Geometric transformations of images

D.A. Forsyth,

University of Illinois at Urbana Champaign

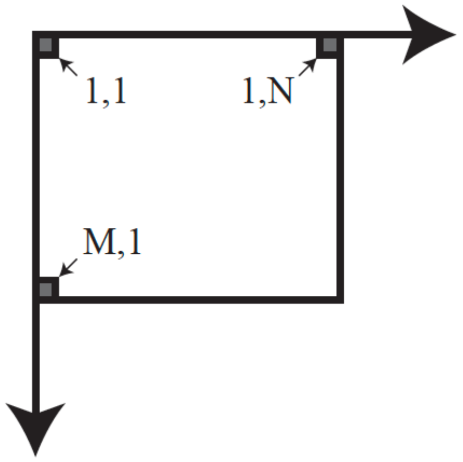
Nuisances in applying geometric tx to images

- Image coordinate systems are weird

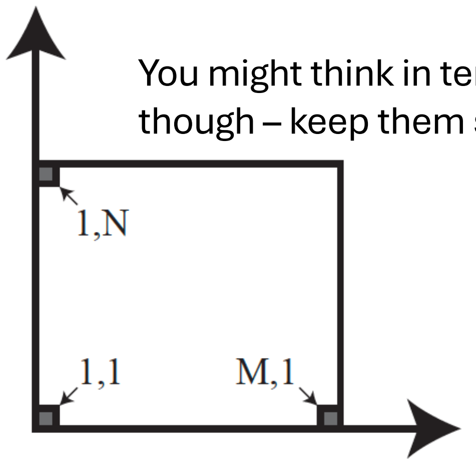
- You have to *put* an image somewhere
  - Very often, the transformation you have in mind puts the image outside the span of the target
    - what to do?

Image coordinate systems

I'll use this one, cause everyone does

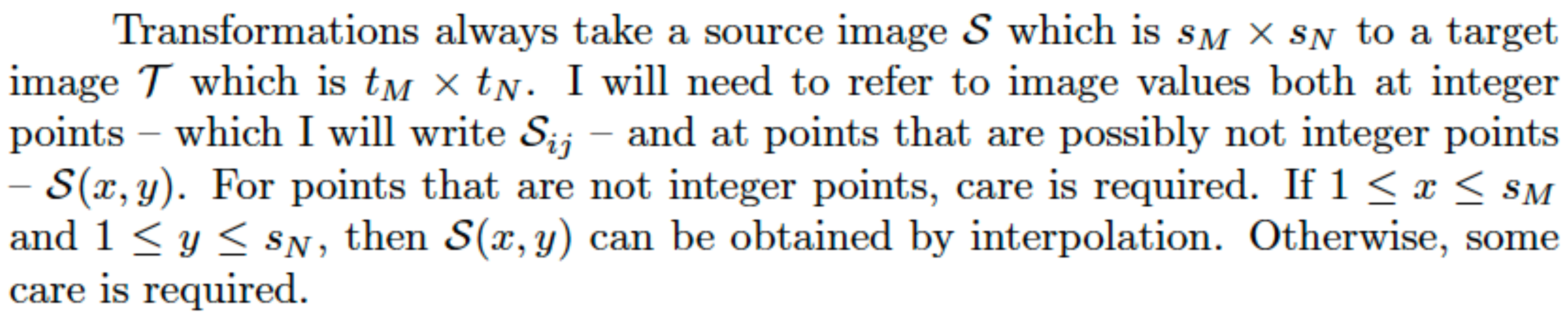


You might think in terms of this one, though – keep them straight

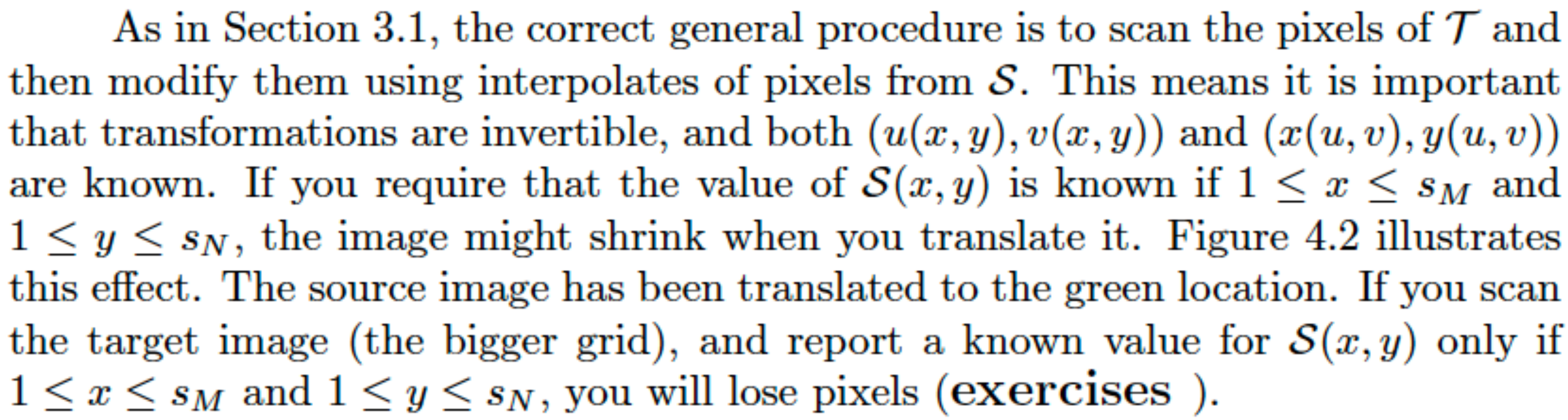


To add to the fun, in some API's images run 0..M-1 (Python, etc) and in others they run 1..M (Matlab). Keep an eye on this point, or you will lose pixels or have code errors.

