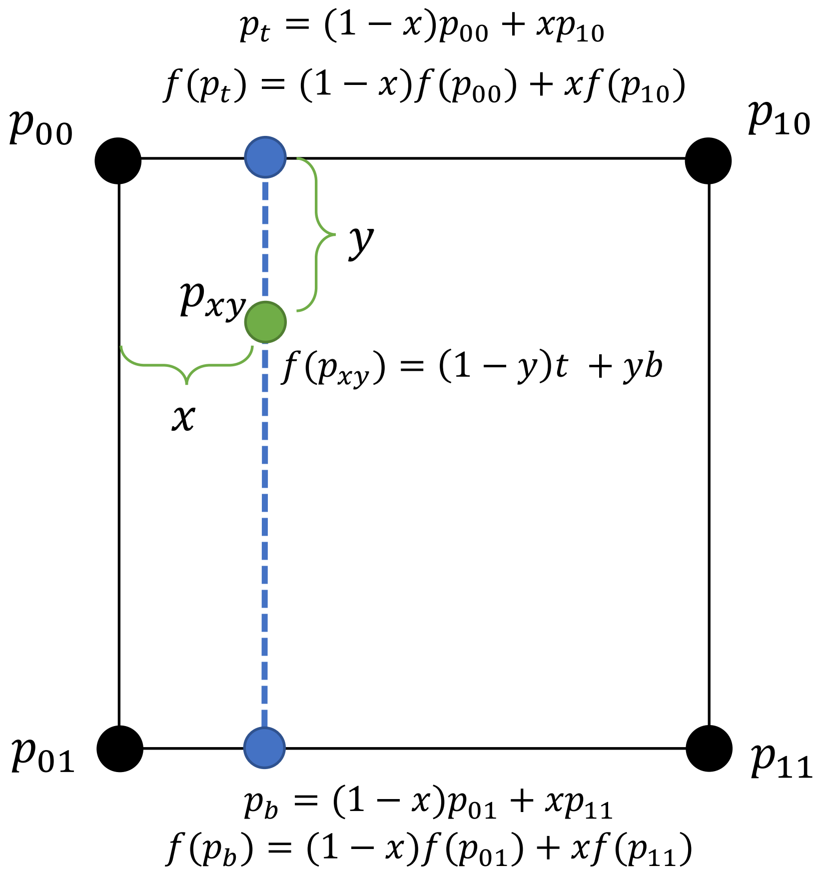
Lecture on Convolution and Warping

Madhav Aggarwal

Bilinear Upsampling

Bilinear Interpolation

- Consider the normalized case, where we are interpolating values at the corners of a unit square
- Linearly interpolate the bounding values along one dimension, then linearly interpolate those along the other
- We can write this out as a simple linear combination of the values at each of our four corners



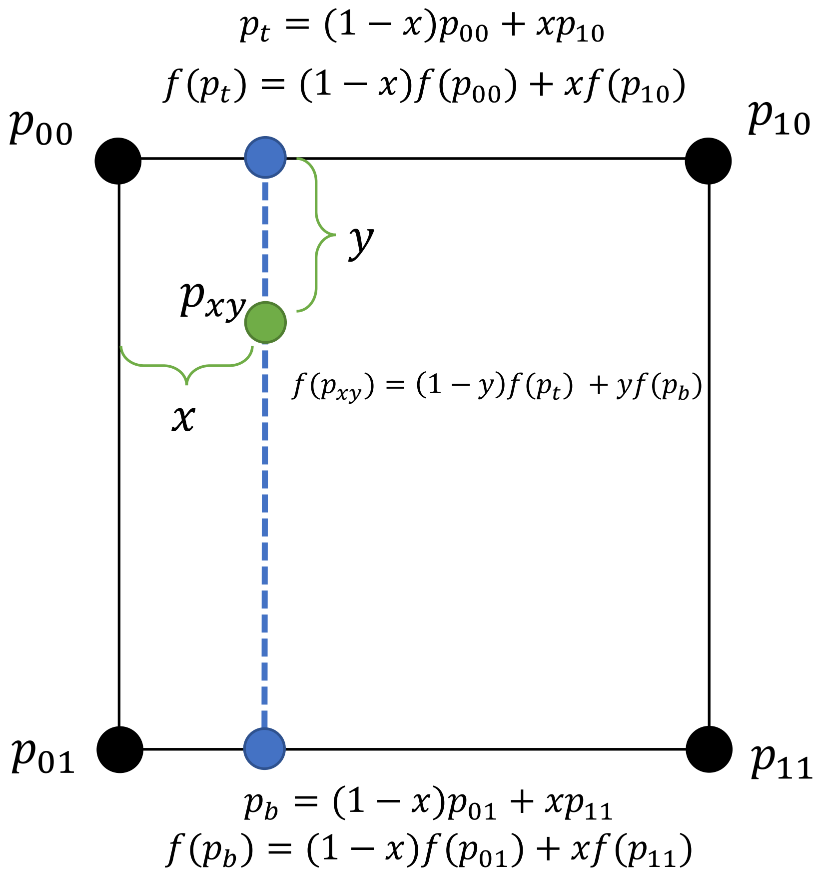
Question:

Derive the coefficients of the linear combination defined by bilinear interpolation

$$[a \quad b \quad c \quad d]^T \begin{bmatrix} f(p_{00}) \\ f(p_{10}) \\ f(p_{01}) \\ f(p_{11}) \end{bmatrix}$$

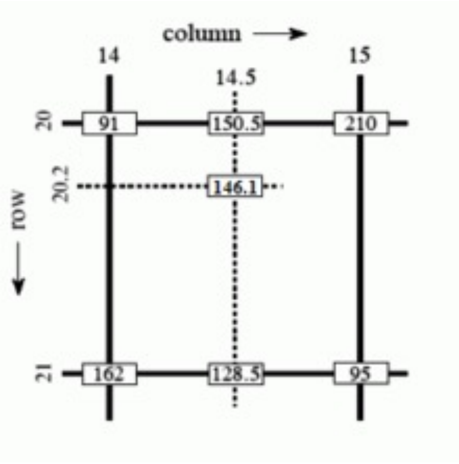
Bilinear Interpolation

- Consider the normalized case, where we are interpolating values at the corners of a unit square
- Linearly interpolate the bounding values along one dimension, then linearly interpolate those along the other
- We can write this out as a simple linear combination of the values at each of our four corners



$$[a \quad b \quad c \quad d]^T \begin{bmatrix} f(p_{00}) \\ f(p_{10}) \\ f(p_{01}) \\ f(p_{11}) \end{bmatrix} \Rightarrow \begin{bmatrix} (1-x)(1-y) \\ x(1-y) \\ (1-x)y \\ xy \end{bmatrix}$$

Try applying what you just learnt!



Find the values at the column 14.5 by first linearly interpolating between values at 14 and 15 on each row 20 and 21

$$I_{20,14.5} = \frac{15-14.5}{15-14} \cdot 91 + \frac{14.5-14}{15-14} \cdot 210 = 150.5,$$
$$I_{21,14.5} = \frac{15-14.5}{15-14} \cdot 162 + \frac{14.5-14}{15-14} \cdot 95 = 128.5,$$

Next you interpolate these two values:

$$I_{20.2,14.5} = \frac{21-20.2}{21-20} \cdot 150.5 + \frac{20.2-20}{21-20} \cdot 128.5 = 146.1.$$

2D Convolution

Discrete filtering in 2D

- Same equation, one more index

$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

  - now the filter is a rectangle you slide around over a grid of numbers
- Commonly applied to images
  - blurring (using box, using gaussian, …)
  - sharpening (impulse minus blur)
  - feature detection (edges, corners, …)
  - in convolutional neural networks (CNNs)
- Usefulness of associativity
  - often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - this is equivalent to applying one filter: a * $(b_1 * b_2 * b_3)$

And in pseudocode…

function convolve2d(filter2d a, filter2d b, int i, int j)

$s = 0$

$r = a.\text{radius}$

for $i' = -r$ to r **do**

    for $j' = -r$ to r **do**

        $s = s + a[i'][j']b[i - i'][j - j']$

return s

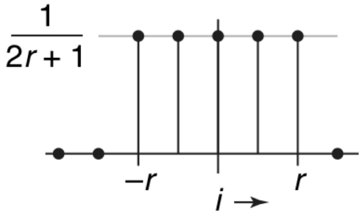
Building 2D filters

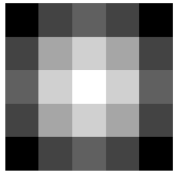
- Almost always, we build 2D filters from 1D filters like this:
  - $a_2[i,j] = a_1[i]a_1[j]$
- This is called a “separable” filter

1D Formula | 1D Shape | 2D Shape

Box:

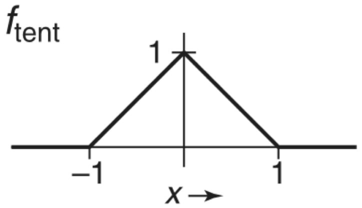
$$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \le r, \\ 0 & \text{otherwise.} \end{cases}$$

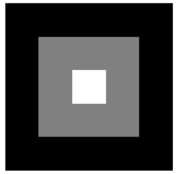




Tent:

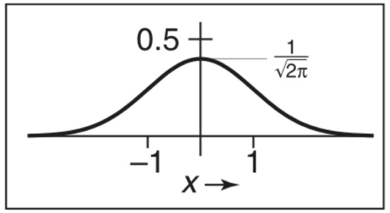
$$f_{\text{tent}}(x) = \begin{cases} 1-|x| & |x| < 1, \\ 0 & \text{otherwise;} \end{cases}$$





Gaussian:

$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$



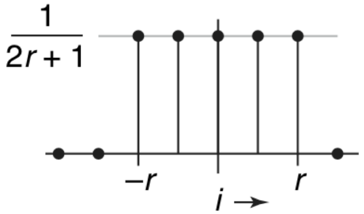
Building 2D filters

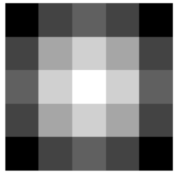
- Almost always, we build 2D filters from 1D filters like this:
  - $a_2[i,j] = a_1[i]a_1[j]$
- This is called a "separable" filter

<u>1D Formula</u> <u>1D Shape</u> <u>2D Shape</u>

Box:

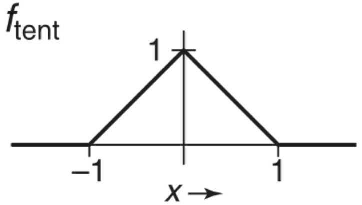
$$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \le r, \\ 0 & \text{otherwise.} \end{cases}$$

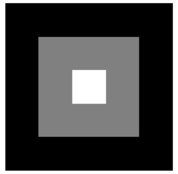




Tent:

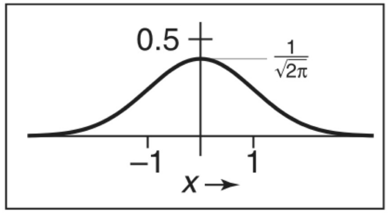
$$f_{\text{tent}}(x) = \begin{cases} 1 - |x| & |x| < 1, \\ 0 & \text{otherwise;} \end{cases}$$





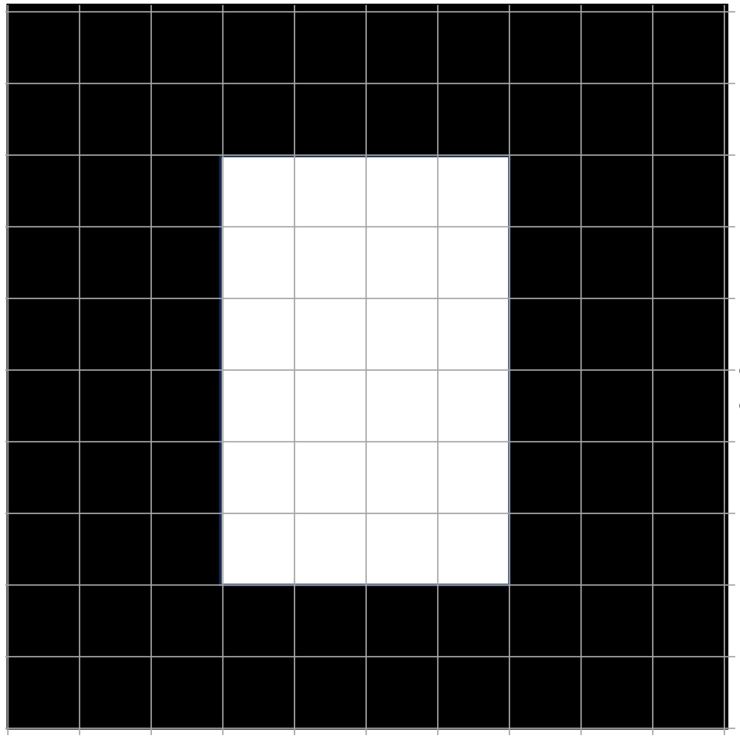
Gaussian:

$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

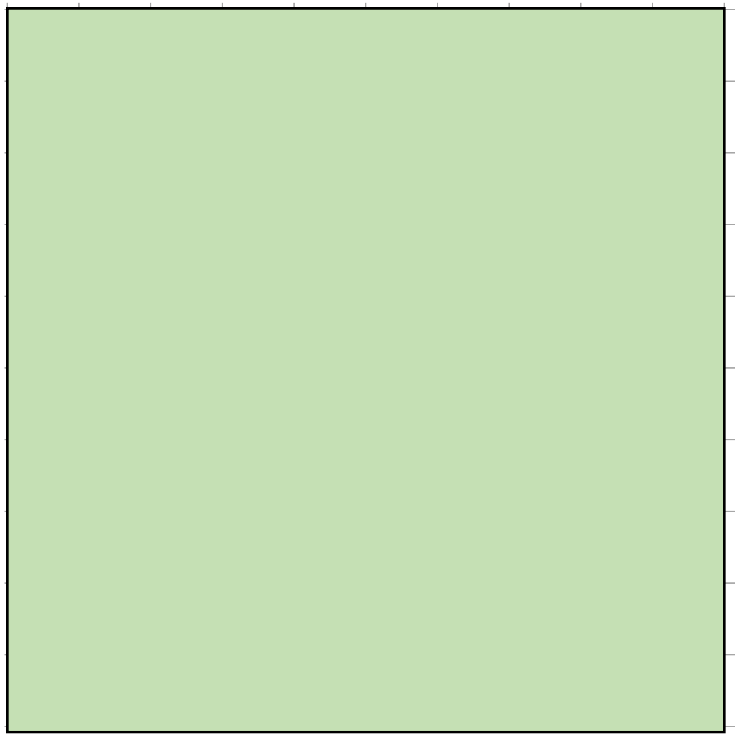


From Functions of Points to Functions of Neighborhoods

- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



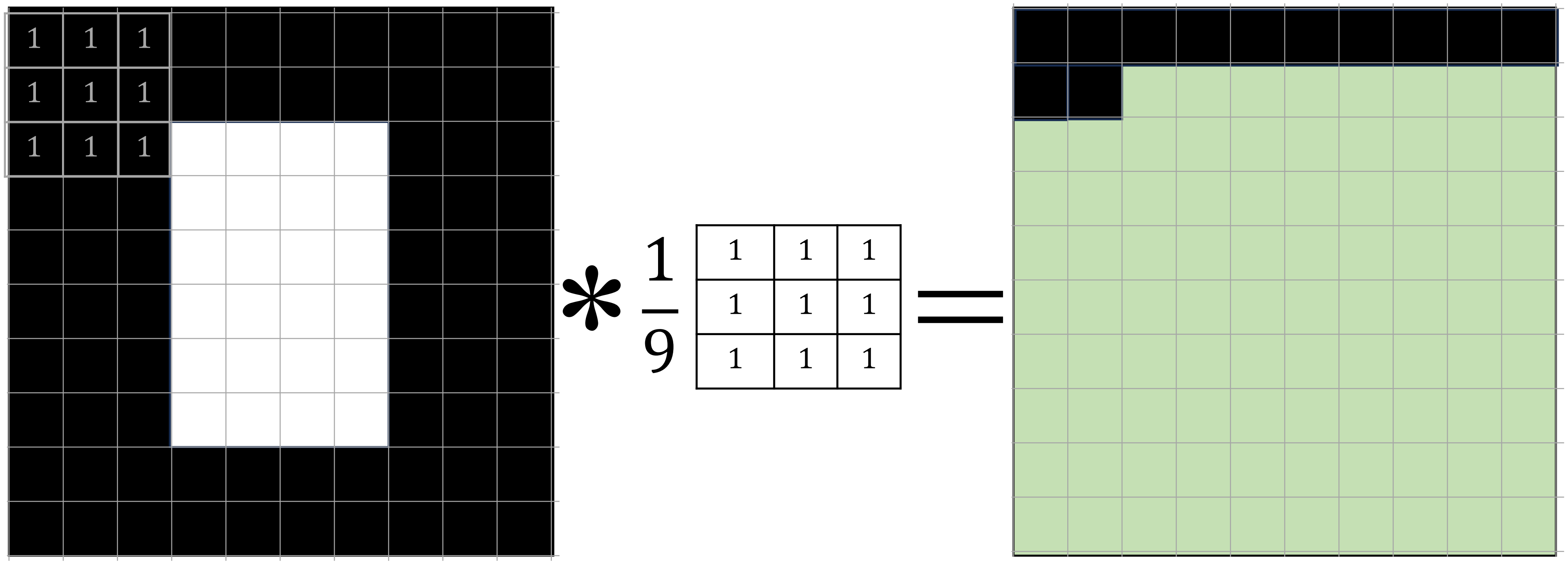
$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



