. Get (max_x, max_y) (bottom right)
        1. Rows = $max_x - min_x$
        2. Cols = max_y - min_y
    3. Size of rotated image = (rows, cols)
3. Create rotated image (R) of size (rows, cols)
4. For each (i, j) in original image:
    1. Compute rotated location (i', j')
    2. $i'_N = i' - min_x,\ j'_N = j' - min_y$
    3. R($i'_N,\ j'_N$) = I(i, j)



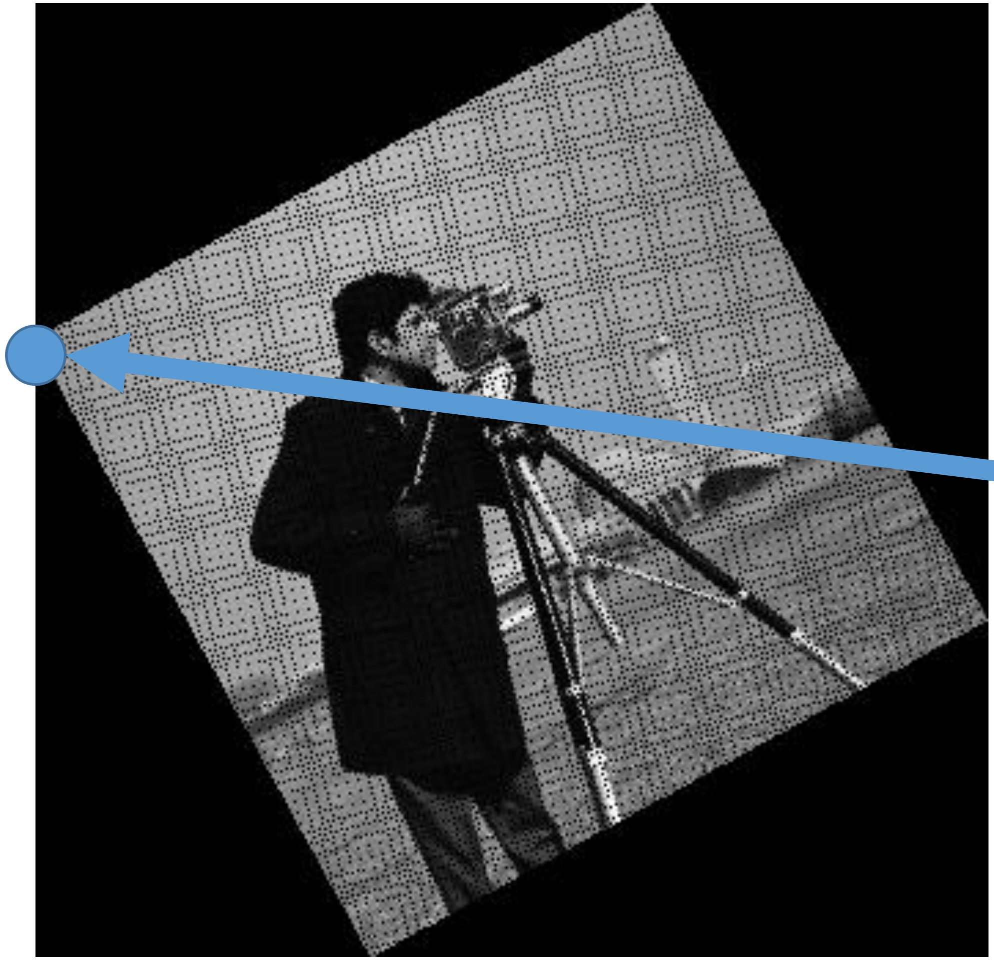
Part 2: Reverse Rotation

Reverse rotate by
$\theta = 0.5$ radians
(~28 deg)

Output

Input:

1. Rotated image
2. Theta: the angle by which the image was rotated
3. Origin ($O = (O_i, O_j)$): the origin of the original image with respect to the origin of rotated image
4. Original shape: Shape of the original image before rotation

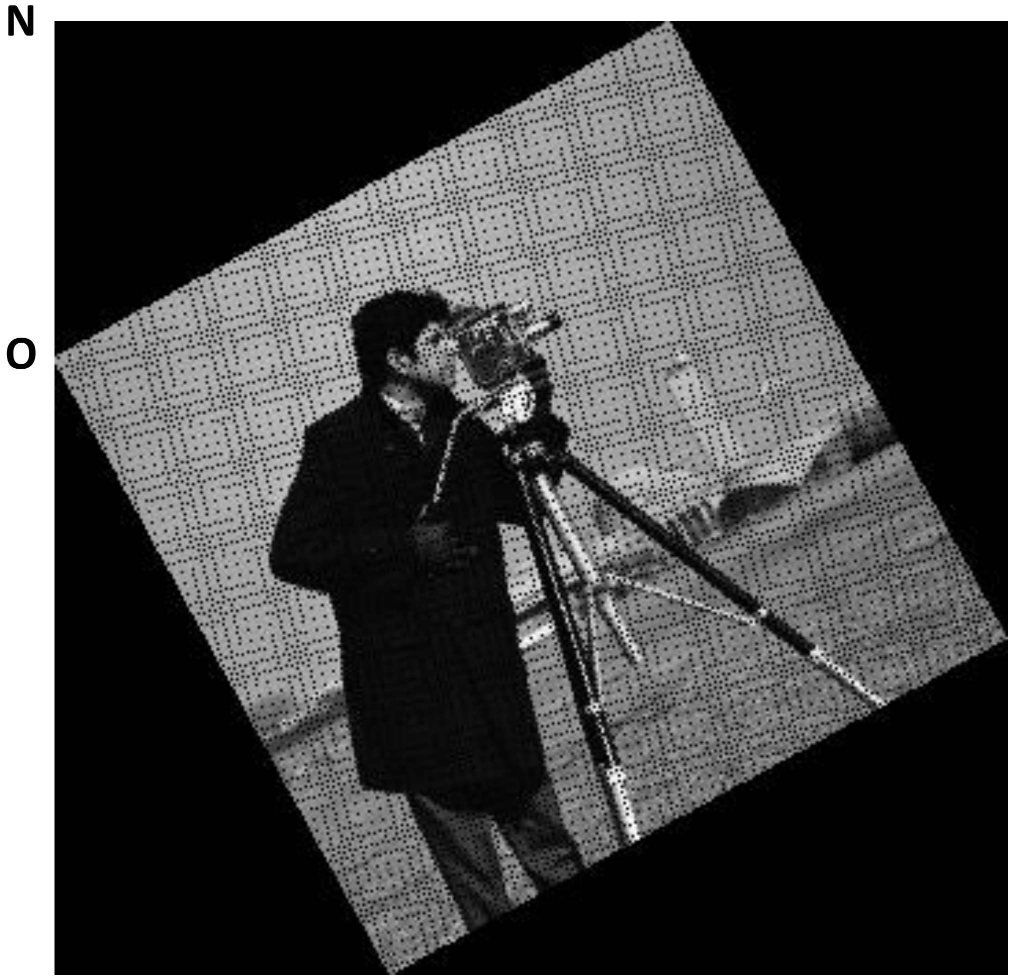


Input:

1. Rotated image
2. Theta: the angle by which the image was rotated
3. Origin ($O = (O_i, O_j)$): the origin of the original image with respect to the origin of rotated image
4. Original shape: Shape of the original image