Inverse warping requires a little care…

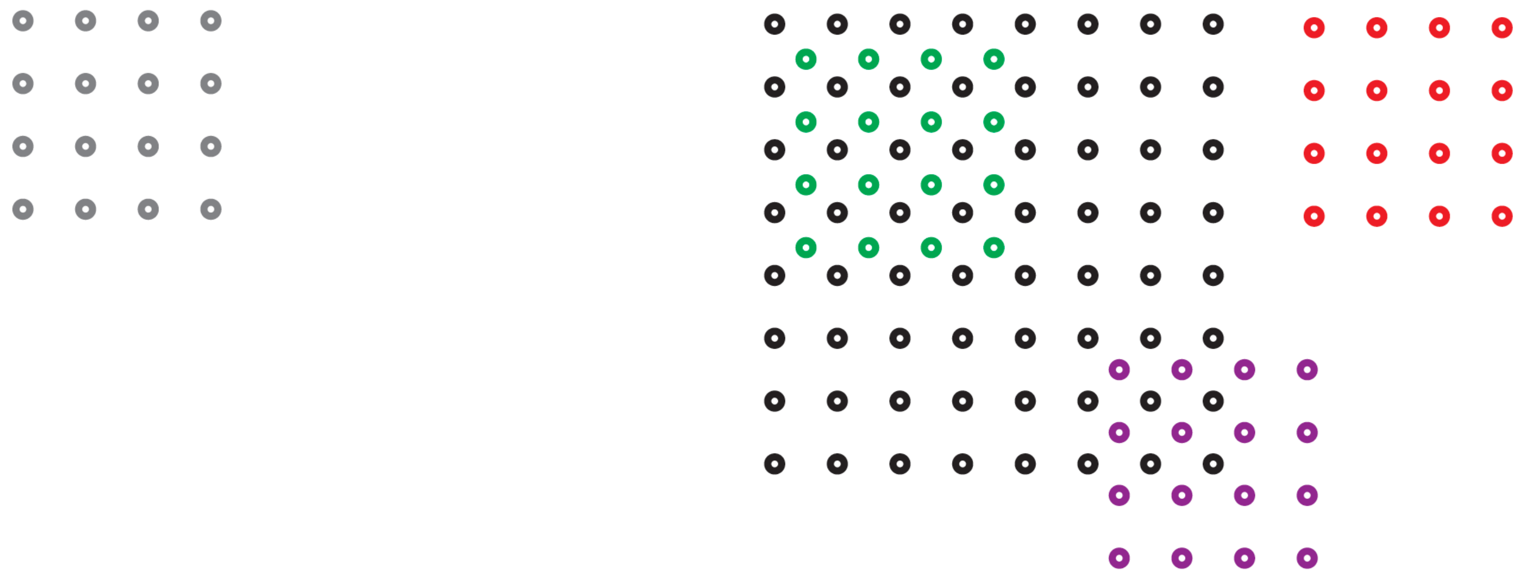


Transformations always take a source image $\mathcal{S}$ which is $s_M \times s_N$ to a target image $\mathcal{T}$ which is $t_M \times t_N$. I will need to refer to image values both at integer points – which I will write $\mathcal{S}_{ij}$ – and at points that are possibly not integer points – $\mathcal{S}(x, y)$. For points that are not integer points, care is required. If $1 \leq x \leq s_M$ and $1 \leq y \leq s_N$, then $\mathcal{S}(x, y)$ can be obtained by interpolation. Otherwise, some care is required.



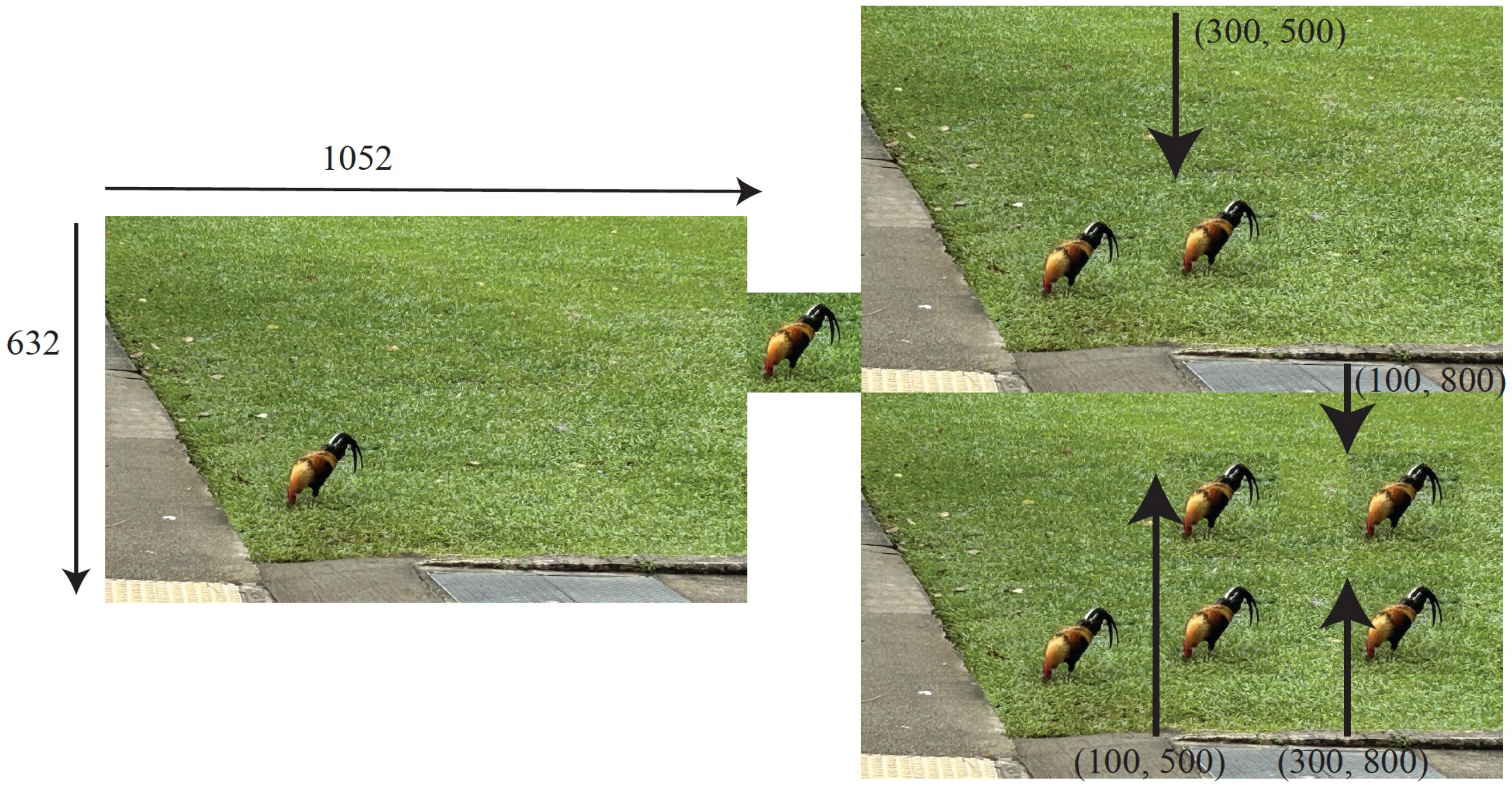
As in Section 3.1, the correct general procedure is to scan the pixels of $\mathcal{T}$ and then modify them using interpolates of pixels from $\mathcal{S}$. This means it is important that transformations are invertible, and both $(u(x, y), v(x, y))$ and $(x(u, v), y(u, v))$ are known. If you require that the value of $\mathcal{S}(x, y)$ is known if $1 \leq x \leq s_M$ and $1 \leq y \leq s_N$, the image might shrink when you translate it. Figure 4.2 illustrates this effect. The source image has been translated to the green location. If you scan the target image (the bigger grid), and report a known value for $\mathcal{S}(x, y)$ only if $1 \leq x \leq s_M$ and $1 \leq y \leq s_N$, you will lose pixels (**exercises**).