$$(a \star b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']$$

From Functions of Points to Functions of Neighborhoods

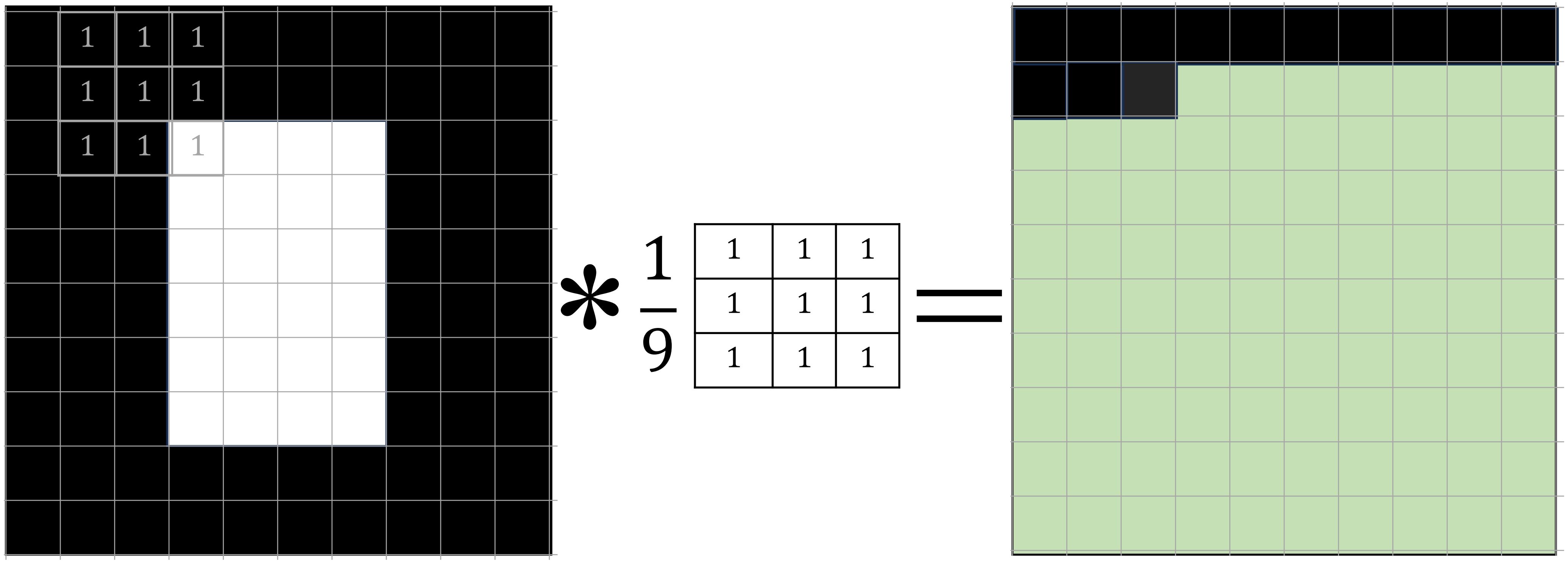
- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



$$(a \star b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']$$

From Functions of Points to Functions of Neighborhoods

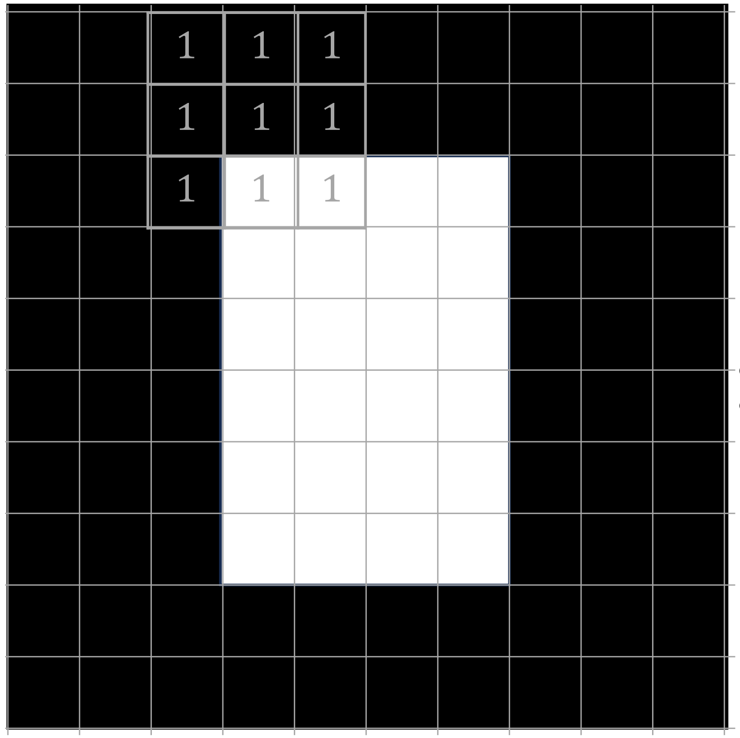
- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



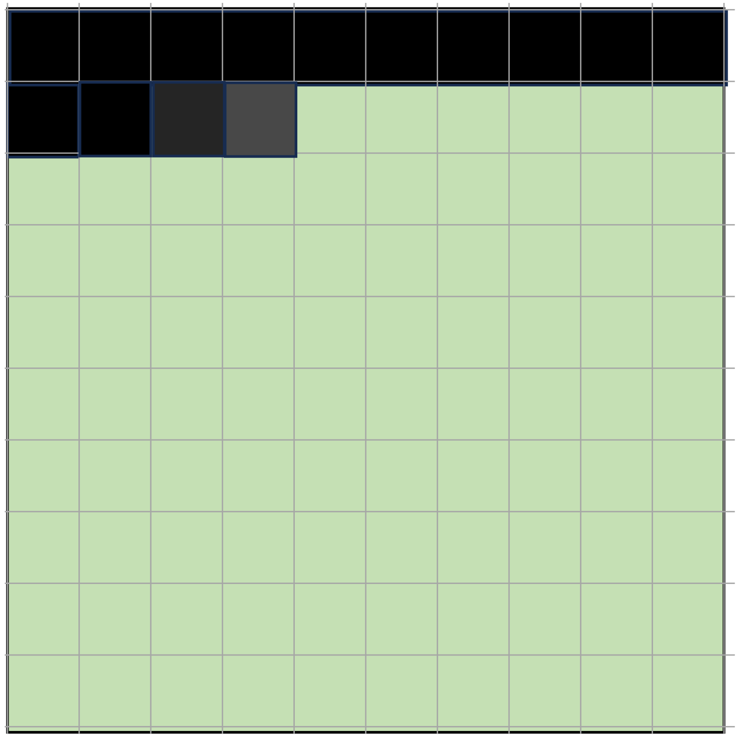
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

From Functions of Points to Functions of Neighborhoods

- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



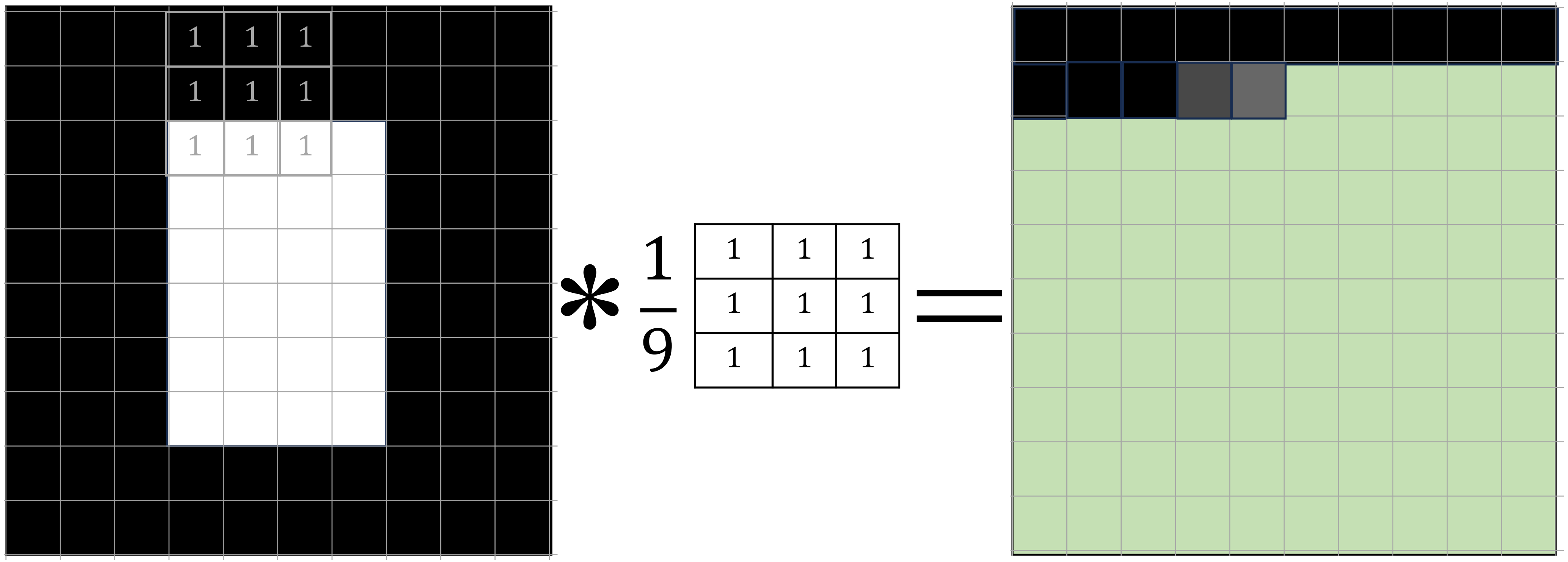
$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



$$(a \star b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']$$

From Functions of Points to Functions of Neighborhoods

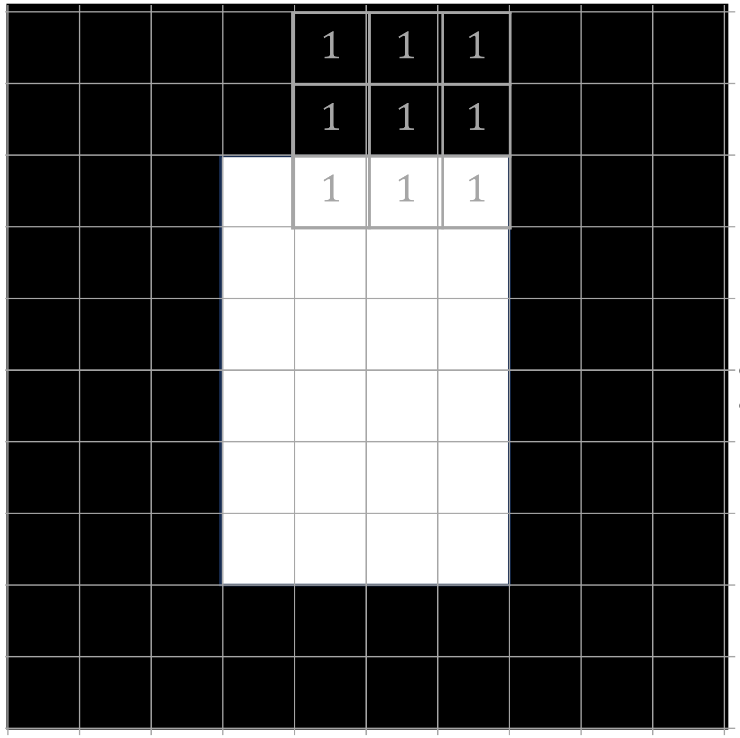
- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



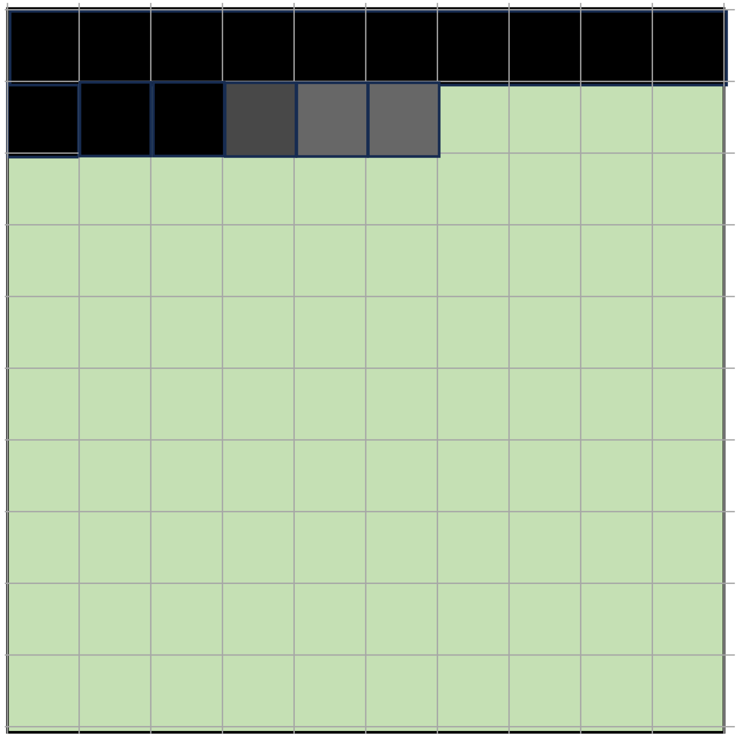
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

From Functions of Points to Functions of Neighborhoods

- Pointwise and warping operations express every output value as a function of ONE input value...
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