Let R be rotated image

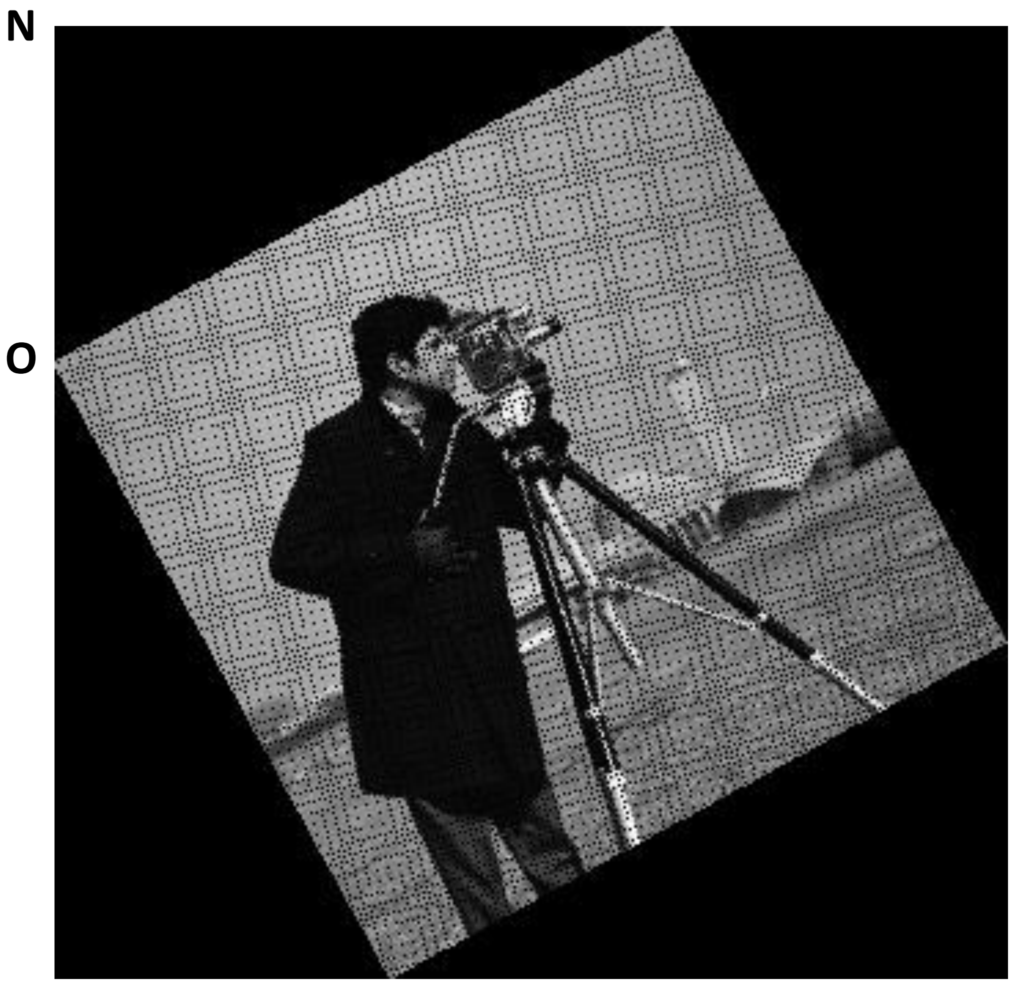
1. Compute inverse rotation matrix



Create empty image with original shape

Let R be rotated image

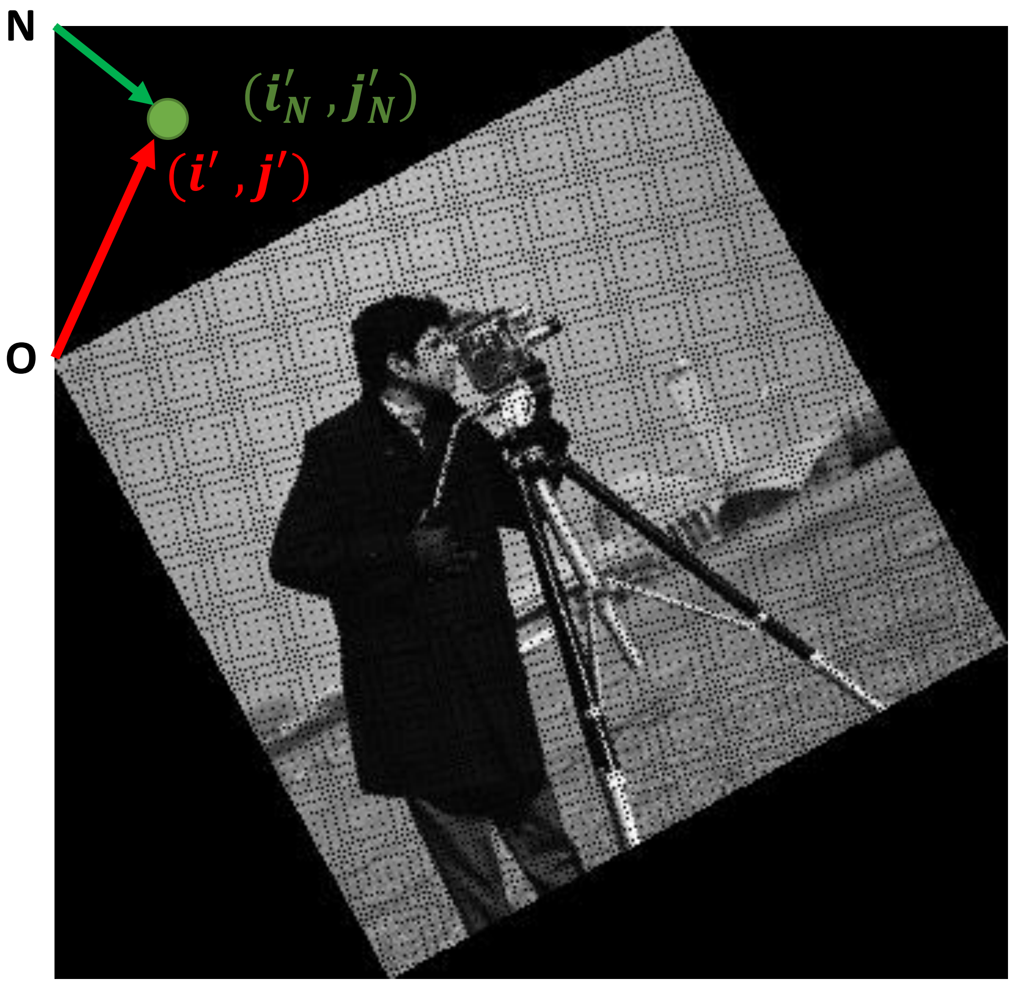
1. Compute inverse rotation matrix
2. Create image (I) of shape (original shape)



Iterate through each pixel and compute original location

Let R be rotated image

1. Compute inverse rotation matrix
2. Create image (I) of shape (original shape)
3. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O

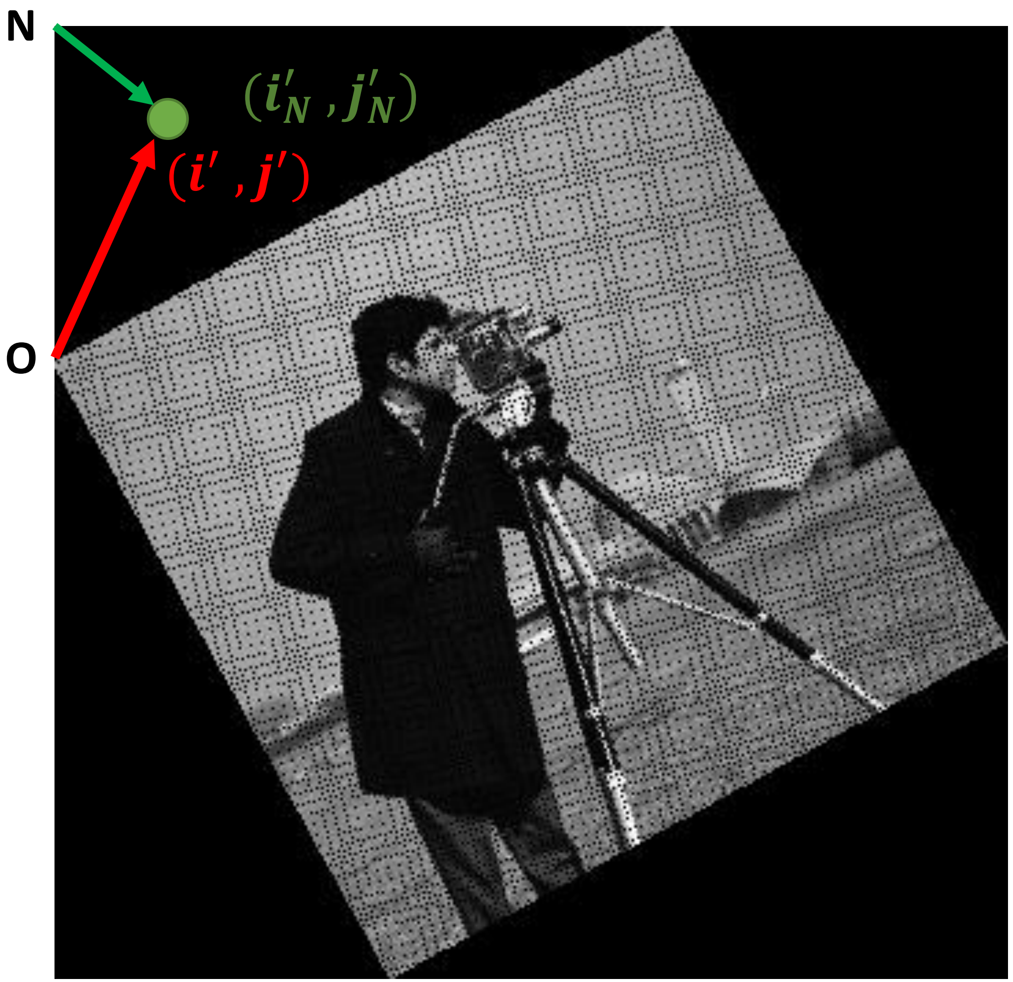


Iterate through each pixel and compute original location

Let R be rotated image

1. Compute inverse rotation matrix
2. Create image (I) of shape (original shape)
3. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O