Losing pixels when inverse warping



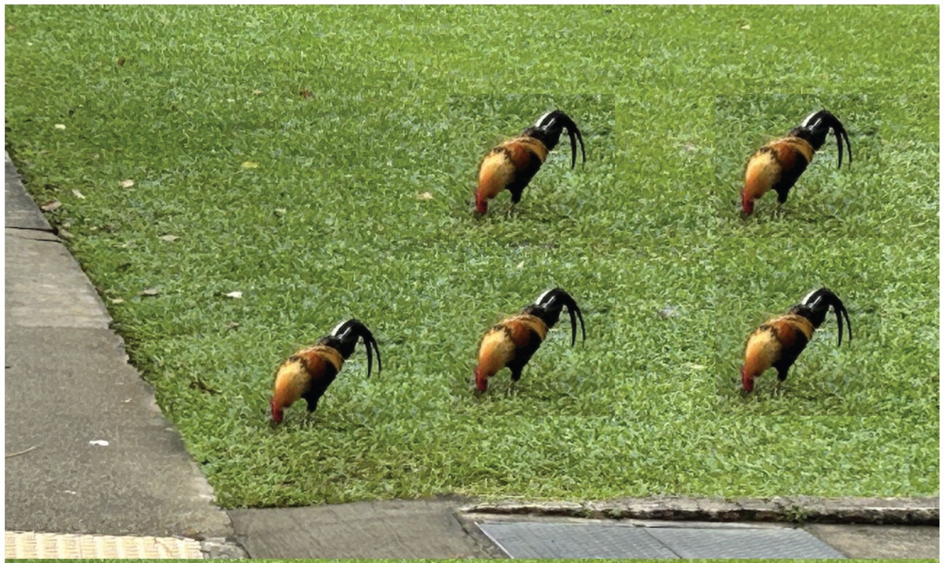
You could mitigate this effect by *padding* the source image so that you know pixel values for $0 \leq x \leq s_M$ and $0 \leq y \leq s_N$. An easy way to do this is to attach a copy of the top row to the top of the image, and the leftmost column to the left of the image. More sophisticated mitigations are out of scope.

Cropping, translating and pasting

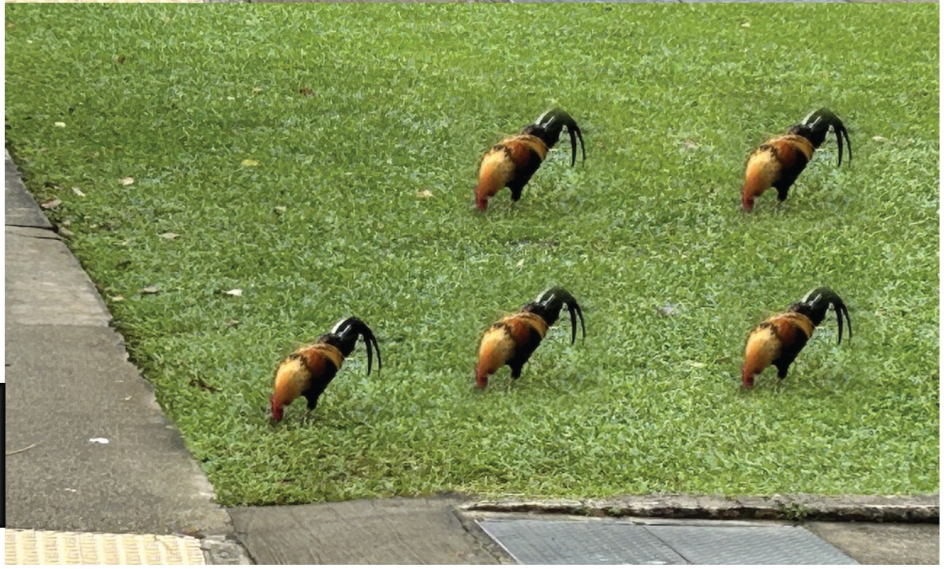


Blending

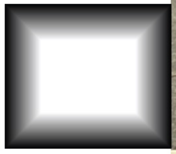
- Can improve over pasting



Blended



Mask



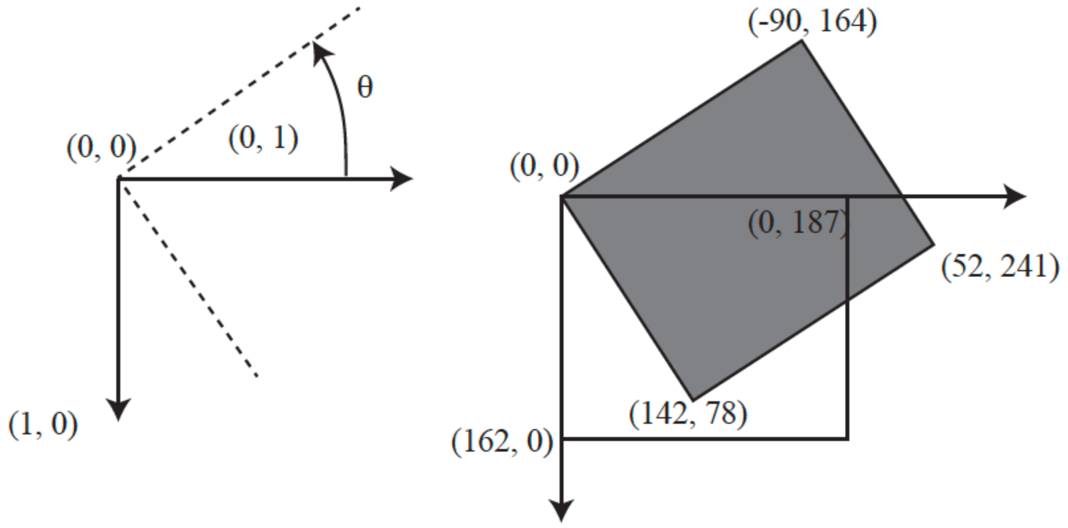
Scaling

- Uniform scaling
  - we did that!
    - upsampling – scale by k>1
    - downsampling – scale by k<1
- Non uniform scaling
  - can be a nuisance
    - upsampling in one direction, downsampling in the other!