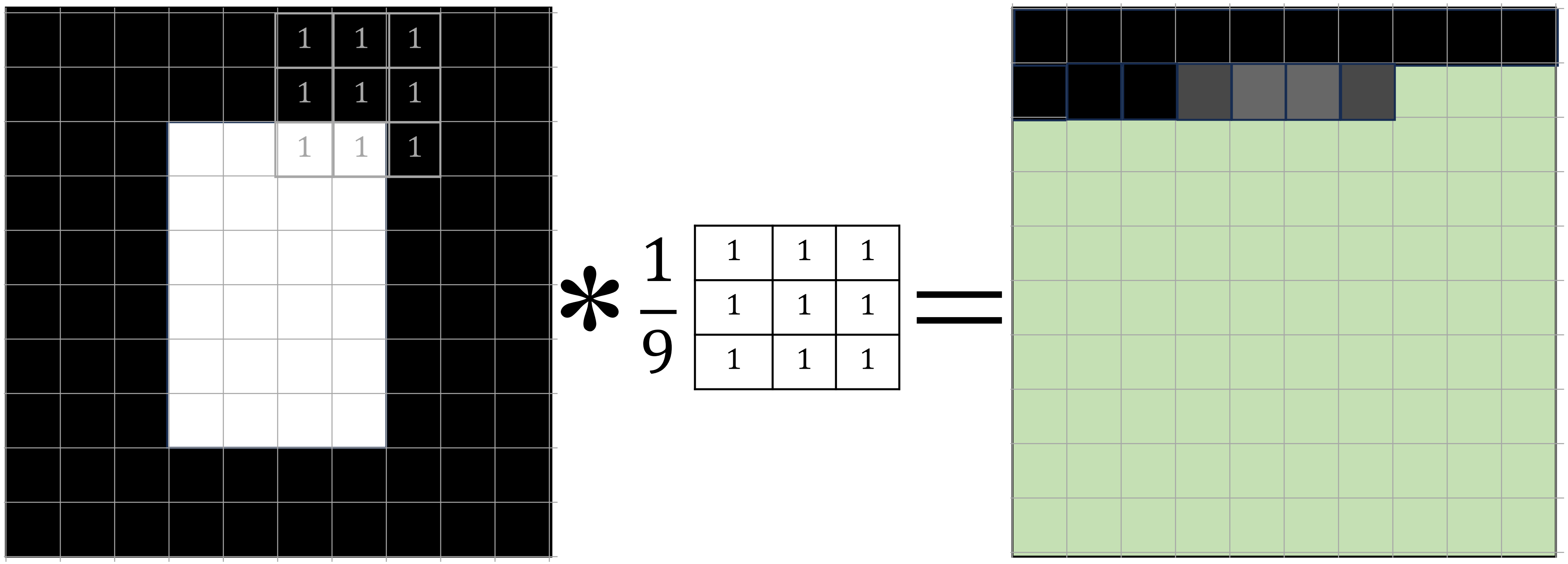
$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



$$(a \star b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']$$

From Functions of Points to Functions of Neighborhoods

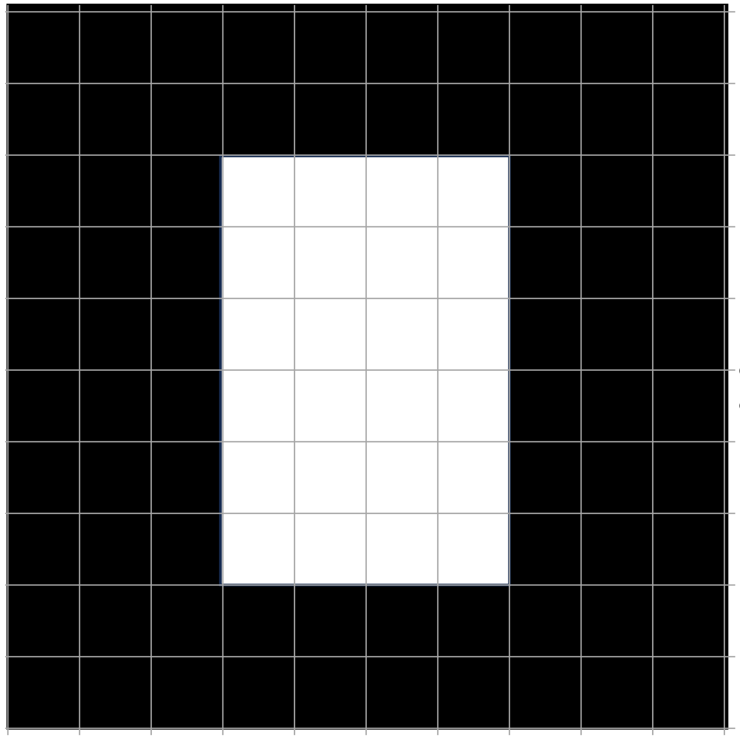
- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



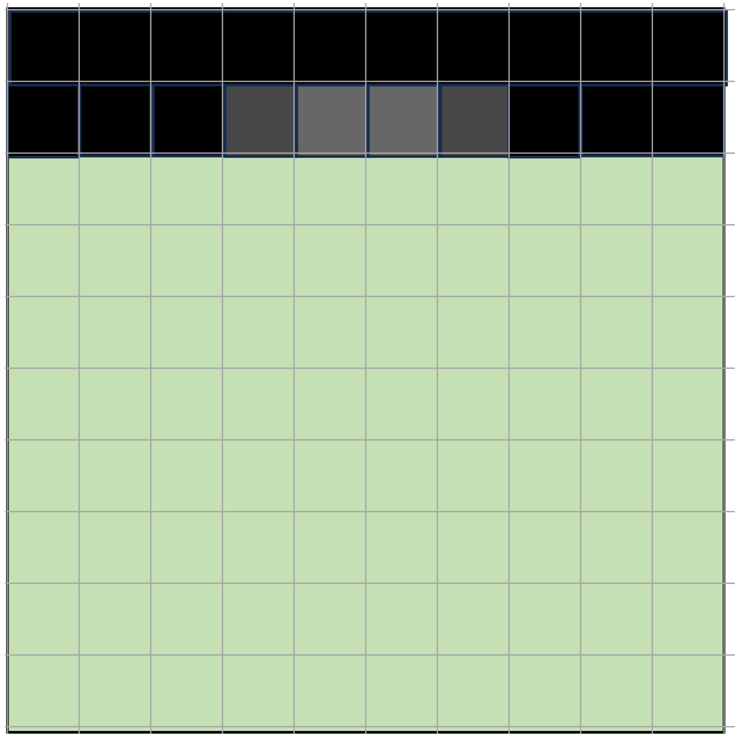
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

From Functions of Points to Functions of Neighborhoods

- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



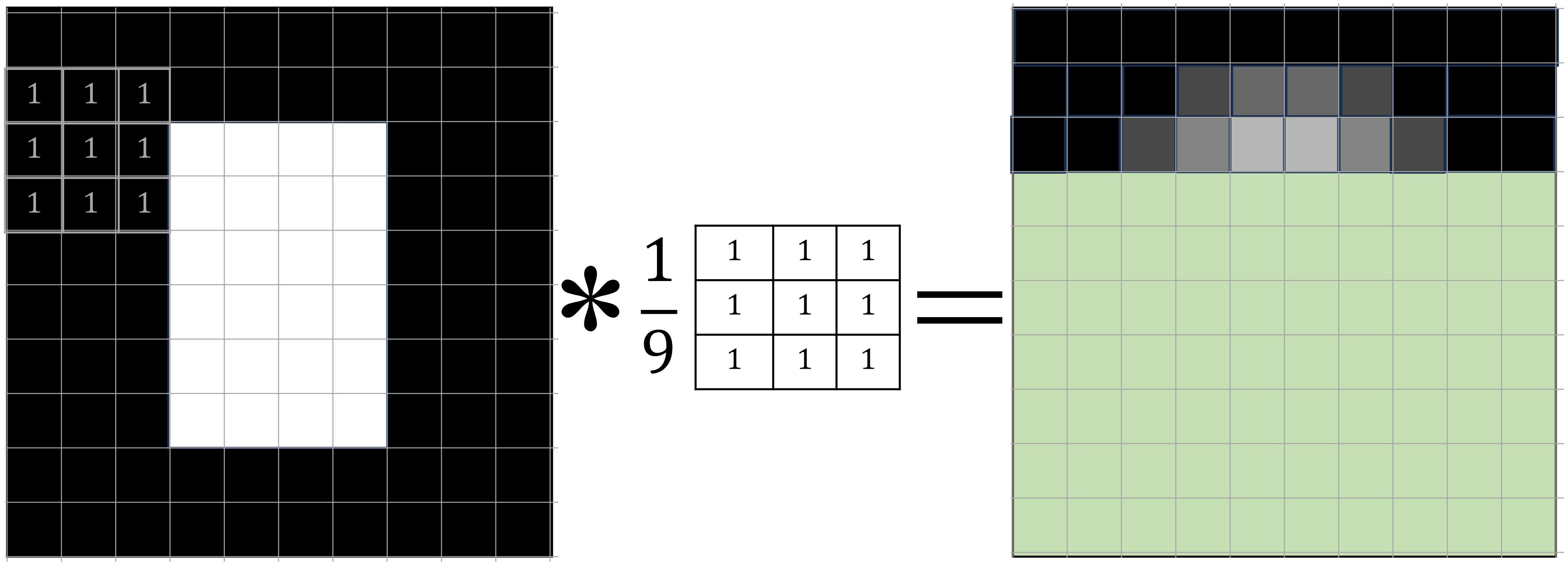
$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

From Functions of Points to Functions of Neighborhoods

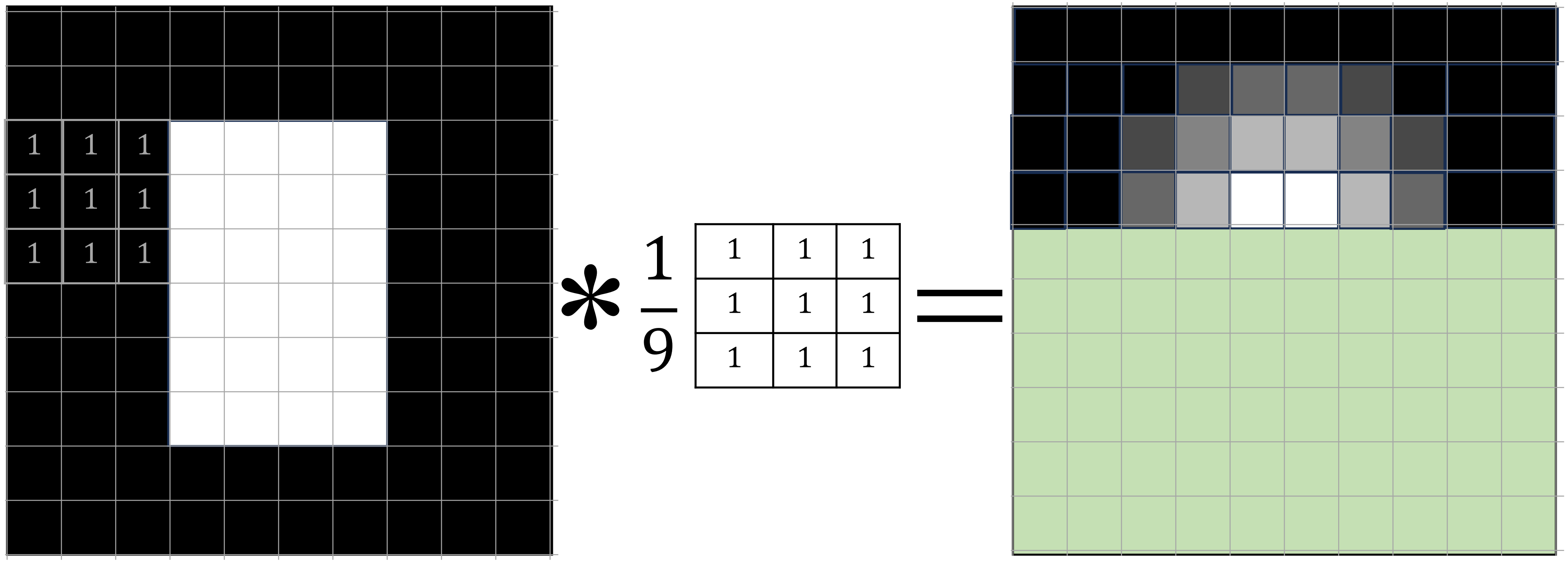
- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

From Functions of Points to Functions of Neighborhoods

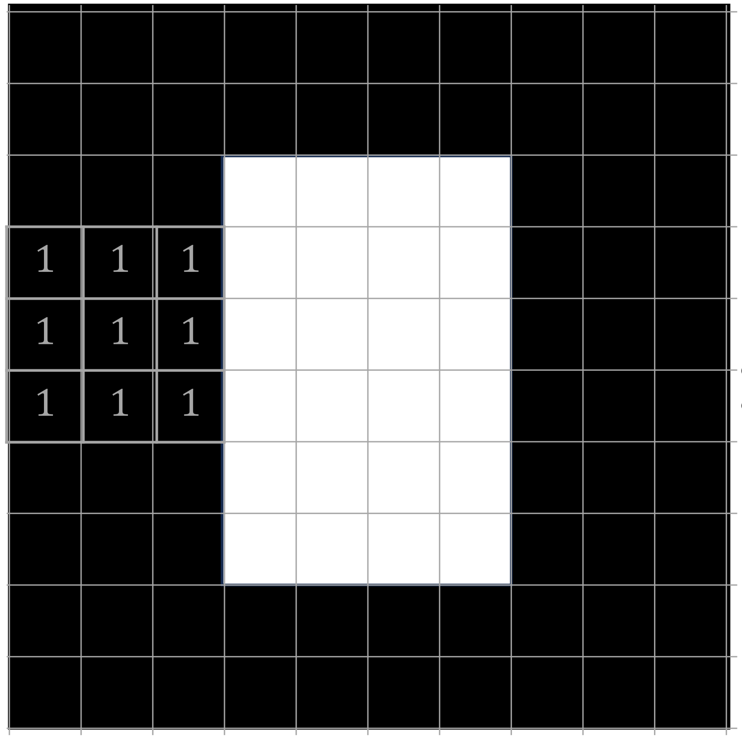
- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



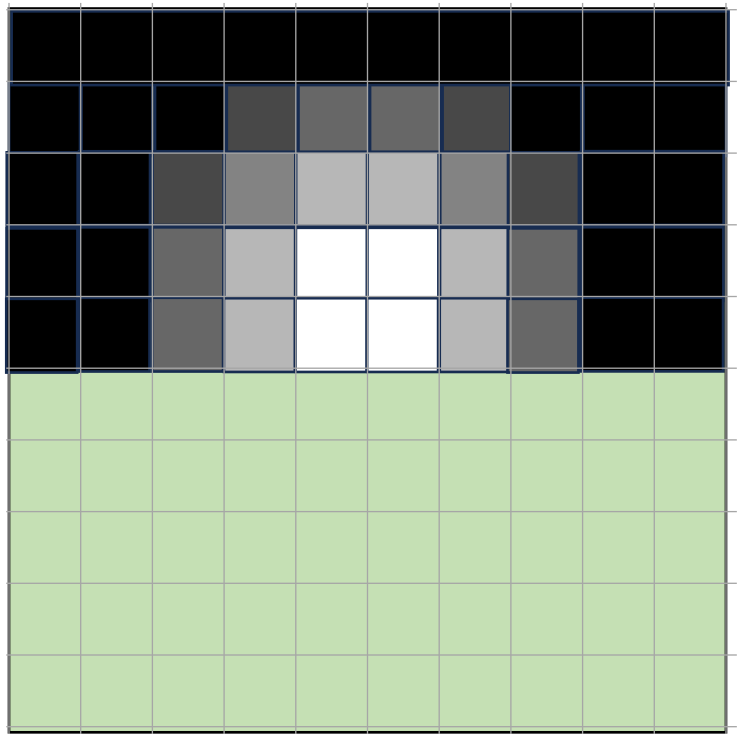
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']\,b[i-i',j-j']$$