    $i' = i'_N - O_i, j' = j'_N - O_j$



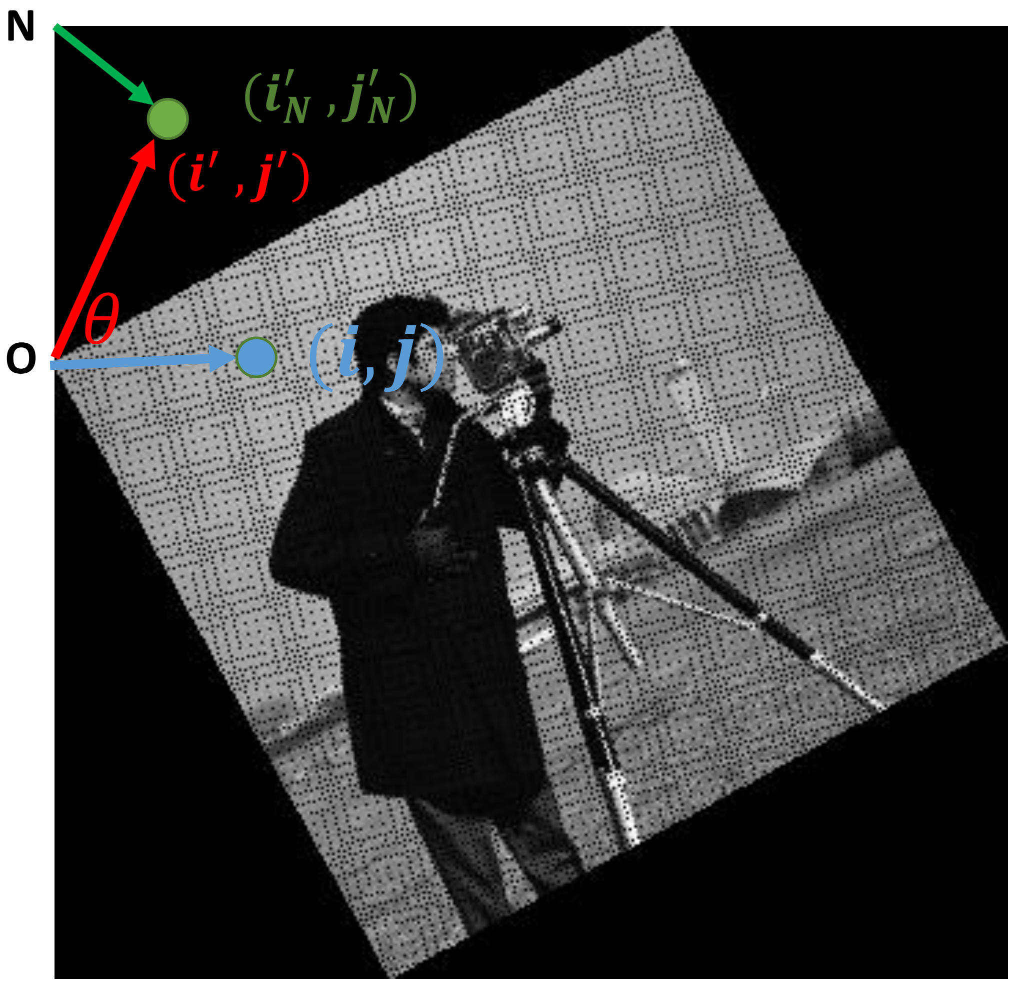
Iterate through each pixel and compute original location

Let R be rotated image

1. Compute inverse rotation matrix
2. Create image (I) of shape (original shape)
3. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O

       $i' = i'_N - O_i, j' = j'_N - O_j$

    2. Compute inverse rotation on (i', j') to get (i, j)

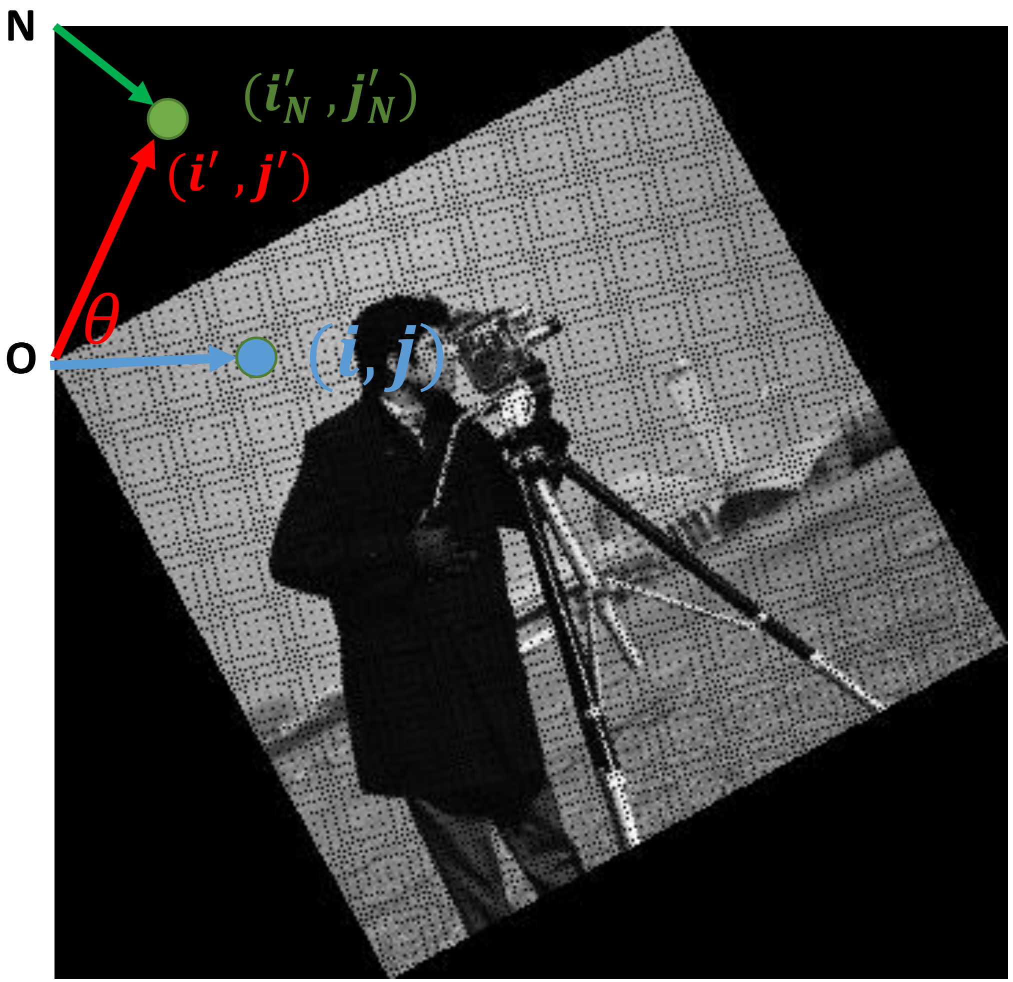


Iterate through each pixel and compute original location

Let R be rotated image

1. Compute inverse rotation matrix
2. Create image (I) of shape (original shape)
3. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O
       $i' = i'_N - O_i, j' = j'_N - O_j$
    2. Compute inverse rotation on (i', j') to get (i, j)

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix}$$



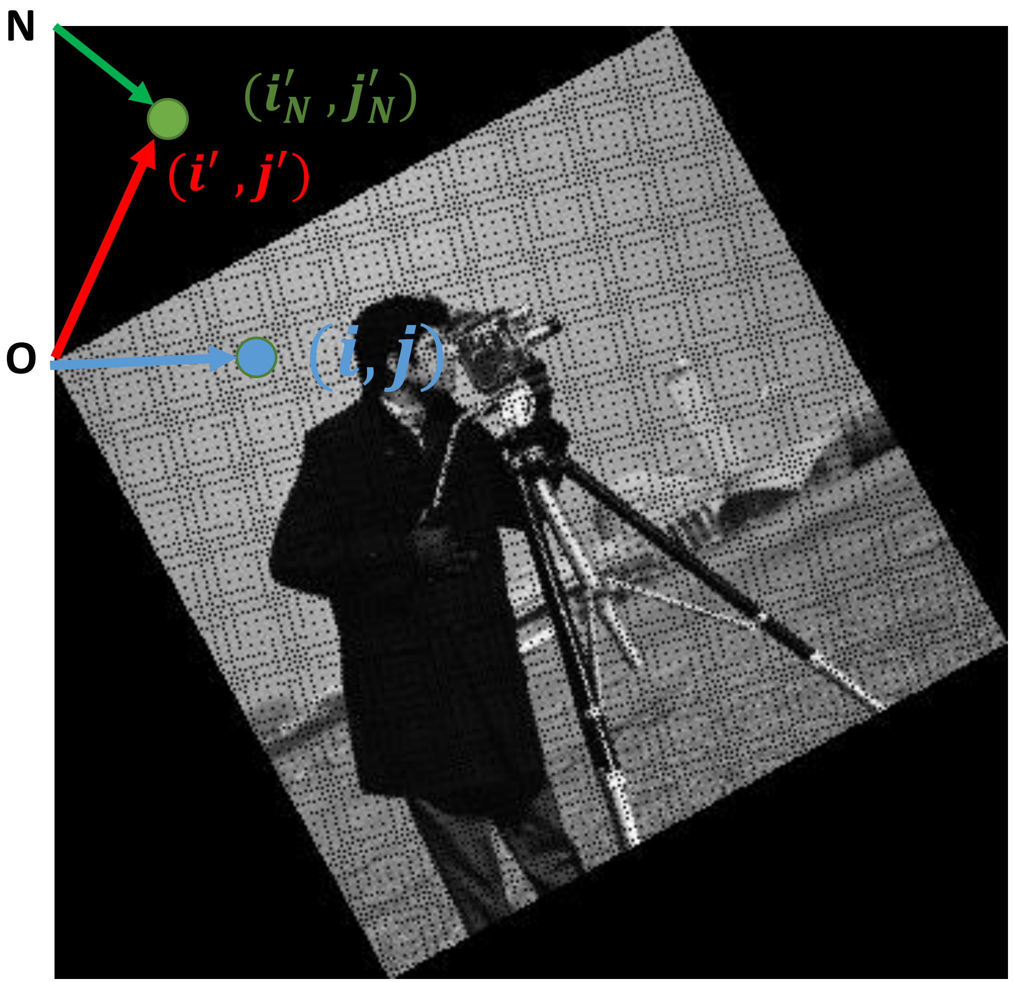
Assign values

Let R be rotated image

1. Compute inverse rotation matrix
2. Create image (I) of shape (original shape)
3. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O

       $i' = i'_N - O_i, j' = j'_N - O_j$

    2. Compute inverse rotation on (i', j') to get (i, j)
    3. $I(i,j) = R(i'_N, j'_N)$



Output

Forward rotation

- Missing Pixel Values
- Perform interpolation to fill in missing values.