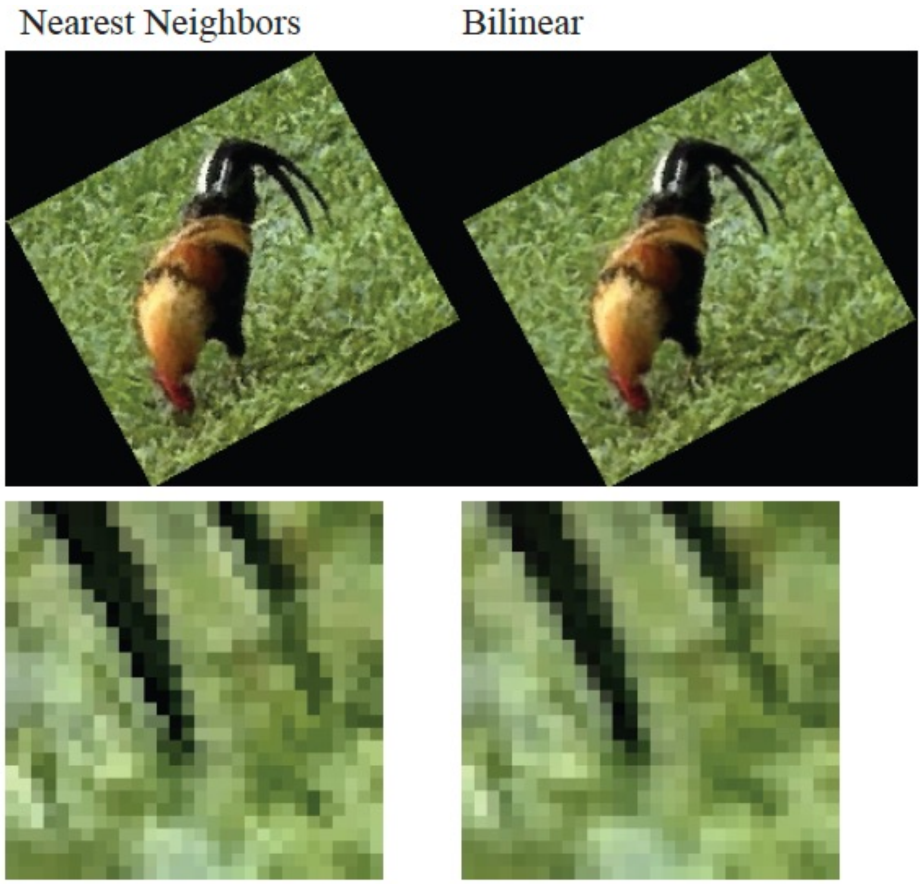
Rotation

- Rotating about the origin can lead to trouble
  - pixels leave the span

- Common fix in APIs
  - rotate about the center of the image
  - pixels still leave the span
  - Choices:
    - all (bigger image, lots of zeros)
    - same (crop to original)

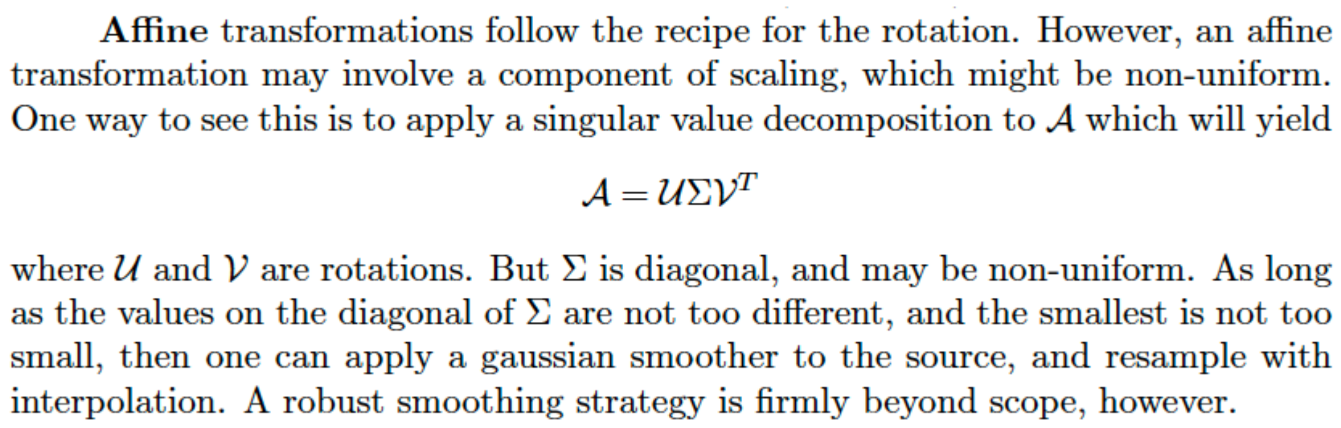


Rotation: Interpolation matters



Affine transformations

- Problems
  - There is often a non-uniform scaling inside the matrix
- Image could end up outside target image range
  - what to do?
  - usually, figure out target image range from affine tx.

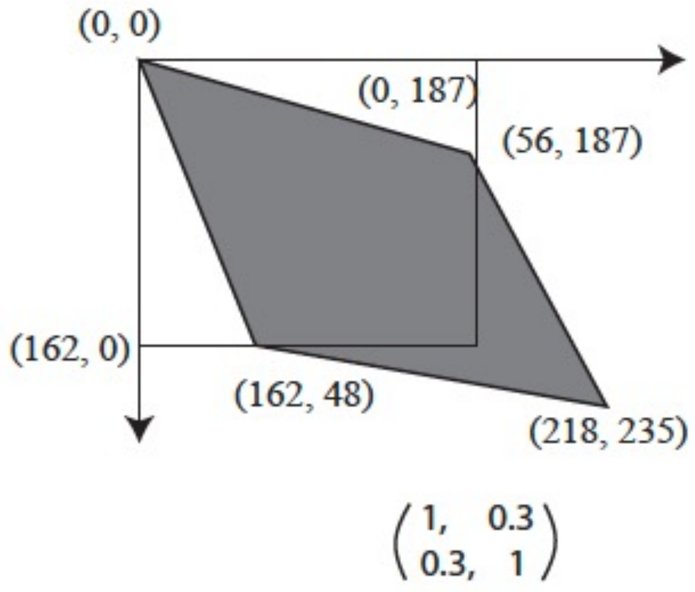


Affine transformations follow the recipe for the rotation. However, an affine transformation may involve a component of scaling, which might be non-uniform. One way to see this is to apply a singular value decomposition to $\mathcal{A}$ which will yield