From Functions of Points to Functions of Neighborhoods

- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a *linear* function of an input ***neighborhood***



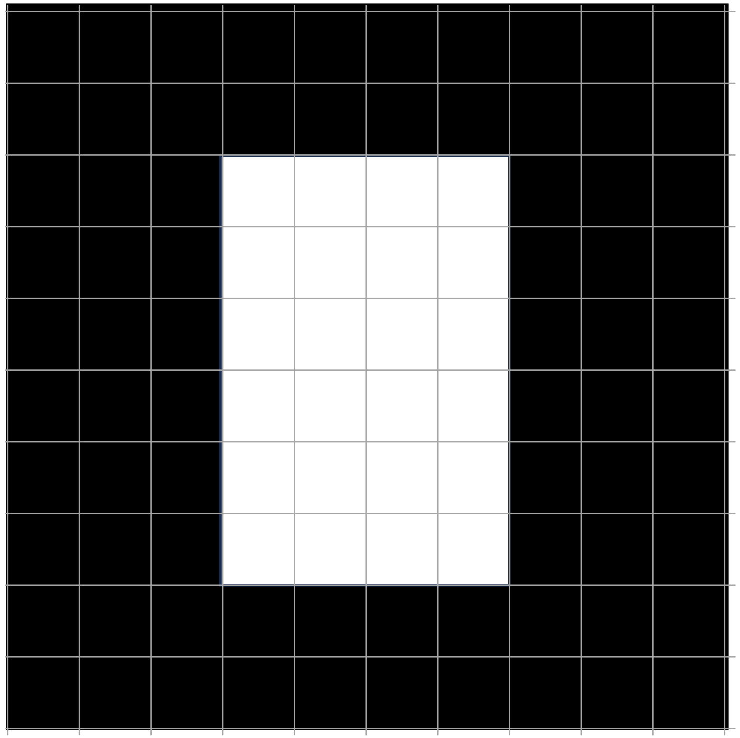
$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



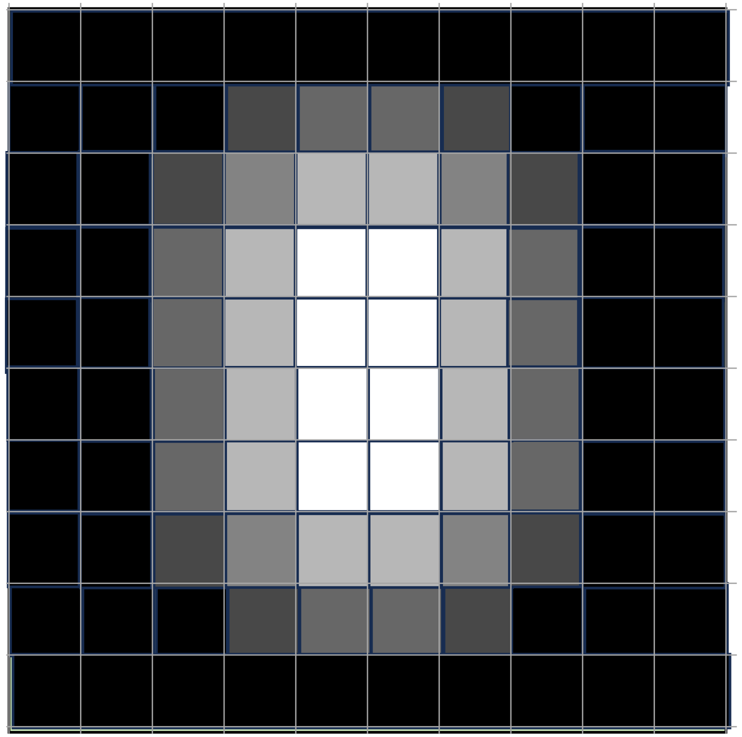
$$(a \star b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']$$

From Functions of Points to Functions of Neighborhoods

- Pointwise and warping operations express every output value as a function of ONE input value…
- Convolutions express every output point as a ***linear*** function of an input ***neighborhood***



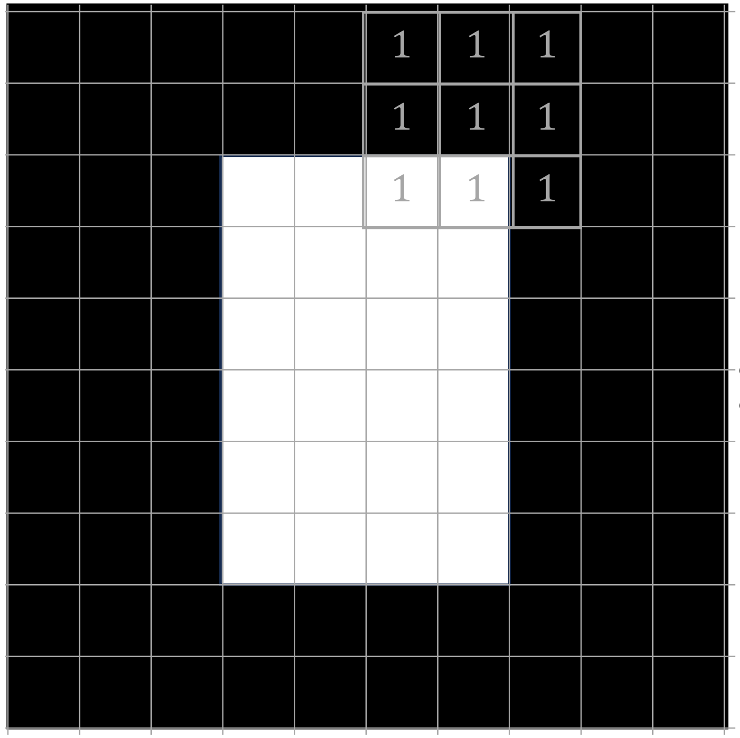
$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



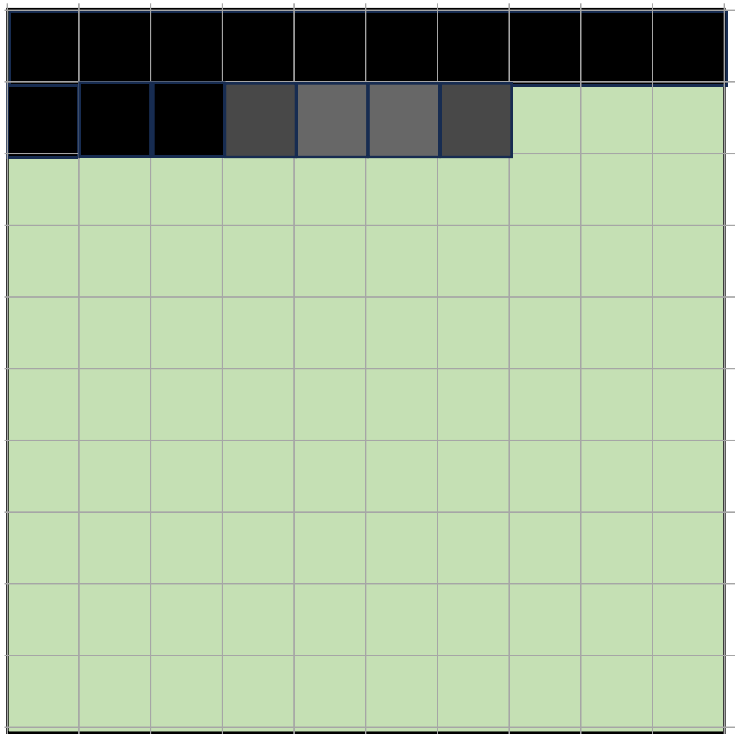
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']\, b[i-i', j-j']$$

Optimization: separable filters

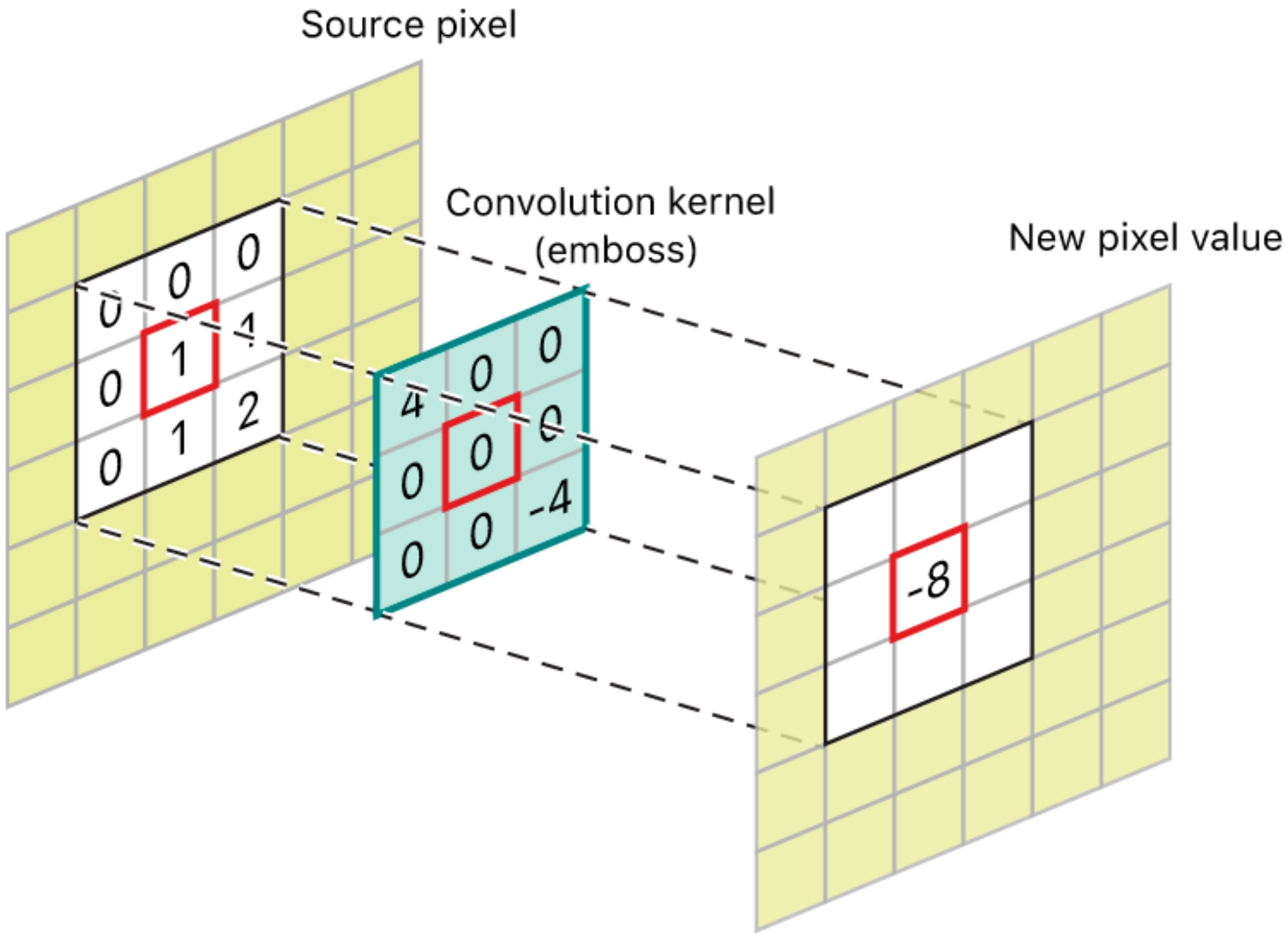
- basic alg. is $O(r^2)$: large filters get expensive fast!



$$* \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} =$$



$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$


Source pixel
Convolution kernel
(emboss)
New pixel value
0
0
0
1
0
1
1
0
2
4
0
0
0
0
0
0
0
-4
-8

[Philip Greenspun]

original △ | ▽ box blur

sharpened △| ▽gaussian blur

Find the filter? Sharpening

What do you think the sharpening filter looks like?

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Separable filtering

$$a_2[i,j] = a_1[i]a_1[j]$$

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\overbrace{\sum_{i'} a_1[i'] \left(\sum_{j'} \overbrace{a_1[j'] b[i-i', j-j']}^{\text{first, convolve with this}} \right)}^{\text{second, convolve with this}}$$

Optimization: separable filters

- basic alg. is $O(r^2)$: large filters get expensive fast!
- definition: $a_2[i,j]$ is *separable* if it can be written as:

$$a_2[i,j] = a_1[i]a_1[j]$$

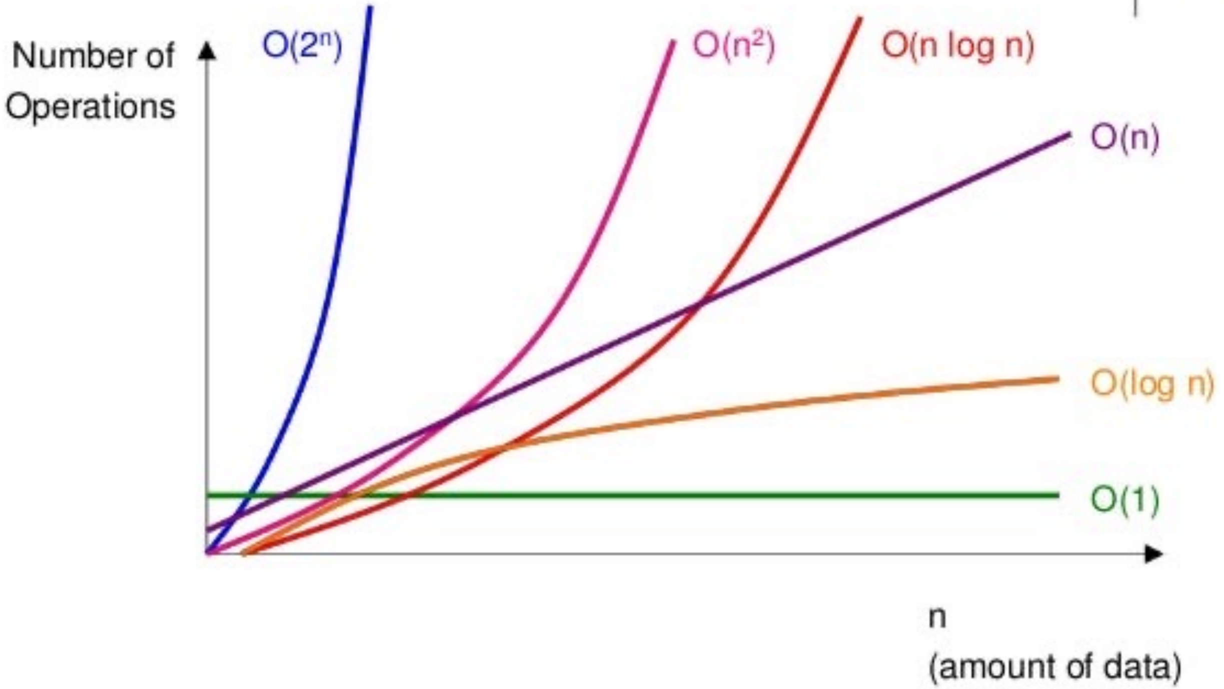
  - this is a useful property for filters because it allows factoring:

$$\begin{aligned}(a_2 \star b)[i,j] &= \sum_{i'}\sum_{j'} a_2[i',j']b[i-i',j-j'] \\ &= \sum_{i'}\sum_{j'} a_1[i']a_1[j']b[i-i',j-j'] \\ &= \sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j']b[i-i',j-j']\right)\end{aligned}$$

Side Note on Big O notation

- basic alg. is $O(r^2)$: large filters get expensive fast!
- What do we mean by $O(r^2)$?
- The O stands for Big O Notation. Used to show algorithmic complexity.
- r is the variable based on which the complexity of the algorithm varies.