Part 3: Rotation with Interpolation

Part 3: Rotation with interpolation

Input

Rotation with interpolation

$\theta = 0.5$ radians

(~28 deg)

Output

Part 3: Forward Rotation

Input

Rotation with interpolation

$\theta = 0.5$ radians

(~28 deg)

Output

Input:

1. Image
2. Theta
3. Type of interpolation

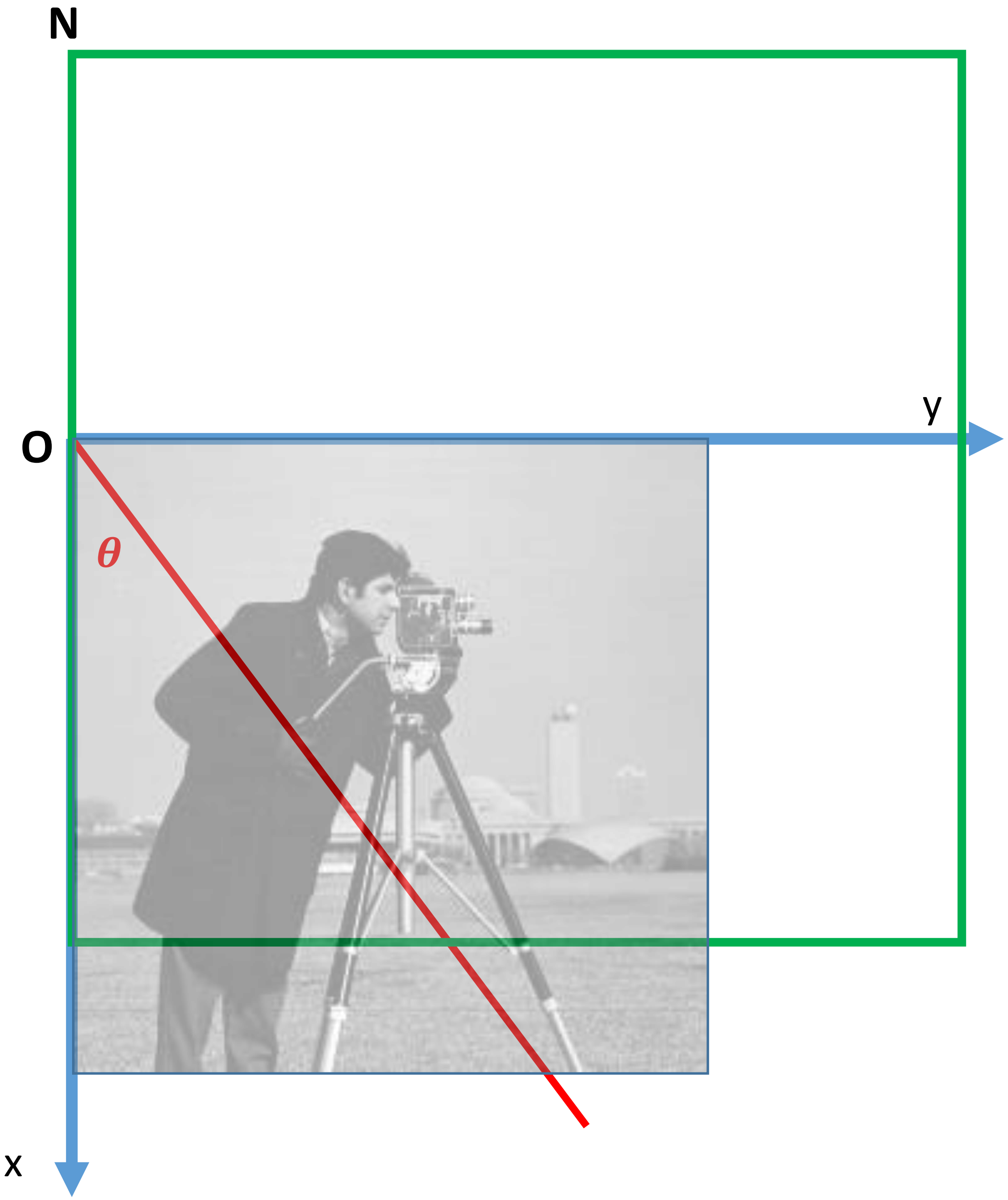
Part 3: Forward Rotation

- Solution: Combination of Part 1 and 2

Idea

Let I be the original image

Assuming R is the image obtained after rotation.

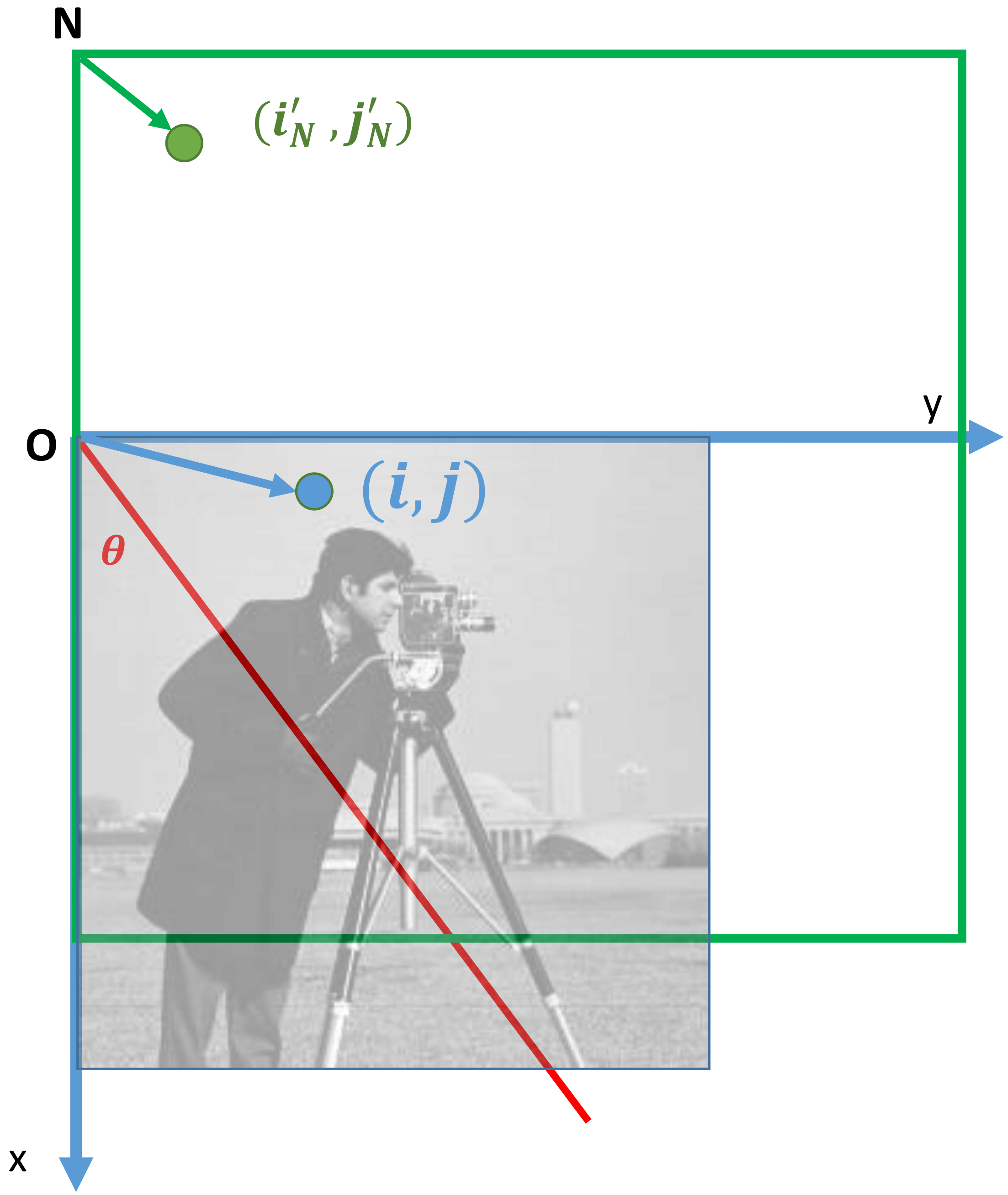


Idea

Let I be the original image

Assuming R is the image obtained after rotation.