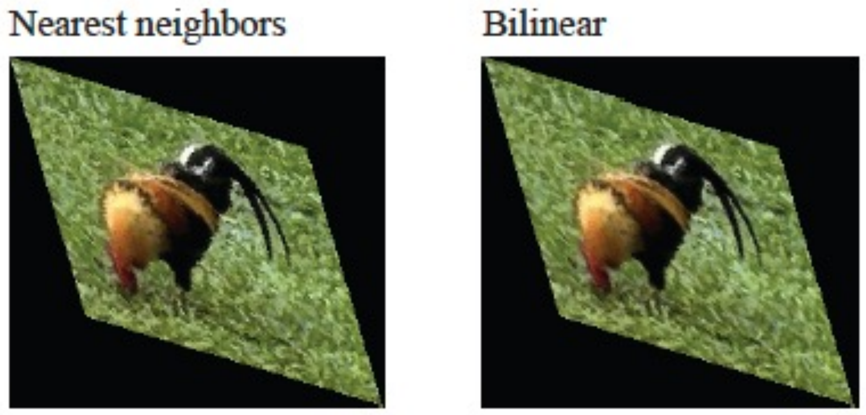
$$\mathcal{A} = \mathcal{U}\Sigma\mathcal{V}^T$$

where $\mathcal{U}$ and $\mathcal{V}$ are rotations. But Σ is diagonal, and may be non-uniform. As long as the values on the diagonal of Σ are not too different, and the smallest is not too small, then one can apply a gaussian smoother to the source, and resample with interpolation. A robust smoothing strategy is firmly beyond scope, however.

Affine transformation

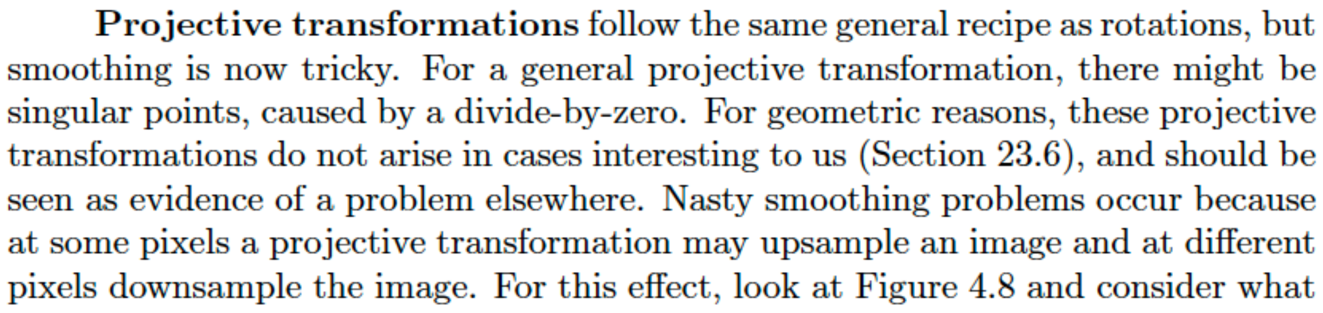


$$\begin{pmatrix} 1, & 0.3 \\ 0.3, & 1 \end{pmatrix}$$



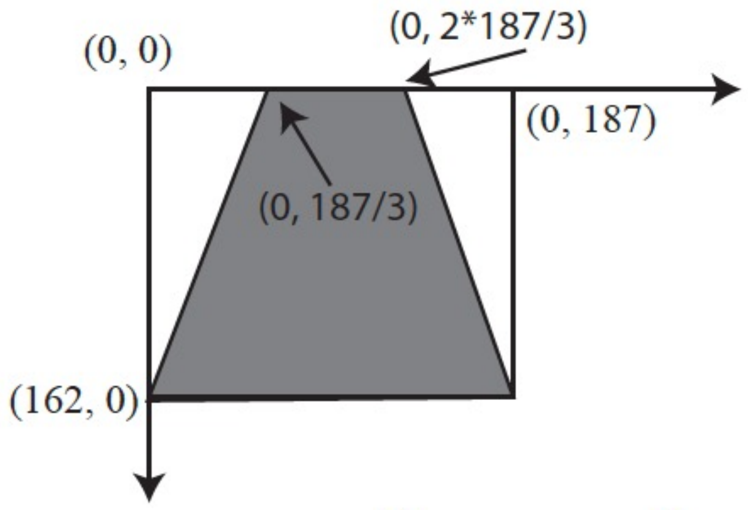
Projective transformations

- Problems
  - There is often a non-uniform scaling inside the matrix
  - Possible divide by zero
- Image could end up outside target image range
  - what to do?
  - usually, figure out target image range from affine tx.



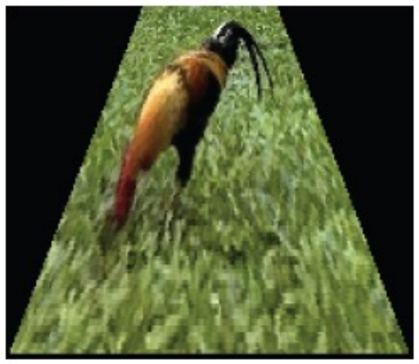
Projective transformations follow the same general recipe as rotations, but smoothing is now tricky. For a general projective transformation, there might be singular points, caused by a divide-by-zero. For geometric reasons, these projective transformations do not arise in cases interesting to us (Section 23.6), and should be seen as evidence of a problem elsewhere. Nasty smoothing problems occur because at some pixels a projective transformation may upsample an image and at different pixels downsample the image. For this effect, look at Figure 4.8 and consider what

Projective transformation

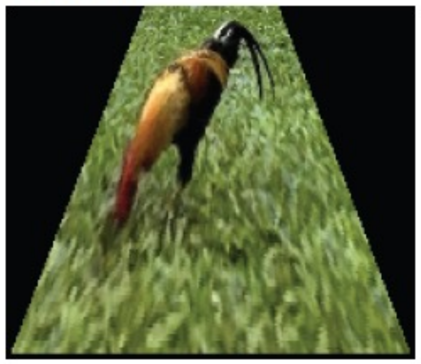


$$\begin{pmatrix} 1/3, & 0, & 0 \\ -187/(3*162), & 1/3, & 187/3 \\ -2/(3*162), & 0, & 1 \end{pmatrix}$$