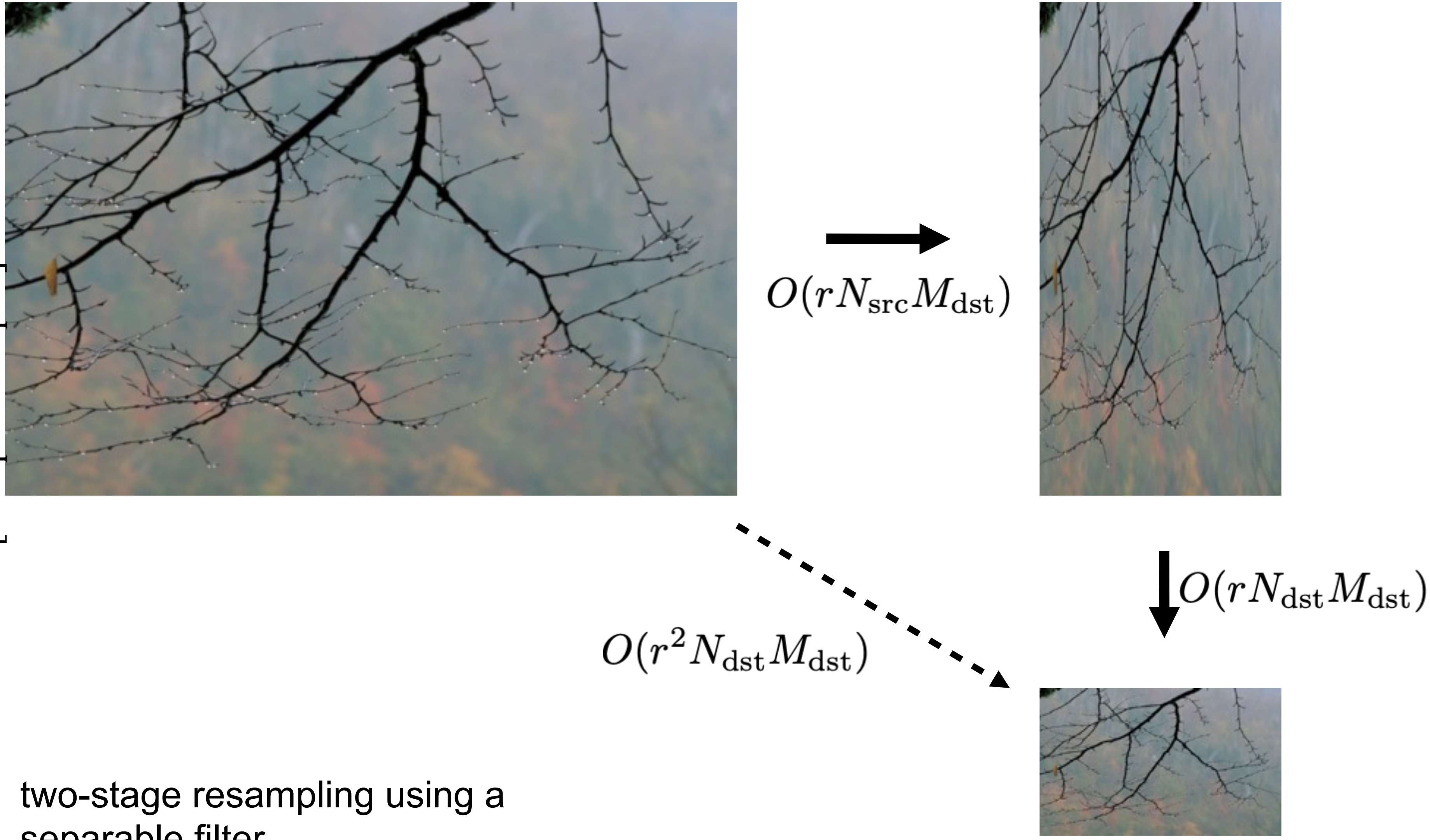
Comparing Big O Functions



Optimization: separable filters

- Why is this faster? Let's talk about the complexity of this operation?

$$
\begin{aligned}
(a_2 \star b)[i,j] &= \sum_{i'} \sum_{j'} a_2[i',j'] b[i-i', j-j'] \\
&= \sum_{i'} \sum_{j'} a_1[i'] a_1[j'] b[i-i', j-j'] \\
&= \sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j'] b[i-i', j-j'] \right)
\end{aligned}
$$

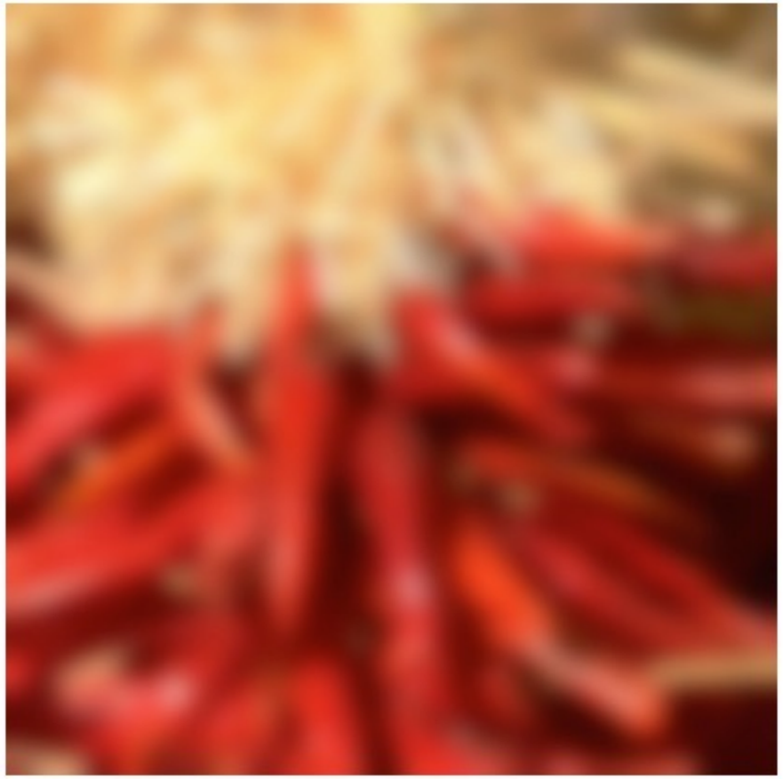


[Philip Greenspun]

two-stage resampling using a separable filter

Fine Details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[Philip Greenspun]

A very very short demo on Convolutional Neural Networks

[Link]

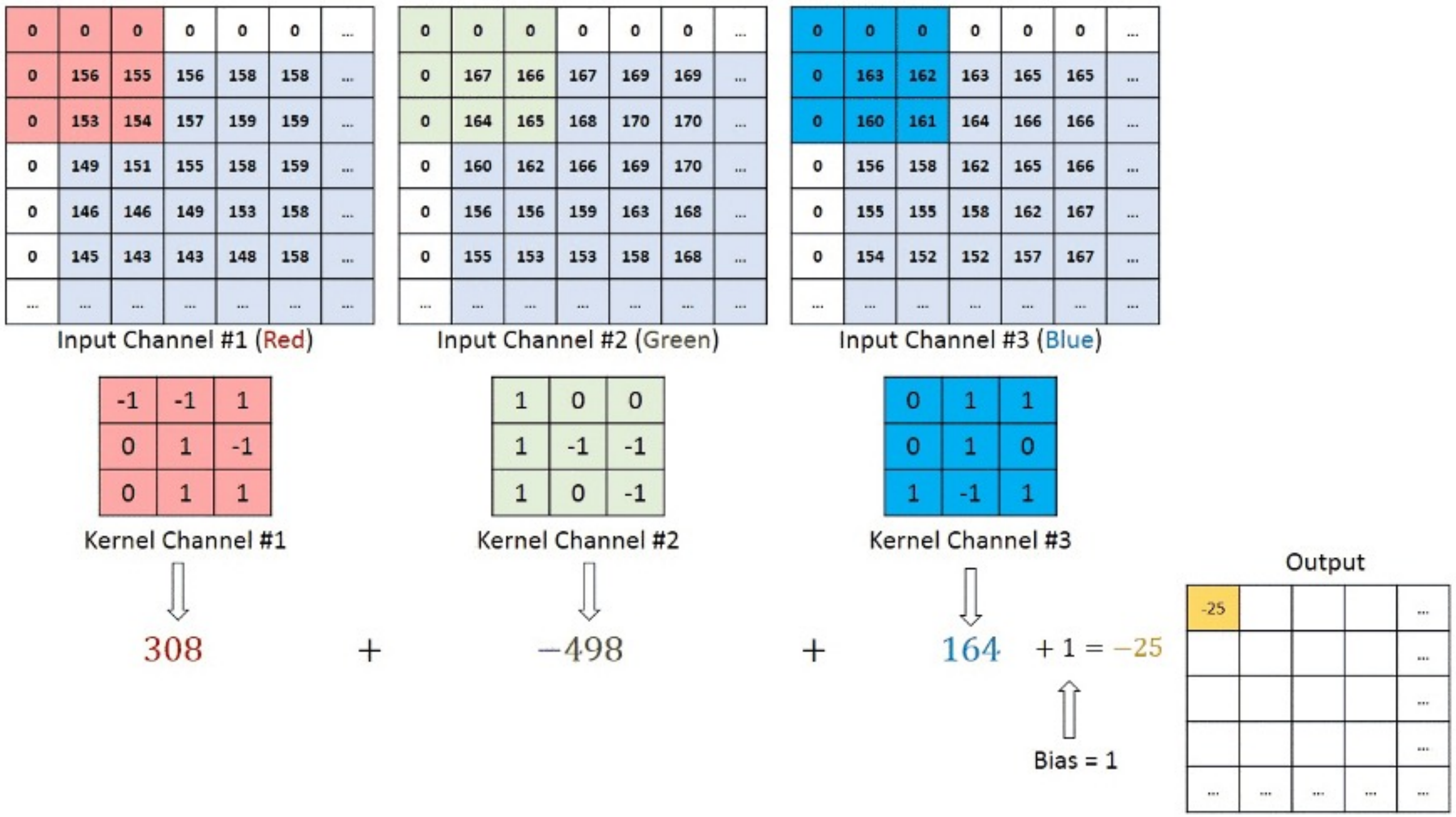
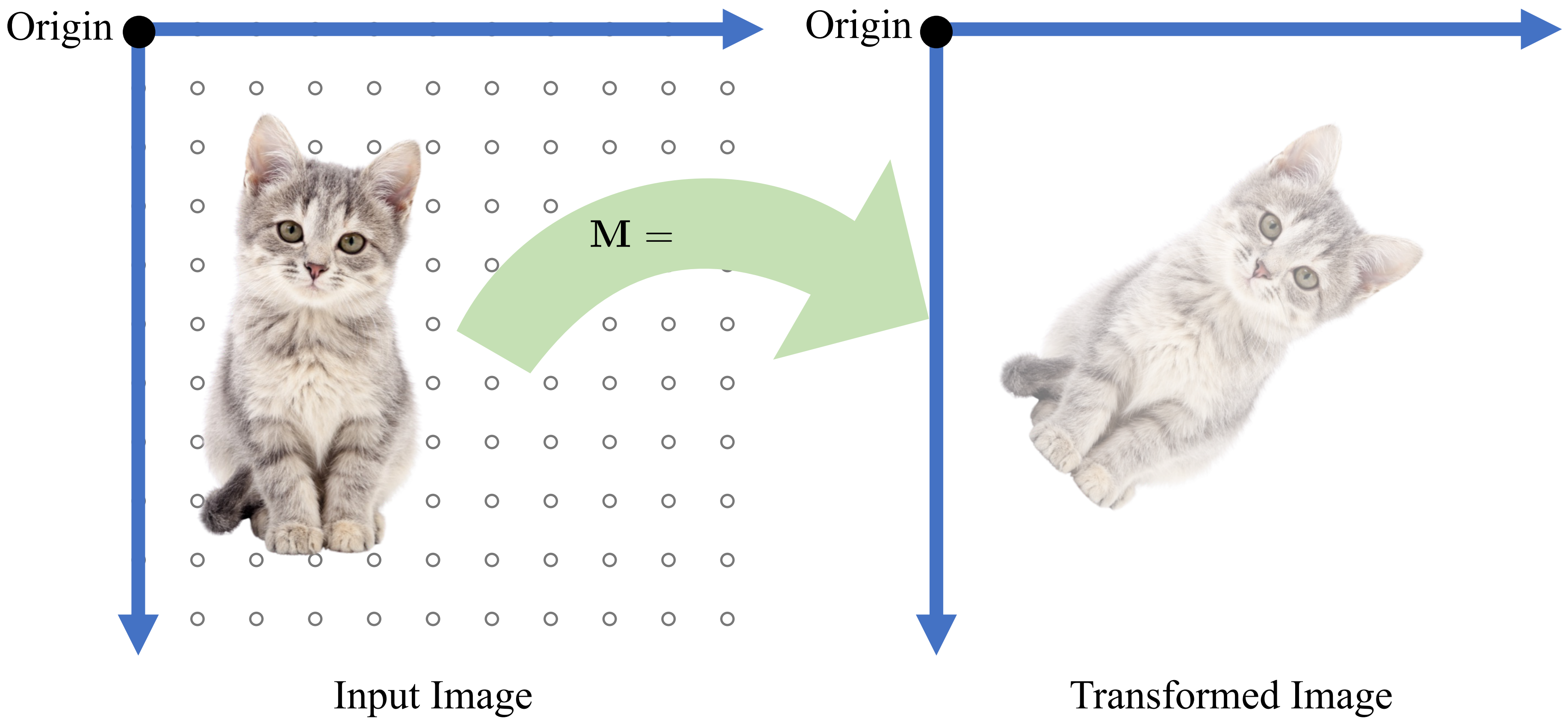