1. For (i'_N, j'_N) in rotated image R.
    1. Determine its location (i, j) before rotation in the original image

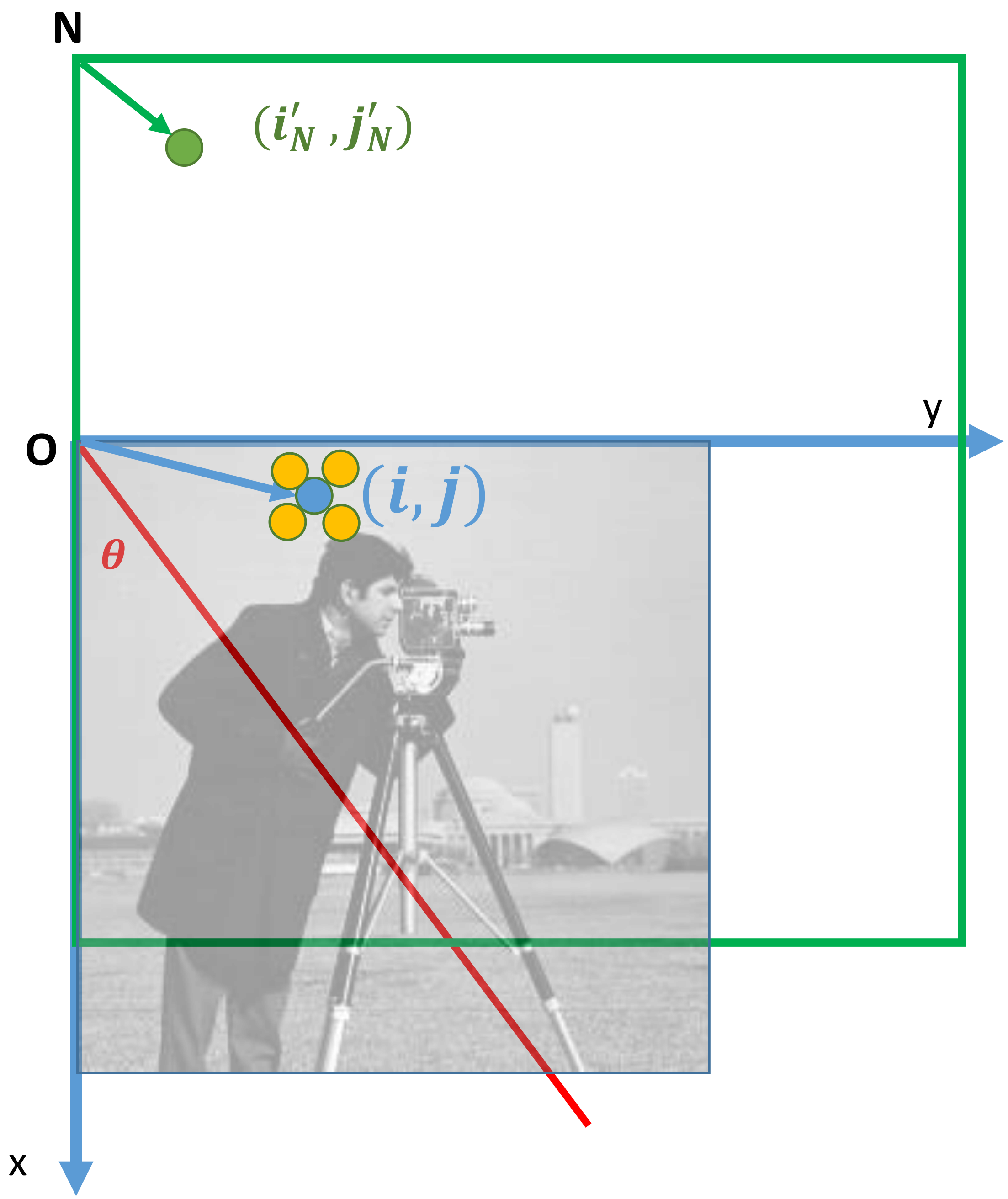


Idea

Let I be the original image

Assuming R is the image obtained after rotation.

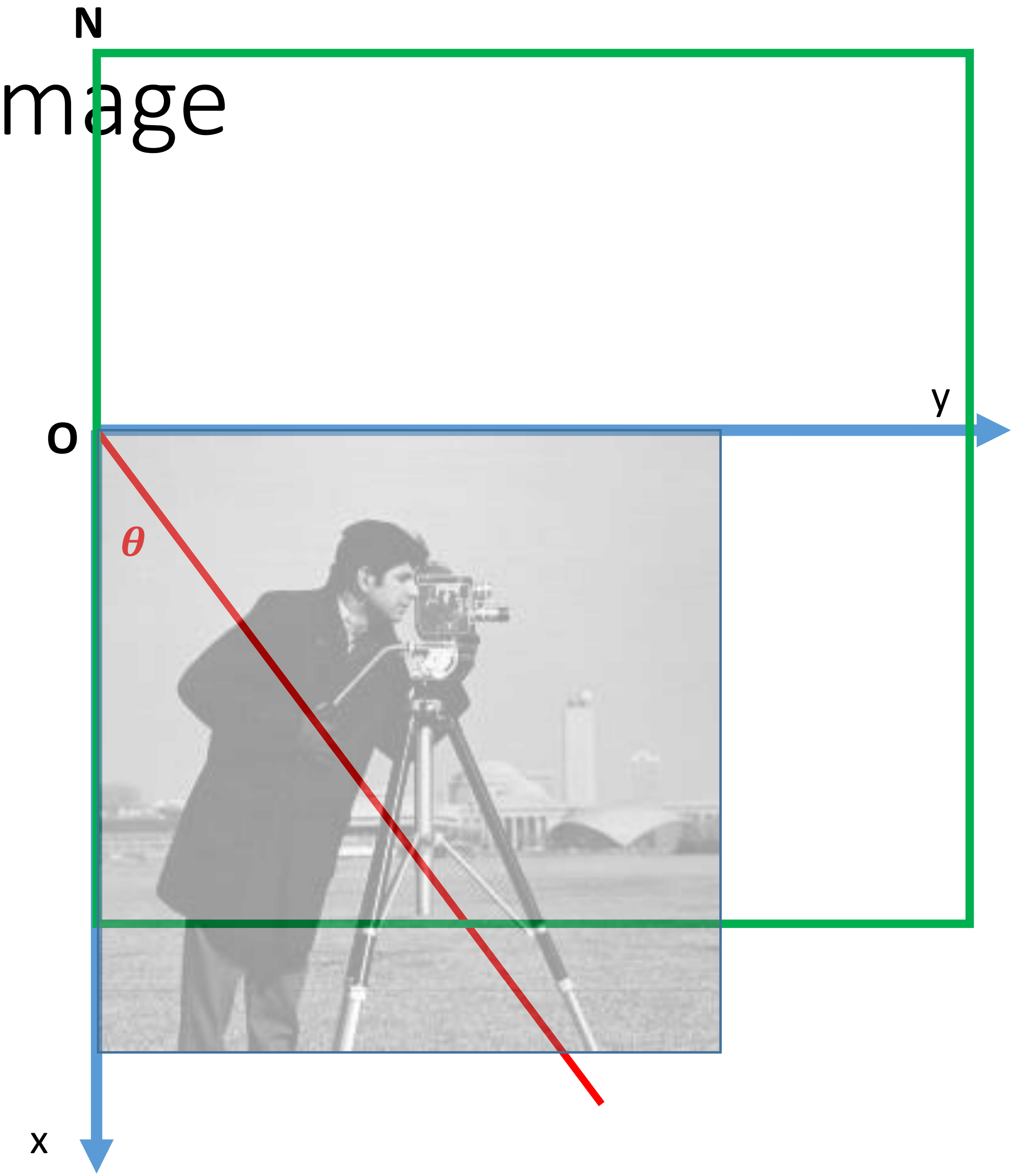
1. For (i'_N, j'_N) in rotated image R.
    1. Determine its location (i, j) before rotation in the original image
    2. Use neighbors of (i, j) to perform either nearest neighbour or bilinear interpolation.
    3. $\mathrm{R}(i'_N, j'_N) = interploted(i, j)$