Nearest neighbors

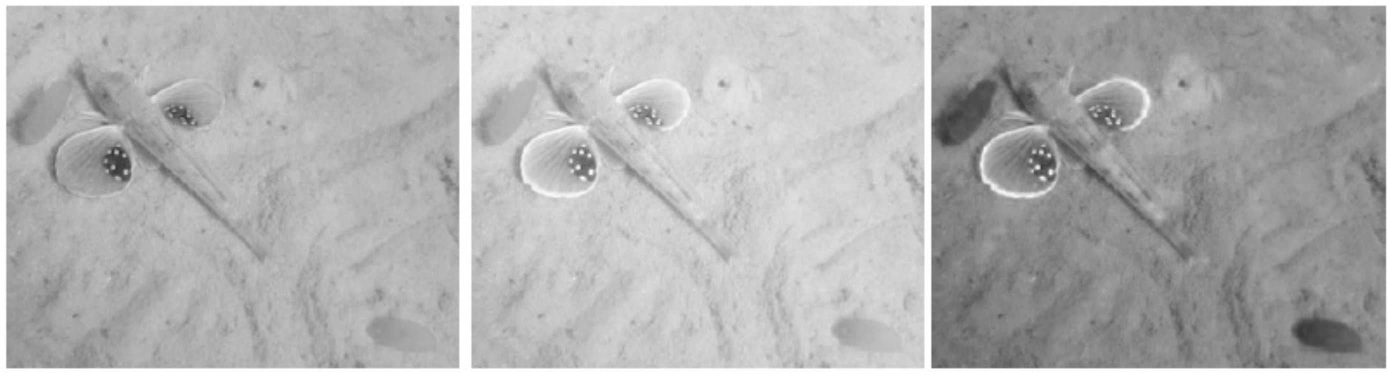


Bilinear



Simple registration with translation

- Color separations:

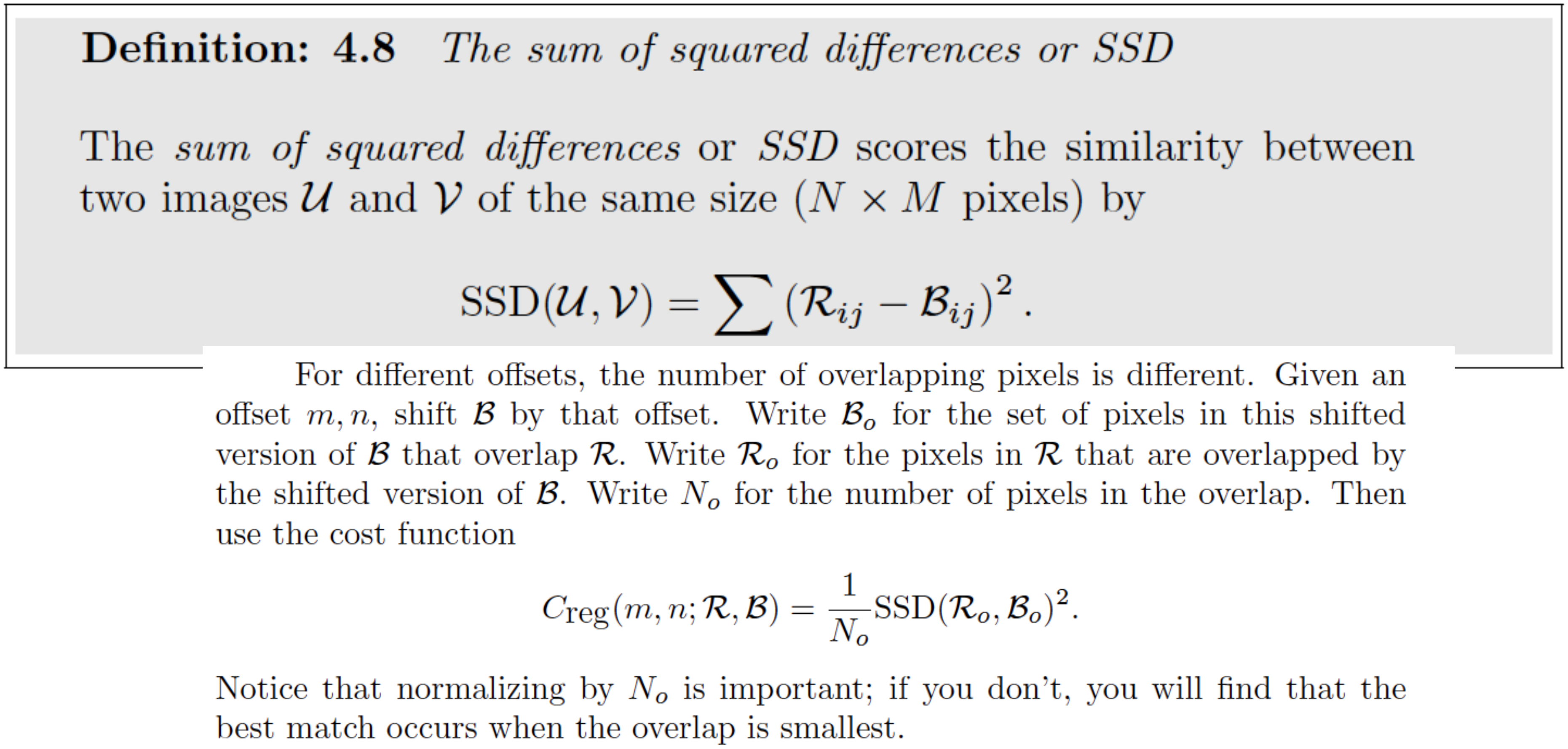


Simple registration with translation

- Fix G in place
    - for many (tx, ty)
        - slide R by tx, ty
        - compute C(tx, ty)
            - which compares overlaps
    - take the (tx, ty) that gives best overlap

- Q:
    - what is cost?
    - other applications?
    - efficiency?

Cost functions: SSD



Definition: 4.8 *The sum of squared differences or SSD*

The *sum of squared differences* or *SSD* scores the similarity between two images $\mathcal{U}$ and $\mathcal{V}$ of the same size ($N \times M$ pixels) by

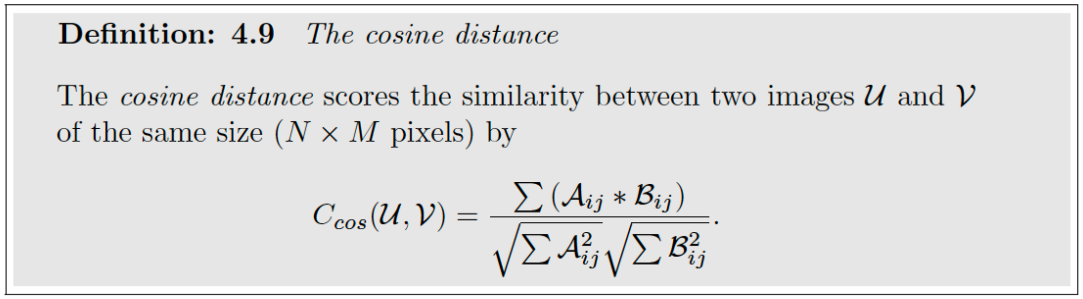
$$\mathrm{SSD}(\mathcal{U}, \mathcal{V}) = \sum (\mathcal{R}_{ij} - \mathcal{B}_{ij})^2 .$$

For different offsets, the number of overlapping pixels is different. Given an offset m, n, shift $\mathcal{B}$ by that offset. Write $\mathcal{B}_o$ for the set of pixels in this shifted version of $\mathcal{B}$ that overlap $\mathcal{R}$. Write $\mathcal{R}_o$ for the pixels in $\mathcal{R}$ that are overlapped by the shifted version of $\mathcal{B}$. Write N_o for the number of pixels in the overlap. Then use the cost function

$$C_{\mathrm{reg}}(m, n; \mathcal{R}, \mathcal{B}) = \frac{1}{N_o}\mathrm{SSD}(\mathcal{R}_o, \mathcal{B}_o)^2.$$

Notice that normalizing by N_o is important; if you don't, you will find that the best match occurs when the overlap is smallest.