Image Warping

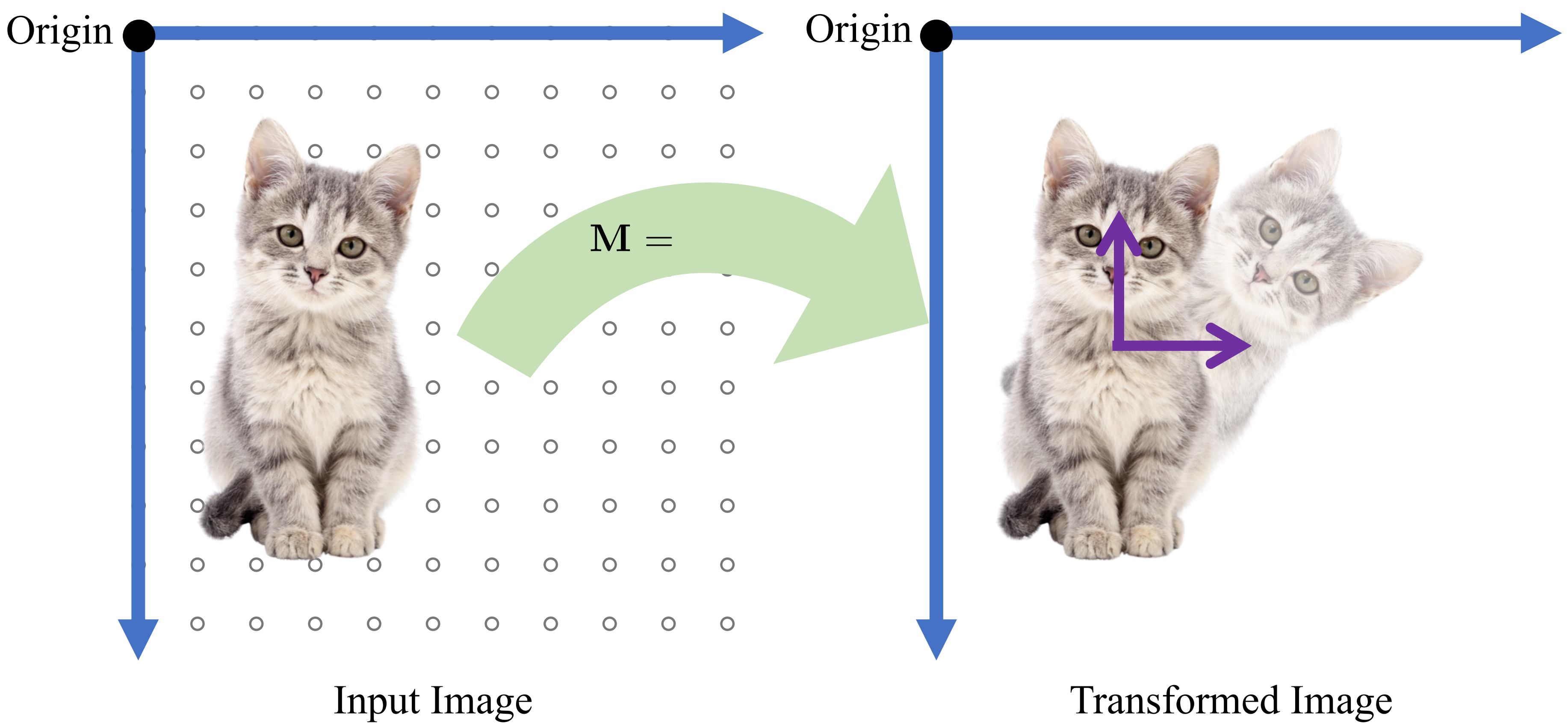
Transformations as Resampling + Interpolation

- Halloween special: [Transformations in Horror movies](#)!



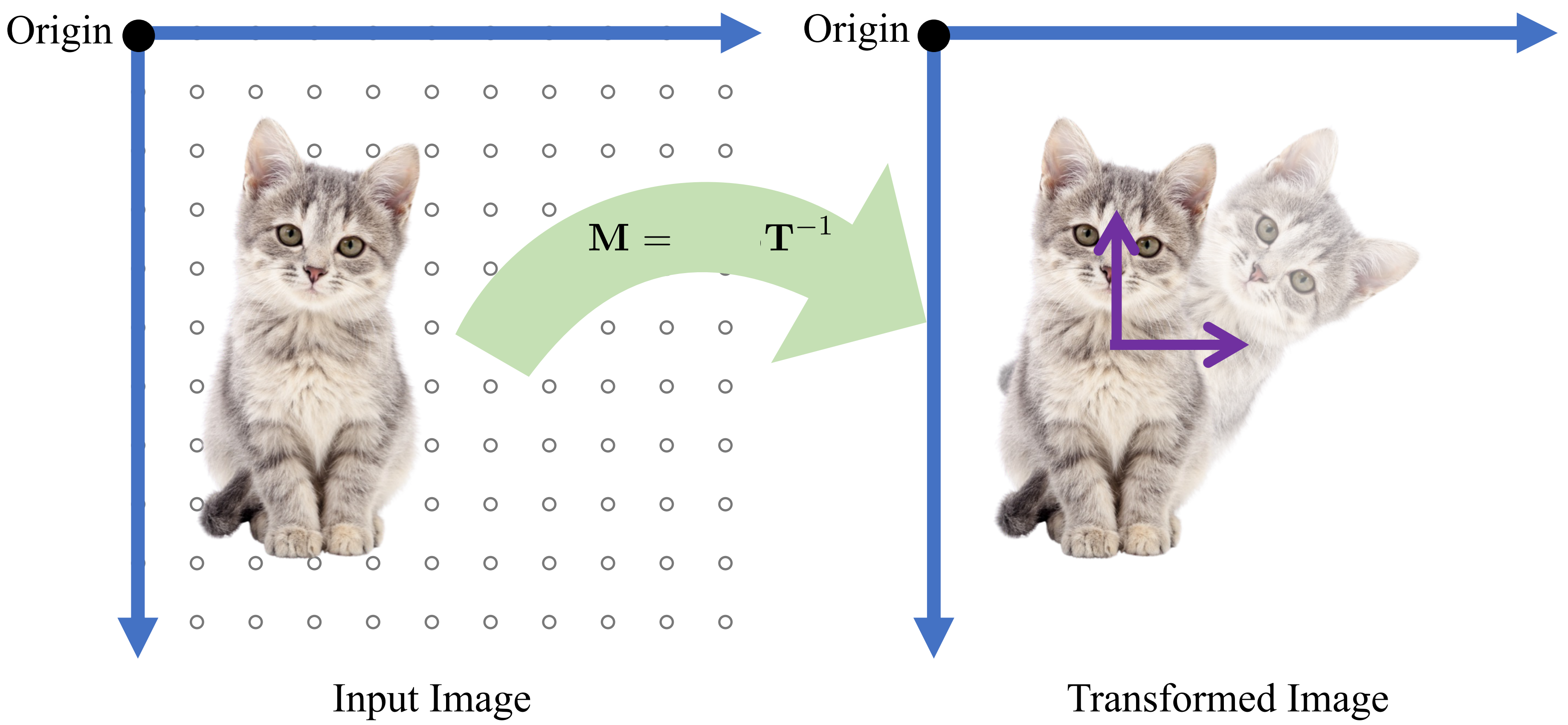
Transformations as Resampling + Interpolation

- In assignments 1-2 we used matrices to transform the scene
- Let's examine the same operation in image space…



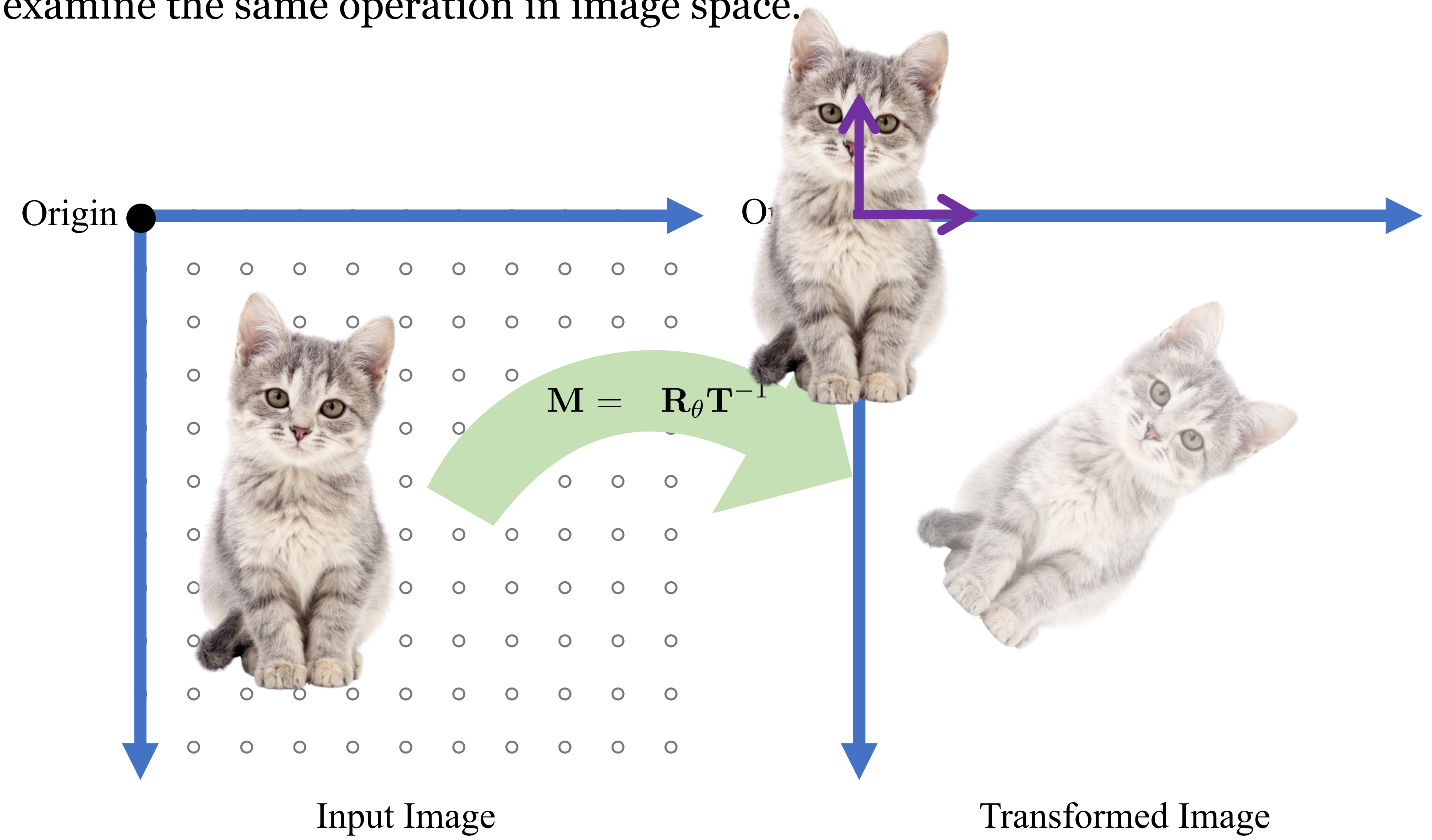
Transformations as Resampling + Interpolation

- In assignments 1-2 we used matrices to transform the scene
- Let's examine the same operation in image space…



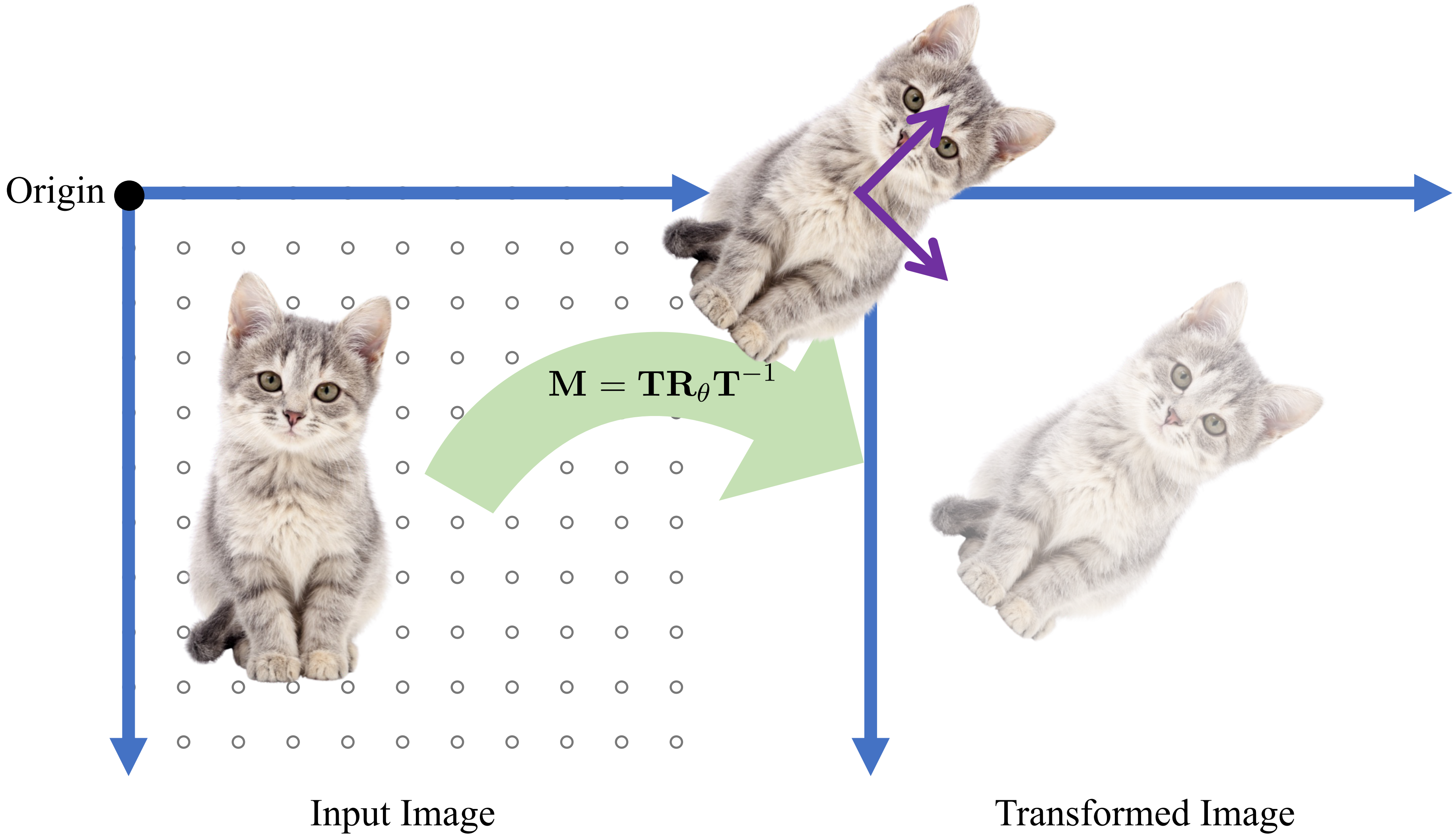
Transformations as Resampling + Interpolation

- In assignments 1-2 we used matrices to transform the scene
- Let's examine the same operation in image space.