Compute size of rotate image

Let I be the original image

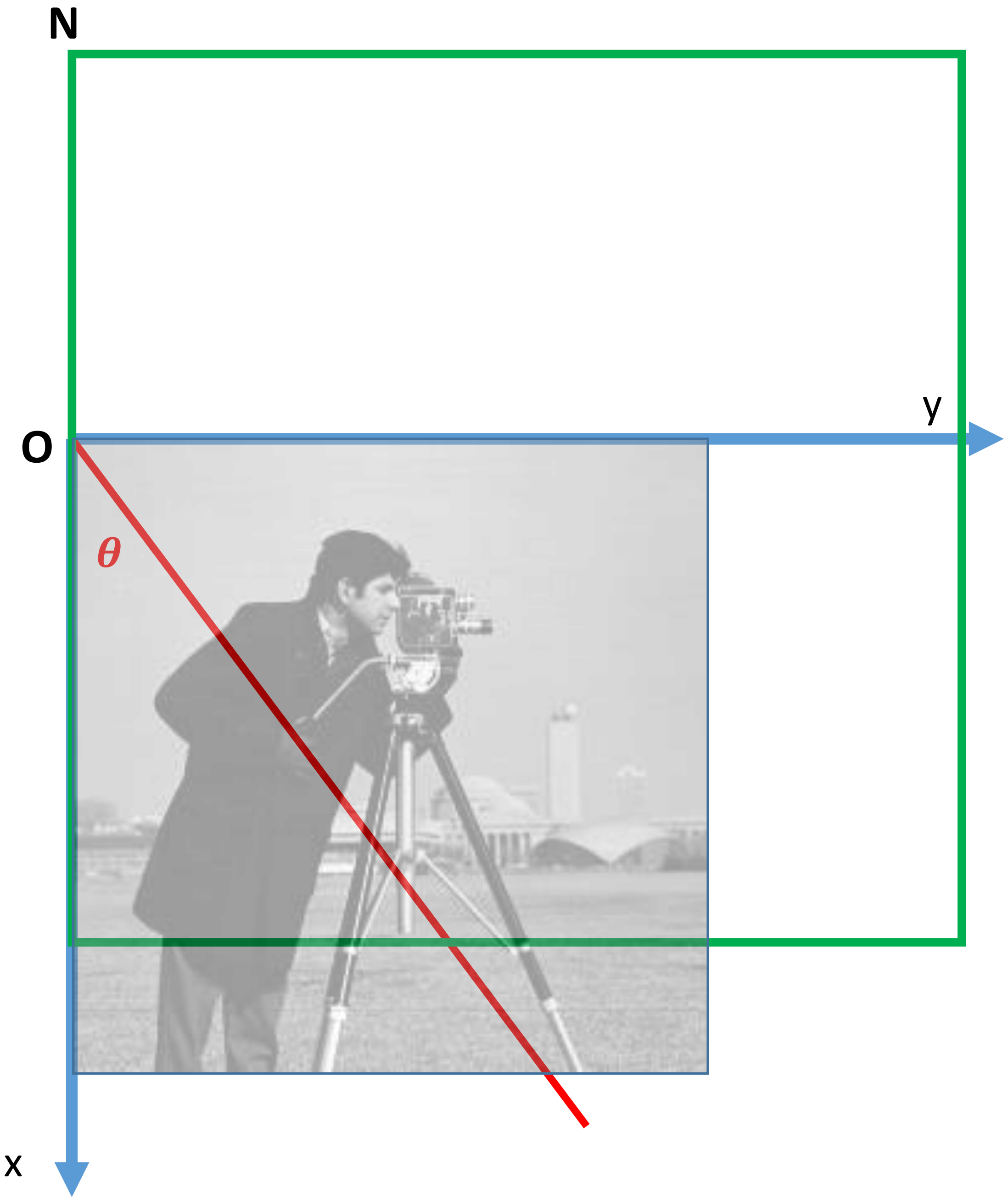
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image



Create Empty Image

Let I be the original image

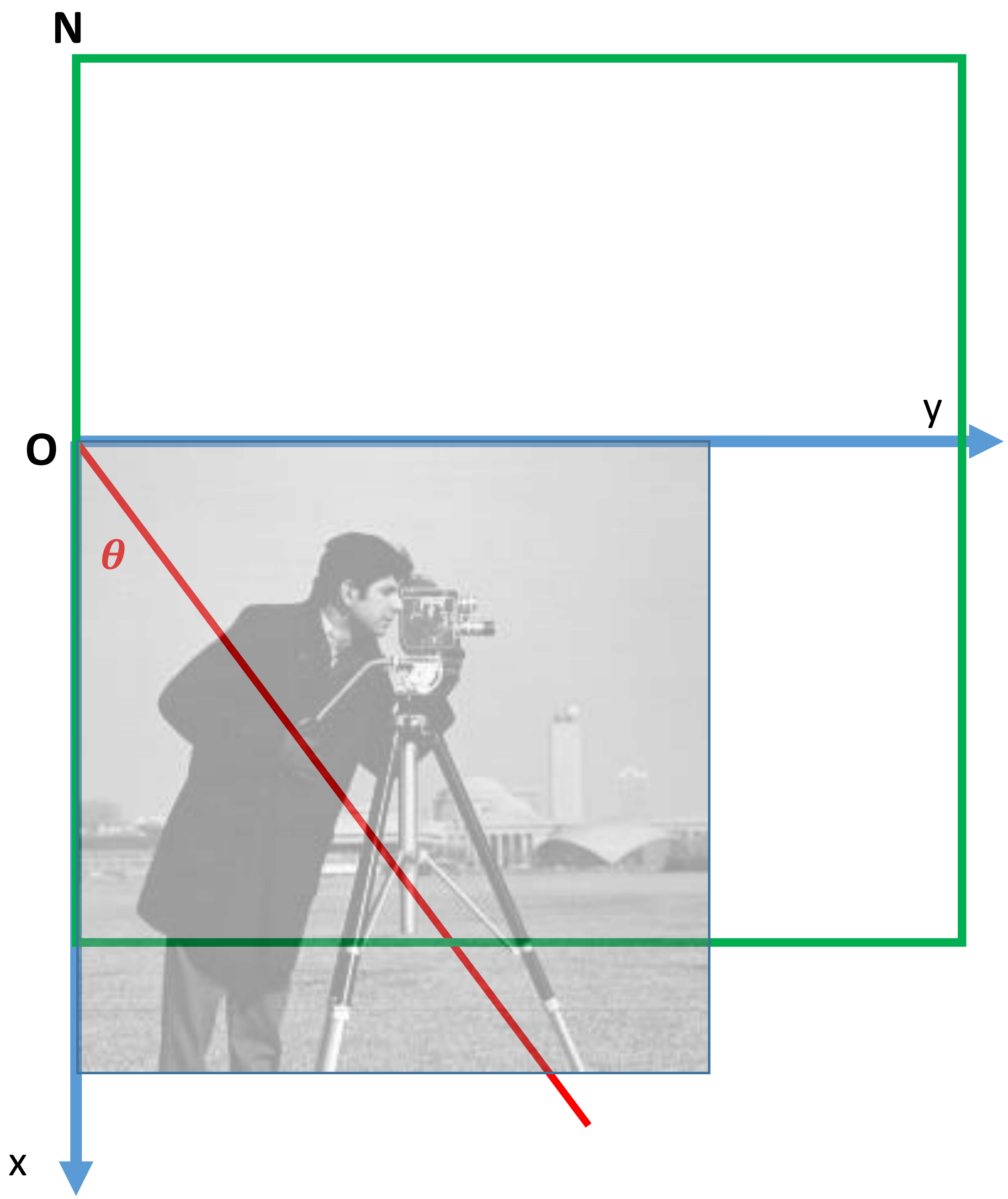
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)



Calculate O

Let I be the original image

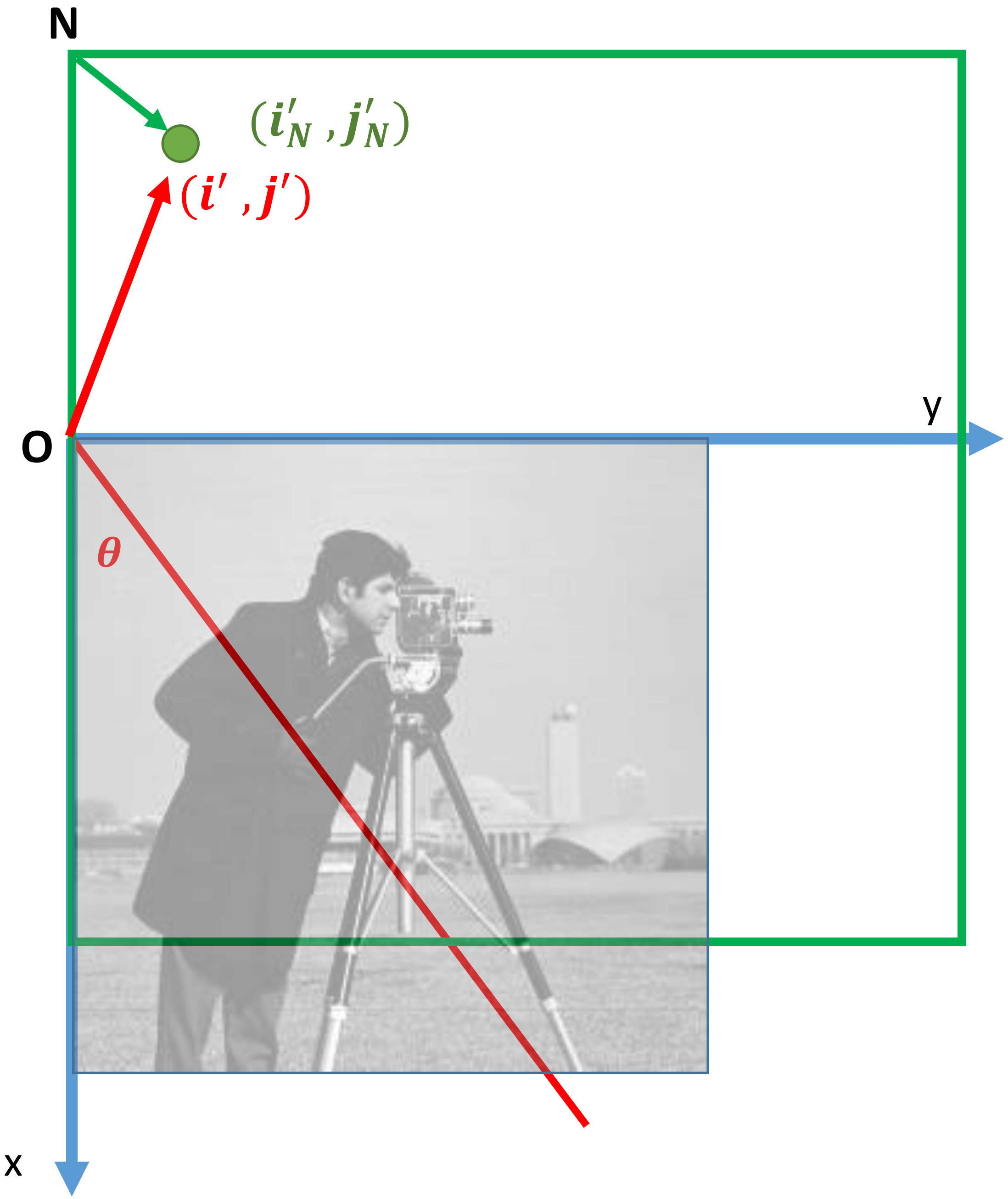
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)
5. Calculate location O, with respect to N ($O = -\min_x, -\min_y$) (Computed from 4 corners)