Cost functions: cosine dist and correlation

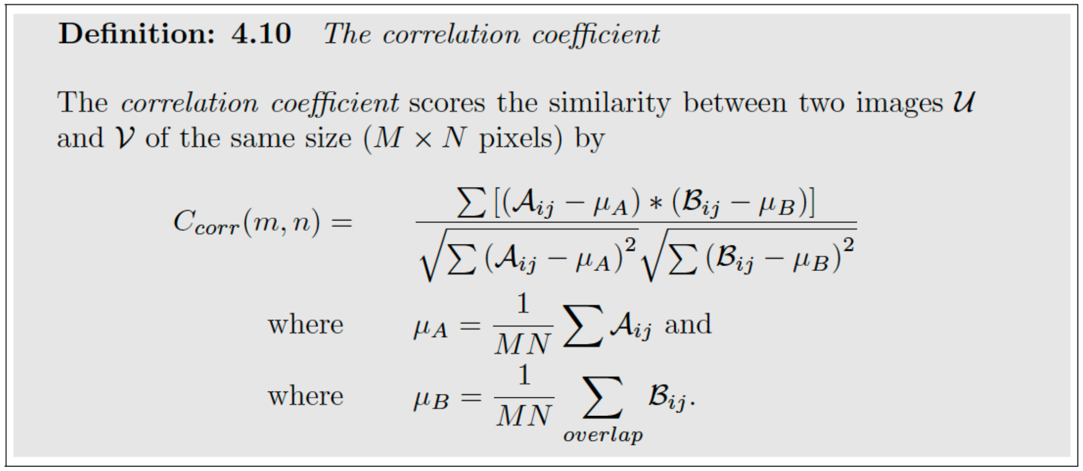


Definition: 4.9 *The cosine distance*

The *cosine distance* scores the similarity between two images $\mathcal{U}$ and $\mathcal{V}$ of the same size ($N \times M$ pixels) by

$$C_{cos}(\mathcal{U},\mathcal{V}) = \frac{\sum(\mathcal{A}_{ij} * \mathcal{B}_{ij})}{\sqrt{\sum \mathcal{A}_{ij}^2}\sqrt{\sum \mathcal{B}_{ij}^2}}.$$

You compute these for the overlap



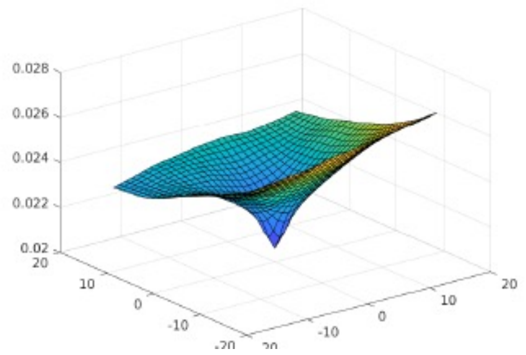
Definition: 4.10 *The correlation coefficient*

The *correlation coefficient* scores the similarity between two images $\mathcal{U}$ and $\mathcal{V}$ of the same size ($M \times N$ pixels) by

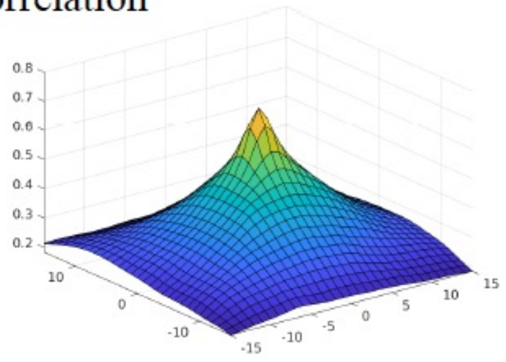
$$\begin{aligned} C_{corr}(m,n) &= \frac{\sum[(\mathcal{A}_{ij} - \mu_A) * (\mathcal{B}_{ij} - \mu_B)]}{\sqrt{\sum(\mathcal{A}_{ij} - \mu_A)^2}\sqrt{\sum(\mathcal{B}_{ij} - \mu_B)^2}} \\ \text{where} \quad \mu_A &= \frac{1}{MN}\sum \mathcal{A}_{ij} \text{ and} \\ \text{where} \quad \mu_B &= \frac{1}{MN}\sum_{overlap} \mathcal{B}_{ij}. \end{aligned}$$