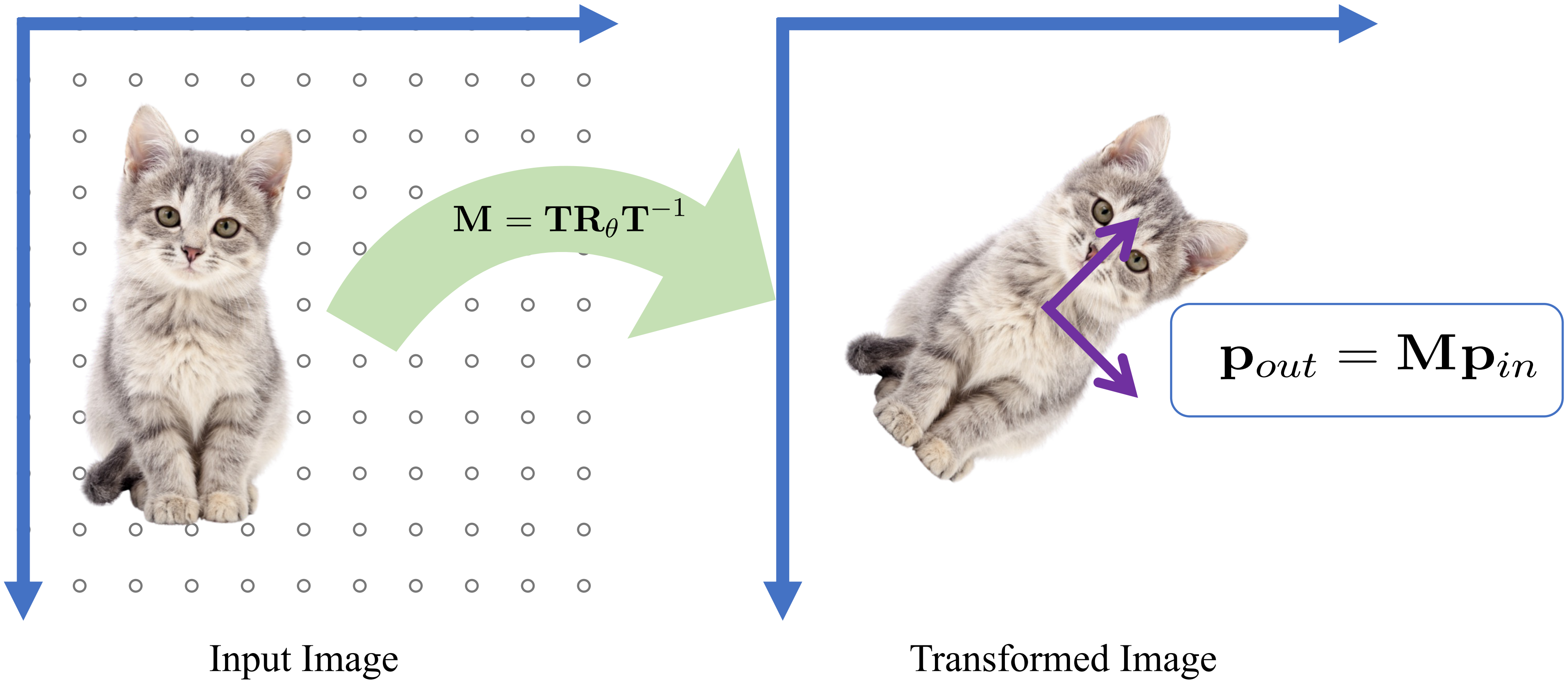
Transformations as Resampling + Interpolation

- In assignments 1-2 we used matrices to transform the scene
- Let's examine the same operation in image space...



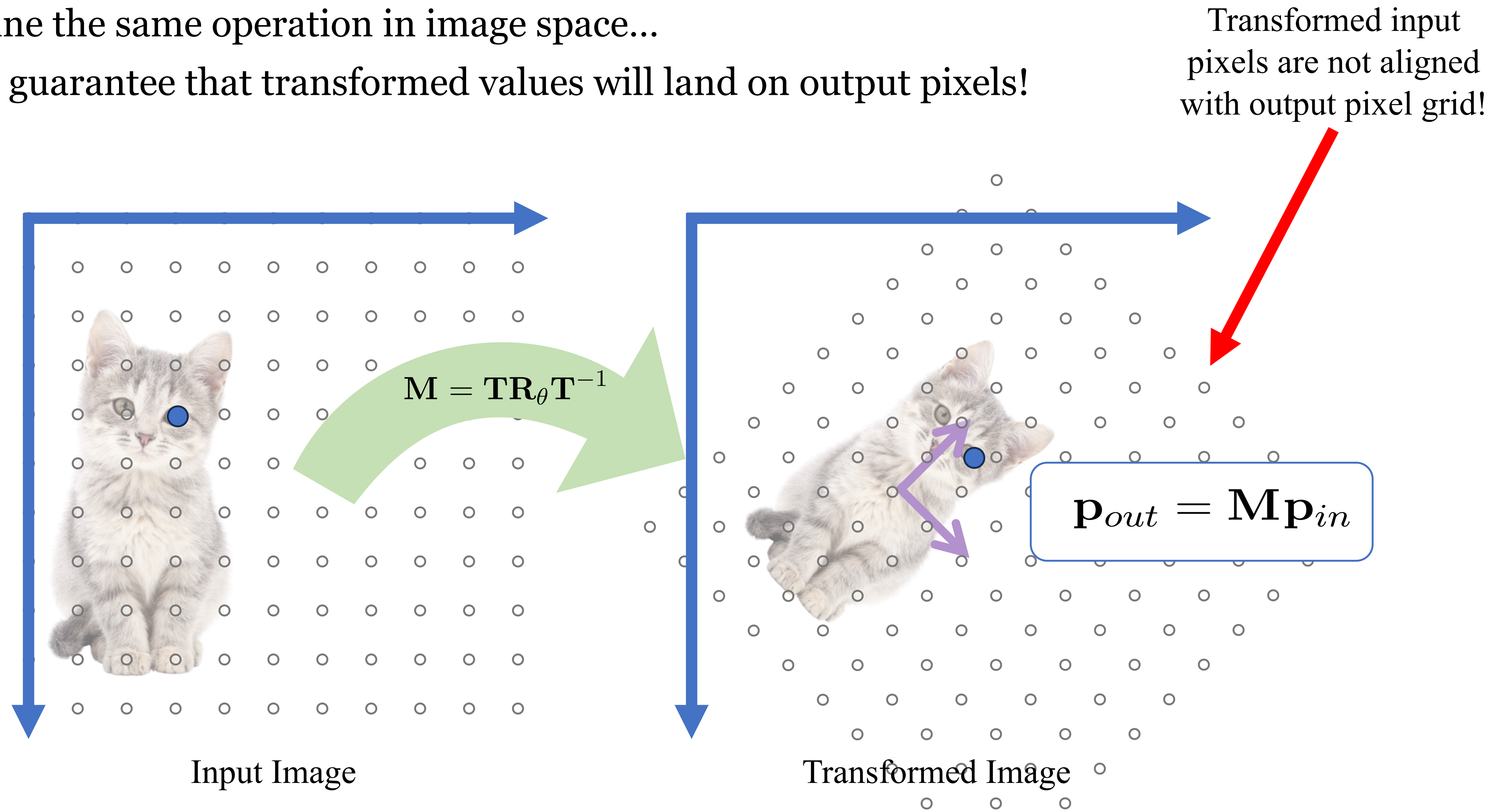
Transformations as Resampling + Interpolation

- In assignments 1-2 we used matrices to transform the scene
- Let's examine the same operation in image space…



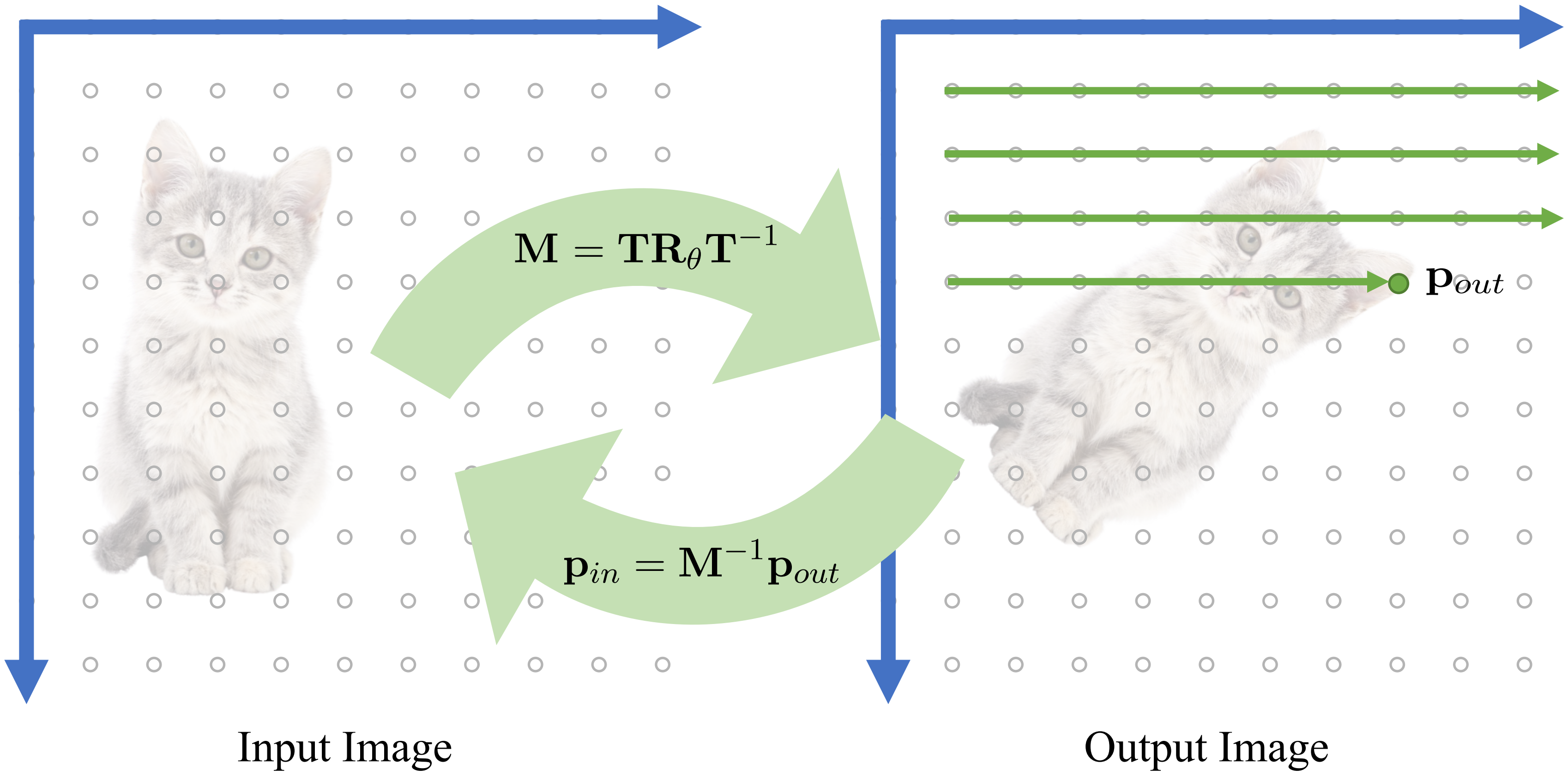
Transformations as Resampling + Interpolation

- In assignments 1-2 we used matrices to transform the scene
- Let's examine the same operation in image space...
- There is no guarantee that transformed values will land on output pixels!



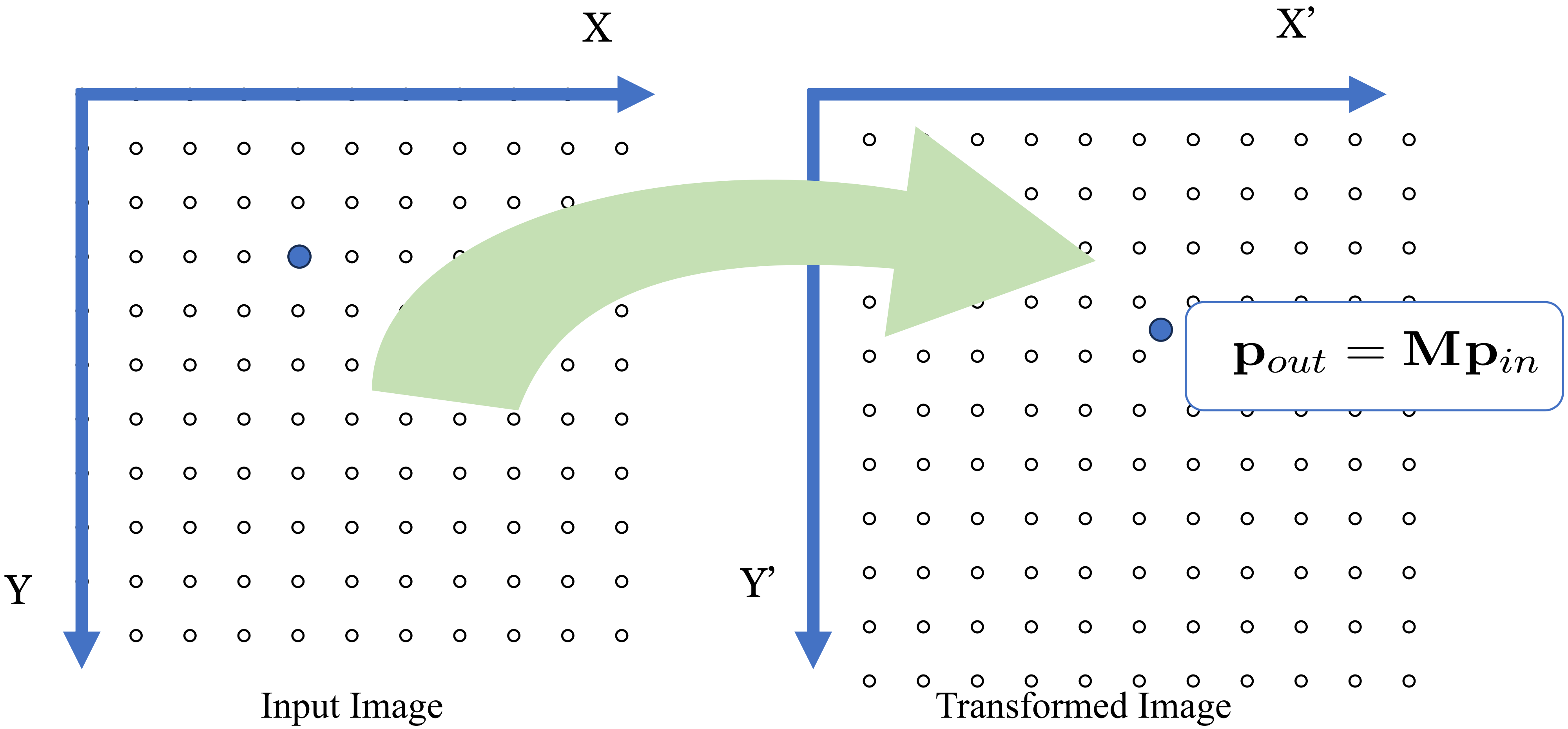
Transformations as Resampling + Interpolation

- **New Strategy**: Iterate through output pixels, and for each output pixel look up the matching input pixel value (with interpolation)
- If $\mathbf{M}$ is the matrix that takes geometry from our input scene to our output scene, then $\mathbf{M}^{-1}$ is the matrix that takes us from our output pixel to its corresponding source pixel