Iterate and interpolate

Let I be the original image

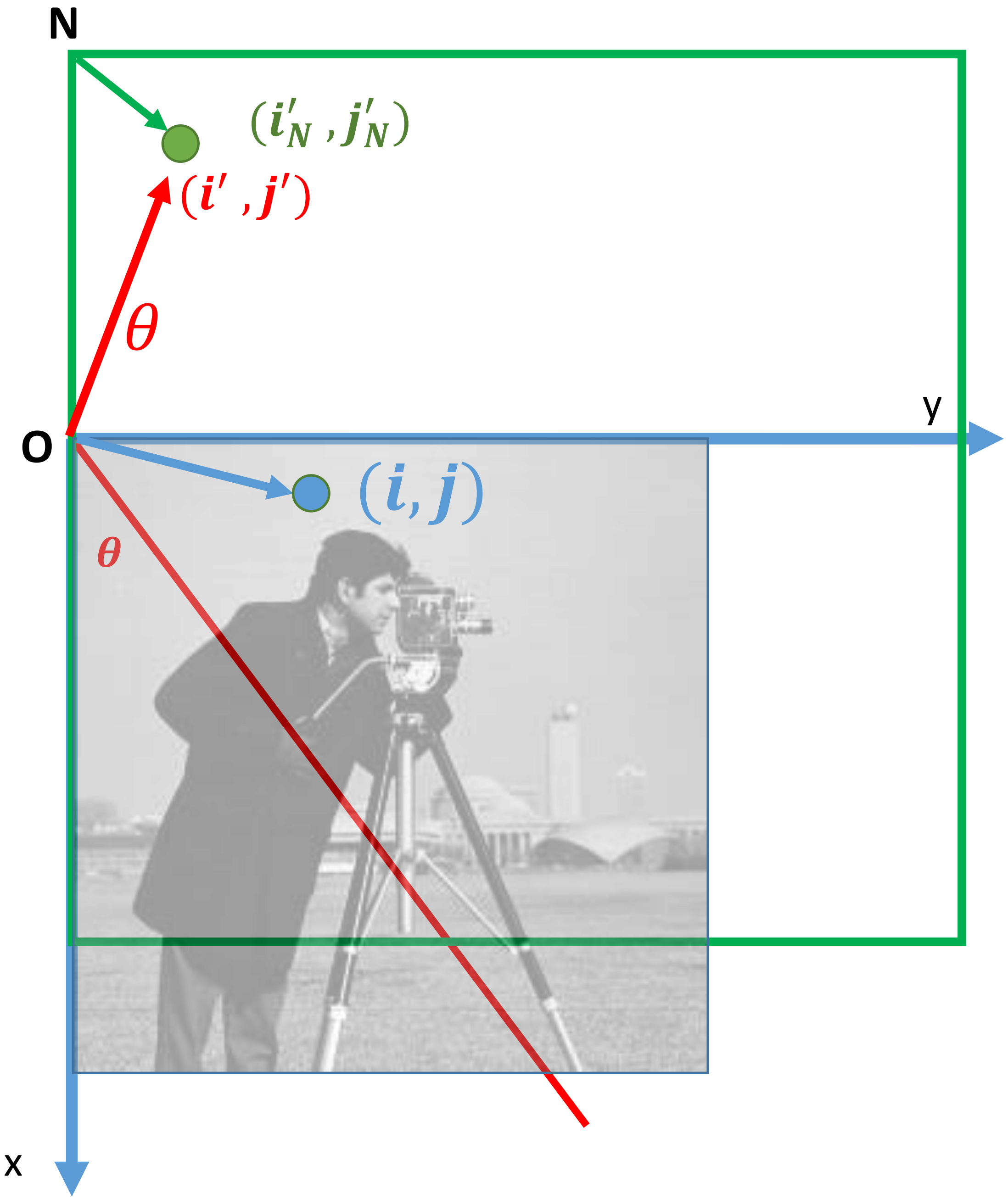
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)
5. Calculate location O, with respect to N ($O = -\min_x, -\min_y$)(Computed from 4 corners)
6. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O
       $i' = i'_N - O_i, j' = j'_N - O_j$
    2.



Iterate and interpolate

Let I be the original image

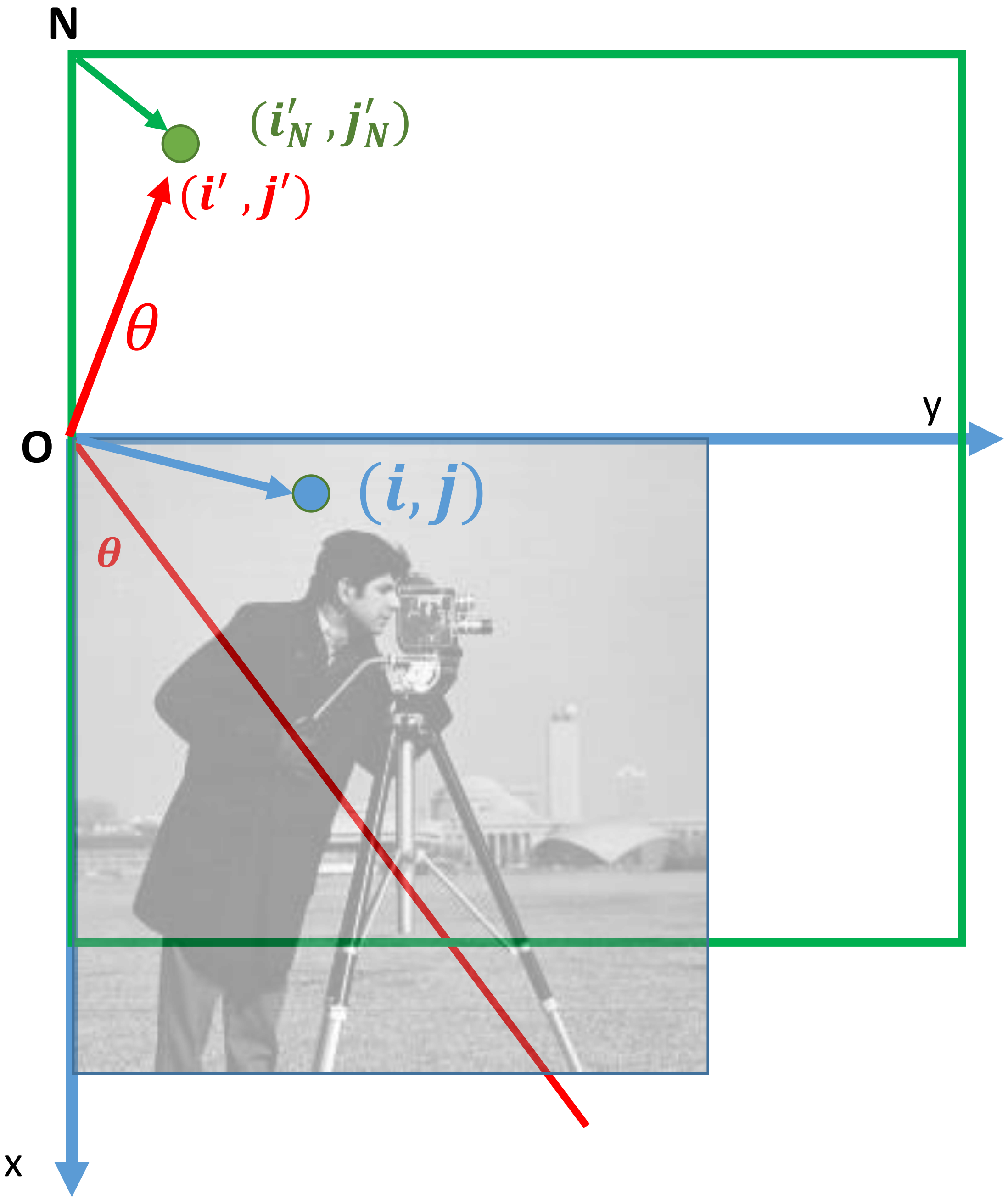
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)
5. Calculate location O, with respect to N ($O = -\min_x, -\min_y$)(Computed from 4 corners)
6. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O
       $i' = i'_N - O_i, j' = j'_N - O_j$
    2. Compute inverse rotation on (i', j') to get (i, j)