Cost functions

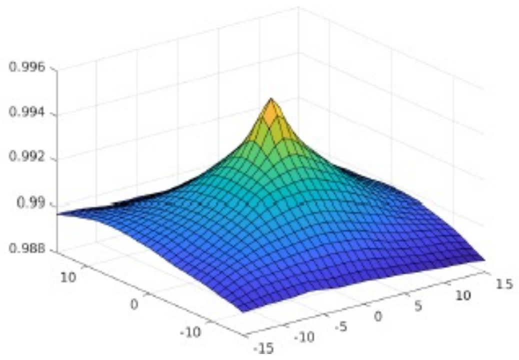
Squared error



Correlation

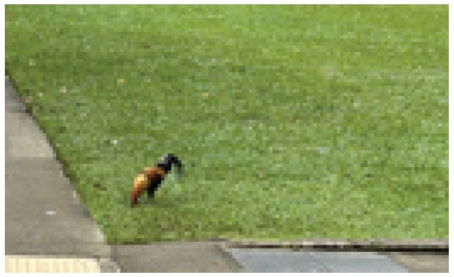


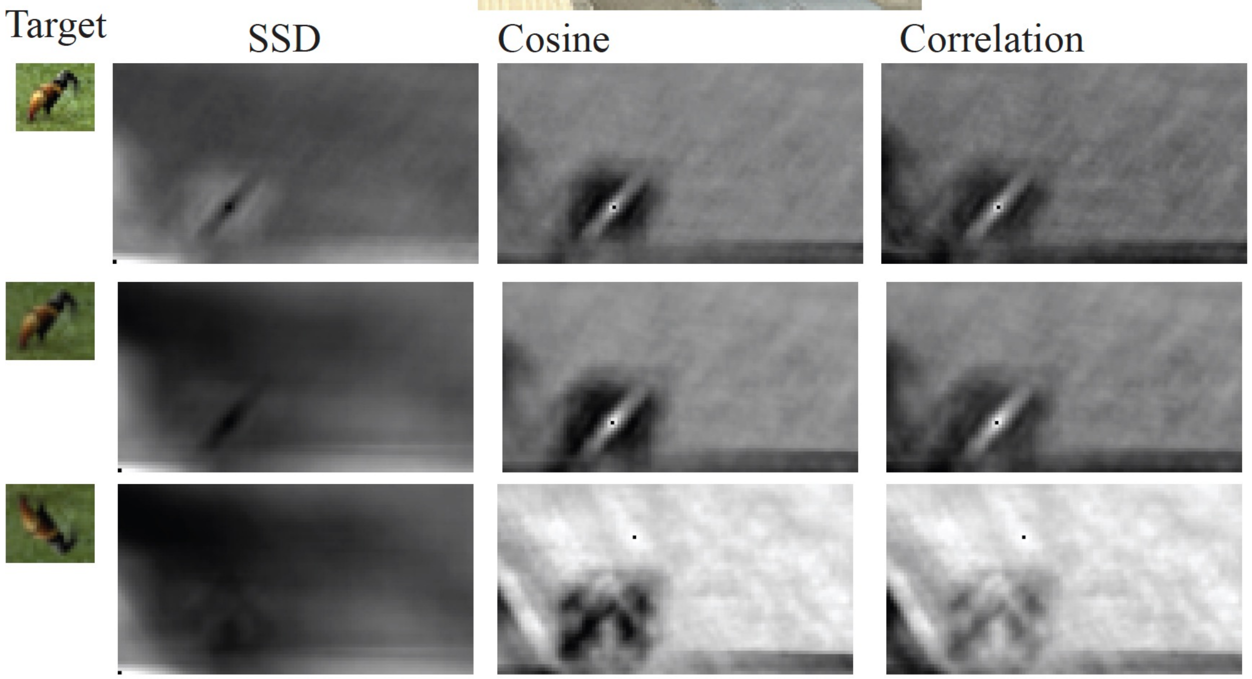
Cosine distance



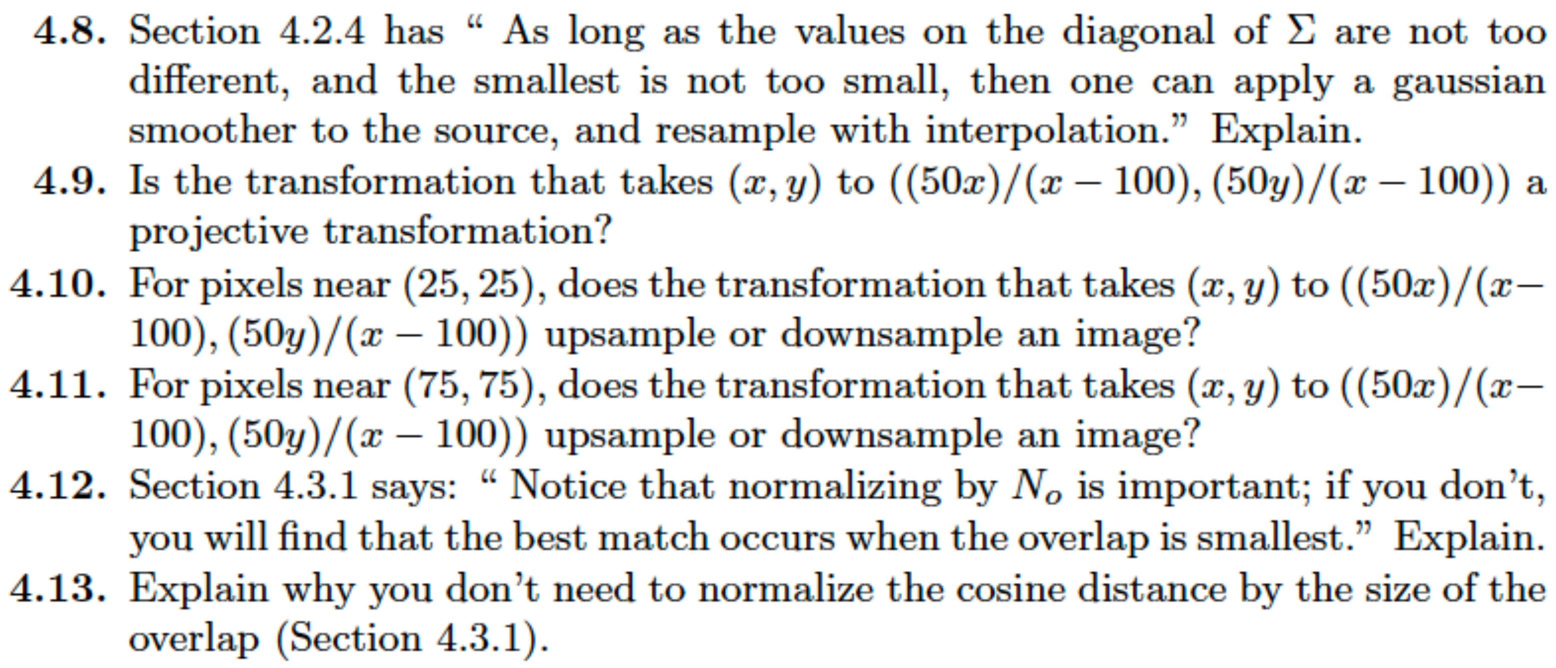
Find the chicken

- Simplest object detection
  - not very good
  - can be fast





Think about this...



4.8. Section 4.2.4 has " As long as the values on the diagonal of Σ are not too different, and the smallest is not too small, then one can apply a gaussian smoother to the source, and resample with interpolation." Explain.

4.9. Is the transformation that takes (x, y) to $((50x)/(x-100), (50y)/(x-100))$ a projective transformation?

4.10. For pixels near $(25, 25)$, does the transformation that takes (x, y) to $((50x)/(x-100), (50y)/(x-100))$ upsample or downsample an image?

4.11. For pixels near $(75, 75)$, does the transformation that takes (x, y) to $((50x)/(x-100), (50y)/(x-100))$ upsample or downsample an image?

4.12. Section 4.3.1 says: " Notice that normalizing by N_o is important; if you don't, you will find that the best match occurs when the overlap is smallest." Explain.

4.13. Explain why you don't need to normalize the cosine distance by the size of the overlap (Section 4.3.1).