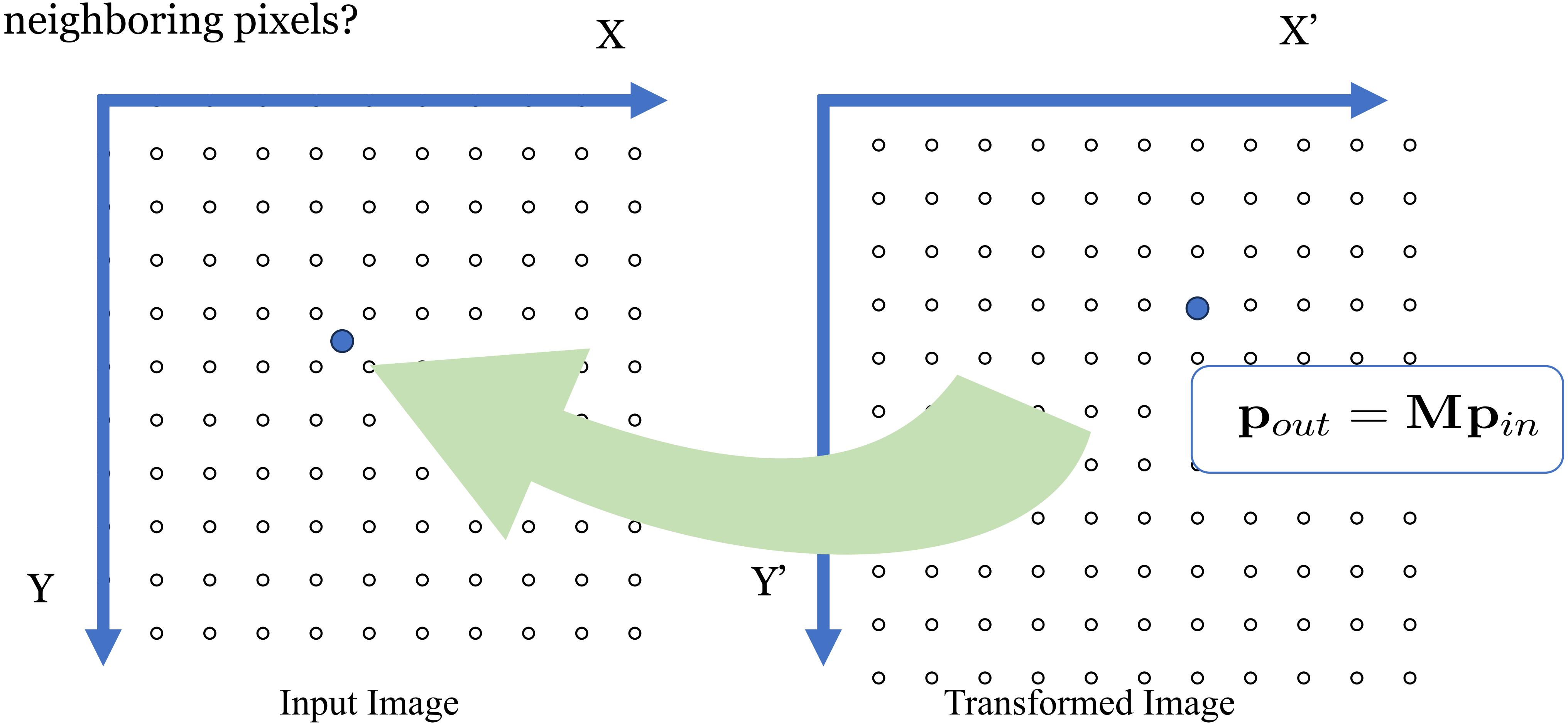
Transformations as Resampling + Interpolation

- You can do a forward warping process. What could be some potential issues with this?
- Basically, finding what portion of the color is distributed to the neighboring pixels when the input pixel shows up in between pixels of the transformed image?



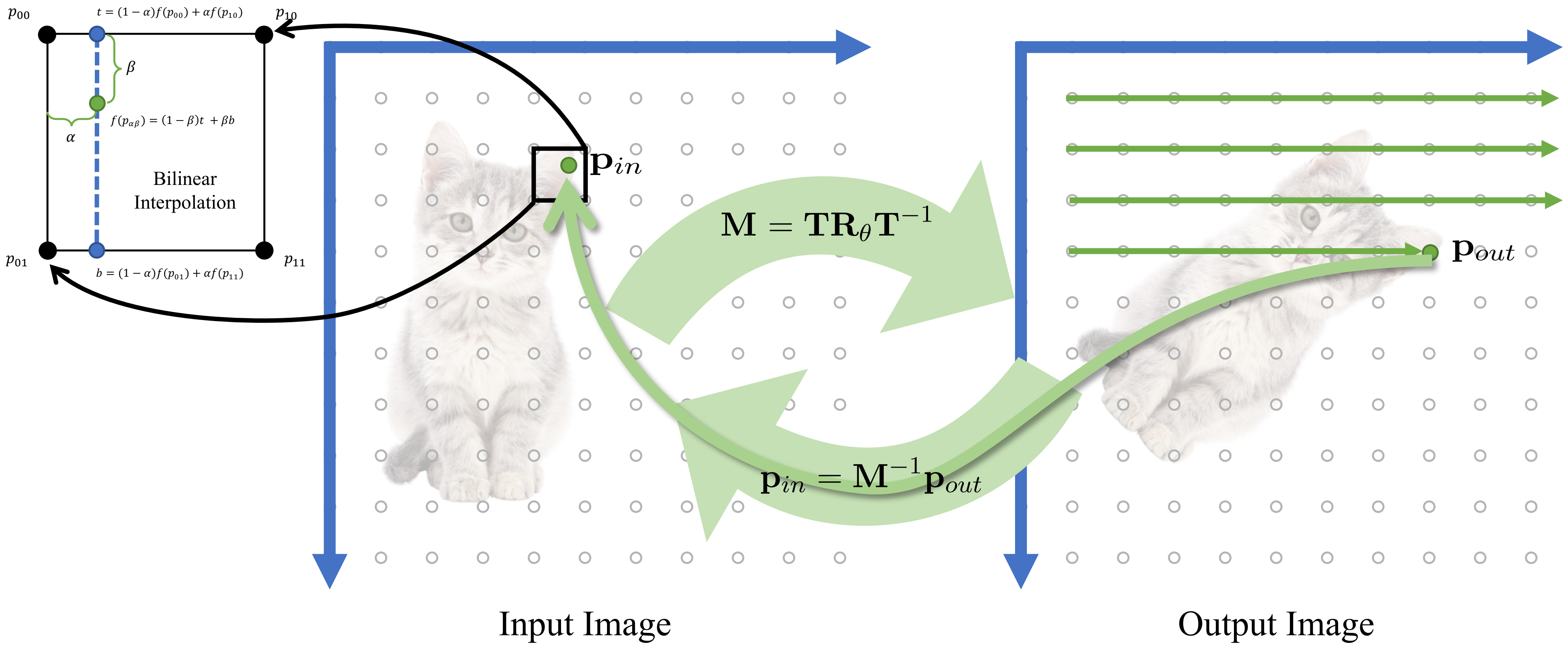
Transformations as Resampling + Interpolation

- Or you can do a backward process where you find the color in the output map by mapping and finding the values from the input image
- In the backward transformation process how do you assign the color of the pixel from the color of the neighboring pixels?



Transformations as Resampling + Interpolation

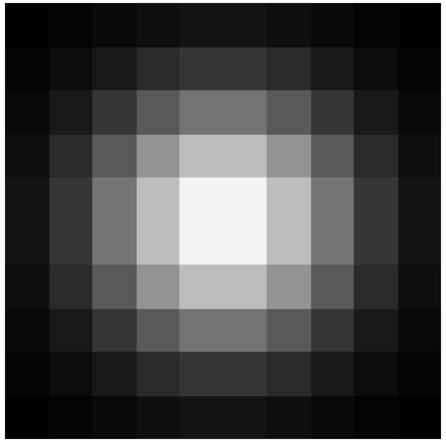
- **New Strategy**: Iterate through output pixels, and for each output pixel look up the matching input pixel value (with interpolation)
- If $\mathbf{M}$ is the matrix that takes geometry from our input scene to our output scene, then $\mathbf{M^{-1}}$ is the matrix that takes us from our output pixel to its corresponding source pixel

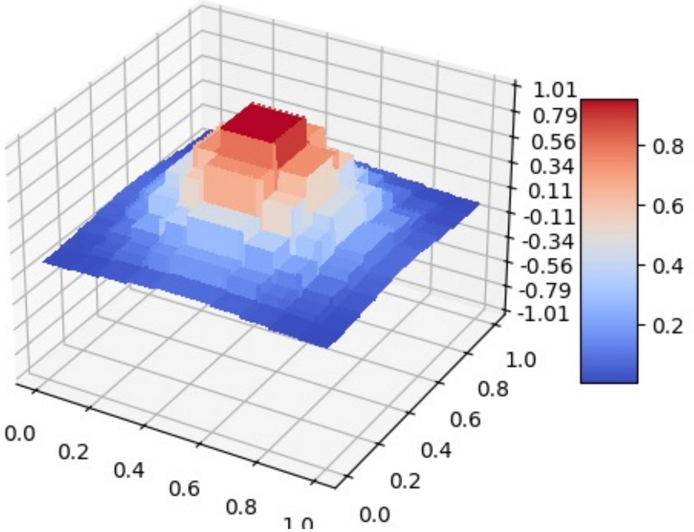


Recap: Nearest Neighbor vs Bilinear Interpolation in Images

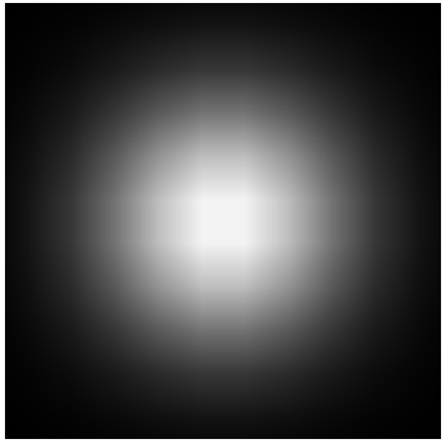
(tiny blur image)

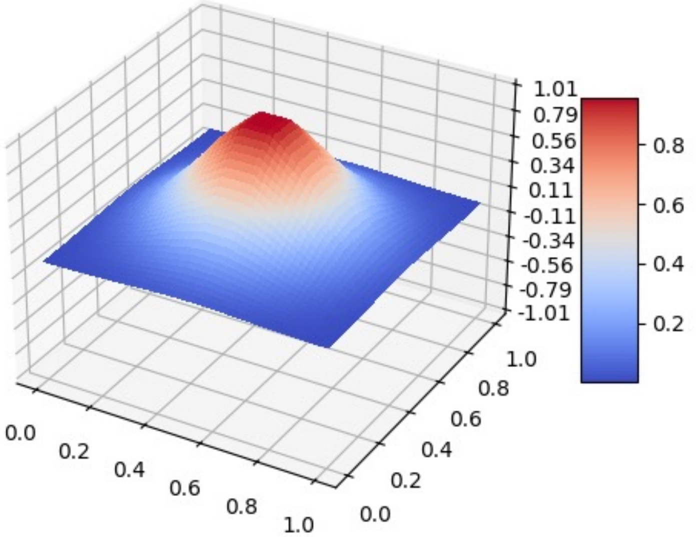
Nearest Neighbor
Interpolation





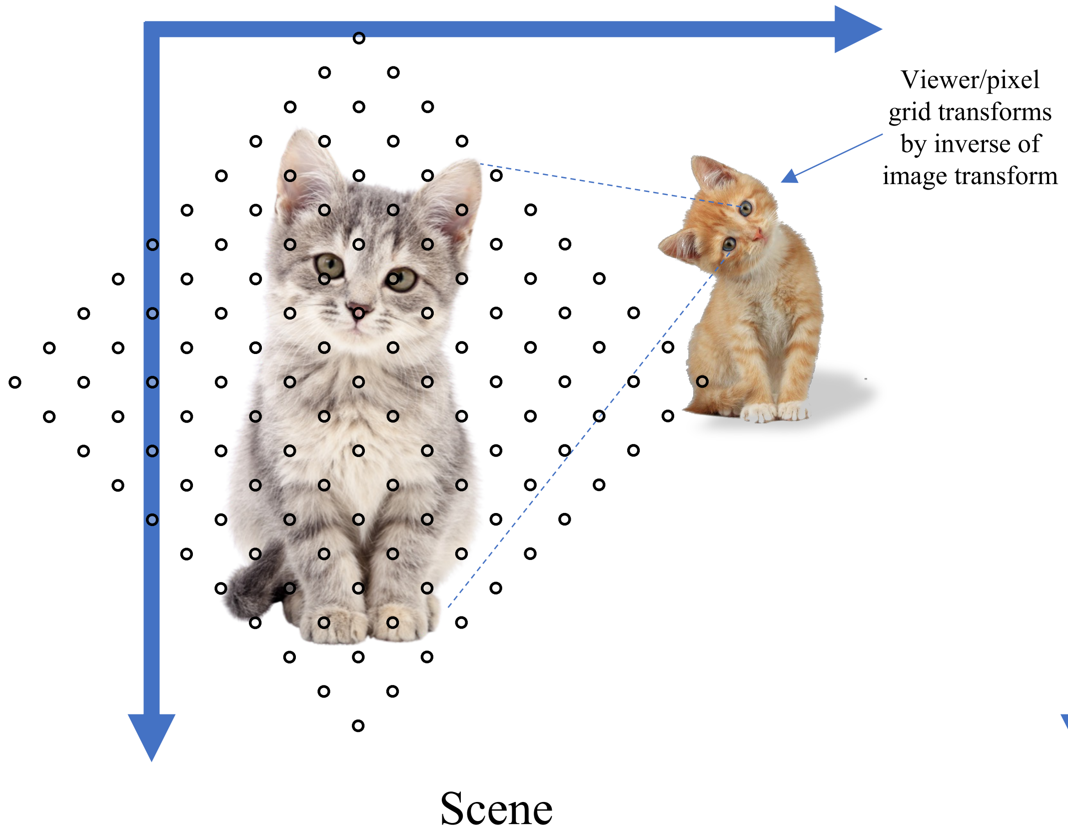
Bilinear
Interpolation

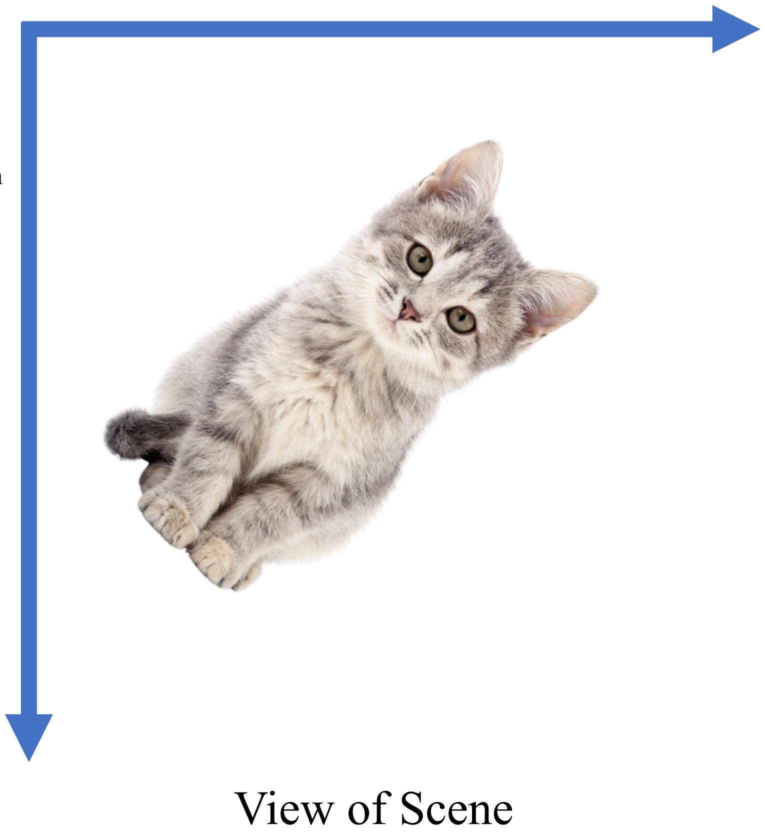




Transformations as Resampling + Interpolation

- **New Strategy**: Iterate through output pixels, and for each output pixel look up the matching input pixel value (with interpolation)
- If $\mathbf{M}$ is the matrix that takes geometry from our input scene to our output scene, then $\mathbf{M}^{-1}$ is the matrix that takes us from our output pixel to its corresponding source pixel





More on Sampling and Aliasing next class!

- How is this even possible?