Nearest interpolate

Let I be the original image

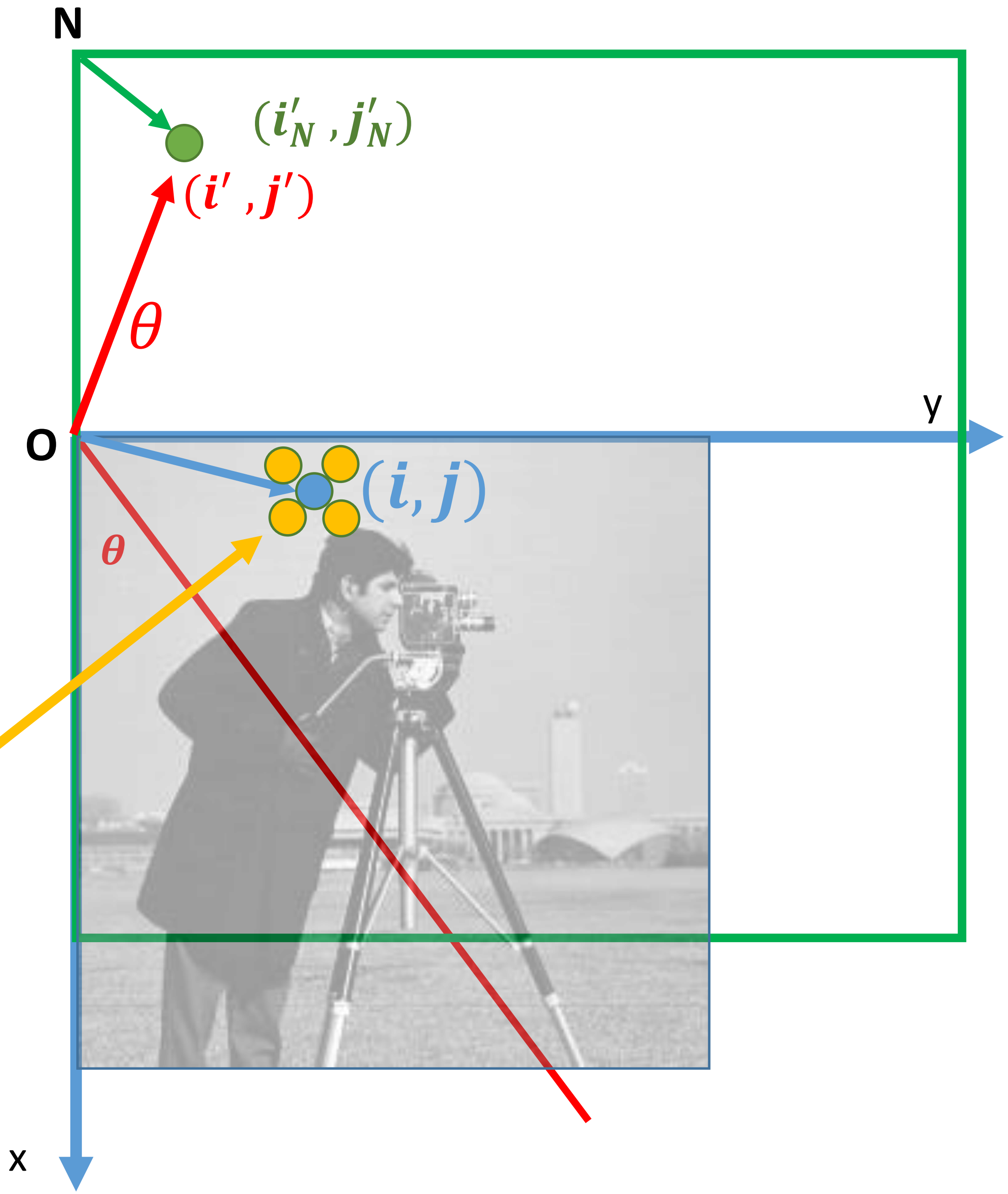
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)
5. Calculate location O, with respect to N ($O = -\min_x, -\min_y$)(Computed from 4 corners)
6. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O
       $i' = i'_N - O_i, j' = j'_N - O_j$
    2. Compute inverse rotation on (i', j') to get (i, j)
    3. If using nearest neighbour interpolation
        1. nearest neighbour
           (i_{nn}, j_{nn})= $(round(i),\ round(j))$
        2. $\mathrm{R}(i'_N, j'_N) = I(i_{nn}, j_{nn})$



Bilinear interpolate

Let I be the original image

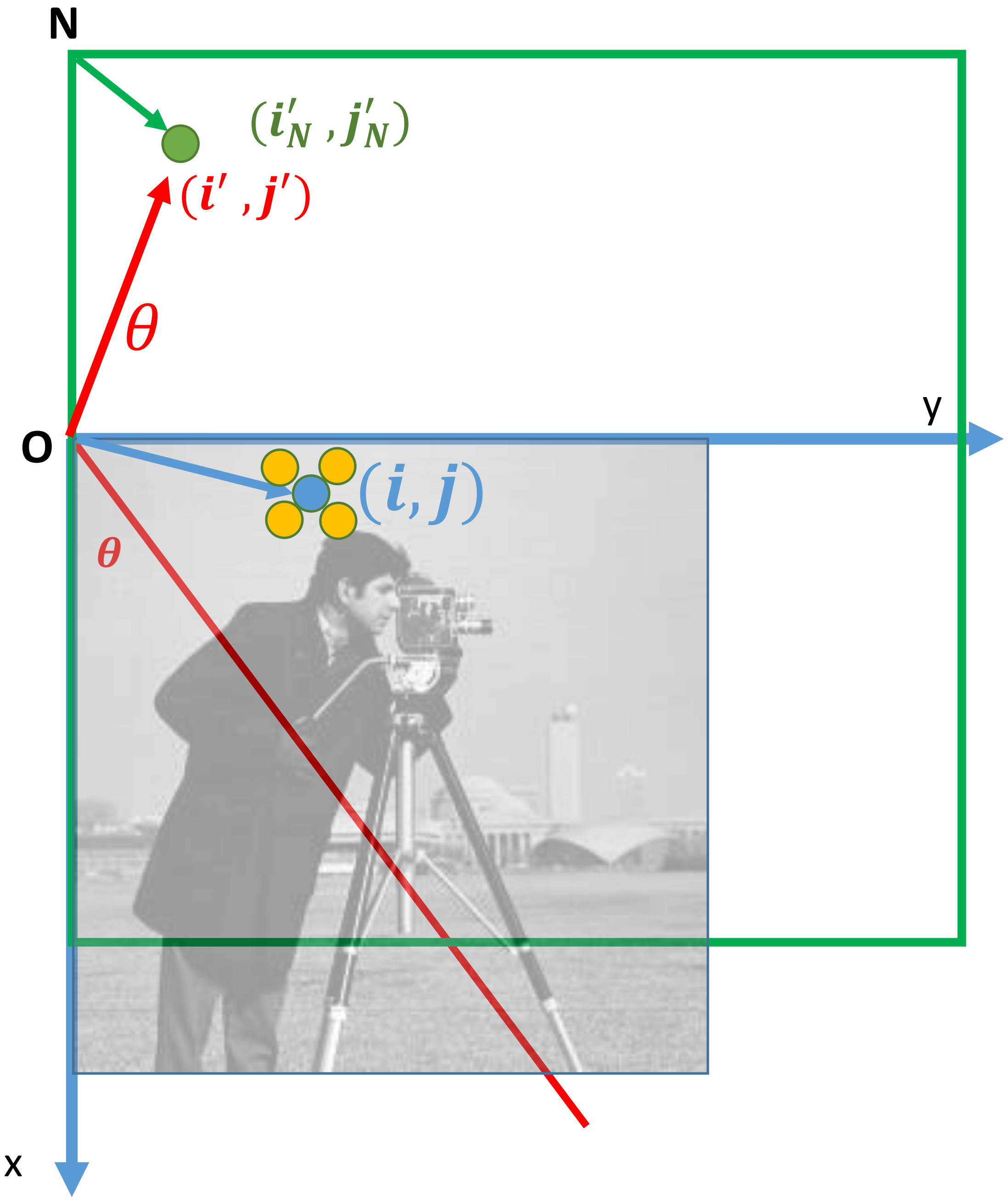
1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)
5. Calculate location O, with respect to N ($O = -\min_x, -\min_y$)(Computed from 4 corners)
6. For (i'_N, j'_N) in rotated image
    1. Calculate location with respect to O
       $i' = i'_N - O_i, j' = j'_N - O_j$
    2. Compute inverse rotation on (i', j') to get (i, j)
    3. If using bilinear interpolation
        1. Find **four** nearest neighbors to (i, j)



Bilinear interpolate

Let I be the original image

1. Compute Rotation matrix
2. Compute Inverse Rotation matrix
3. Compute size of the rotated image
4. Create rotated image (R) of size (rows, cols)
5. Calculate location O, with respect to N ($O = -\min_x, -\min_y$)(Computed from 4 corners)
6. For (i'_N, j'_N) in rotated image
   1. Calculate location with respect to O
      $i' = i'_N - O_i, j' = j'_N - O_j$
   2. Compute inverse rotation on (i', j') to get (i, j)
   3. If using bilinear interpolation
      1. Find **four** nearest neighbors to (i, j)
      2. perform bi-linear interpolated value (b)
      2. $\mathrm{R}(i'_N, j'_N) = b$



Without Interpolation

Nearest neighbor Interpolation

Assignment - 1

1. Forward Rotate (20 Pts.)
2. Reverse Rotate (20 Pts)
3. Rotate with interpolation (35 Pts)

Total: 75 Pts.

Due Date: Feb 24th

Submission Instructions

- Must use the **starter code** available in **Github**
- Submission allowed only through **Github**
- You will receive an email with invitation to join **Github** classroom
- Start by reading the **readme.md** file.
- Instructions are available here
- Github will **automatically** save the **last commit as a submission** before the deadline