CPSC 4040/6040 Computer Graphics Images

Joshua Levine
levinej@clemson.edu

Lecture 17
Warping

Oct. 20, 2015

Agenda

- PA05 Question?
- Q03 Due Thurs
- Project Proposals Due 10/29

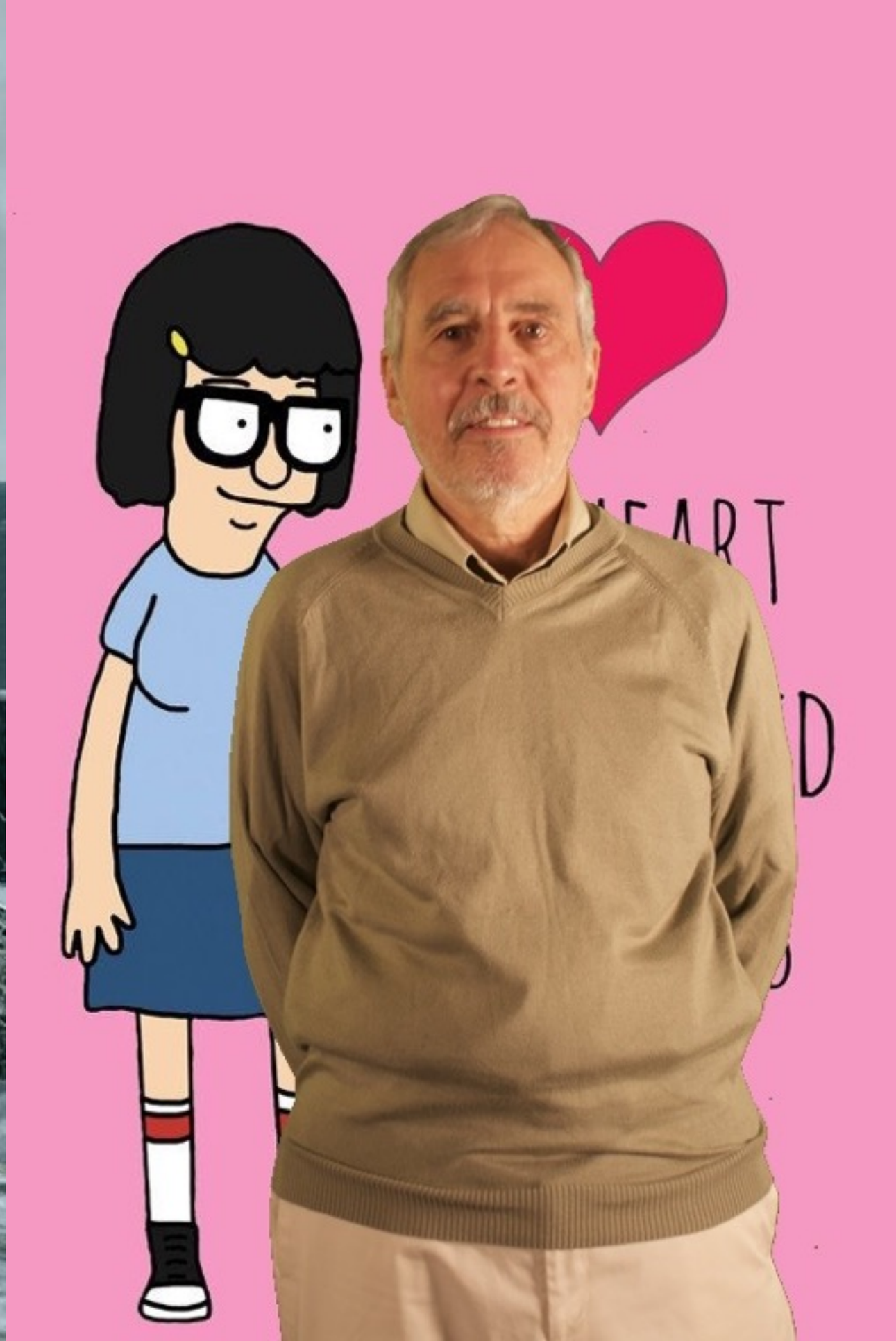
PA03 Highlights

Areas Where The Class, In General, Could Improve

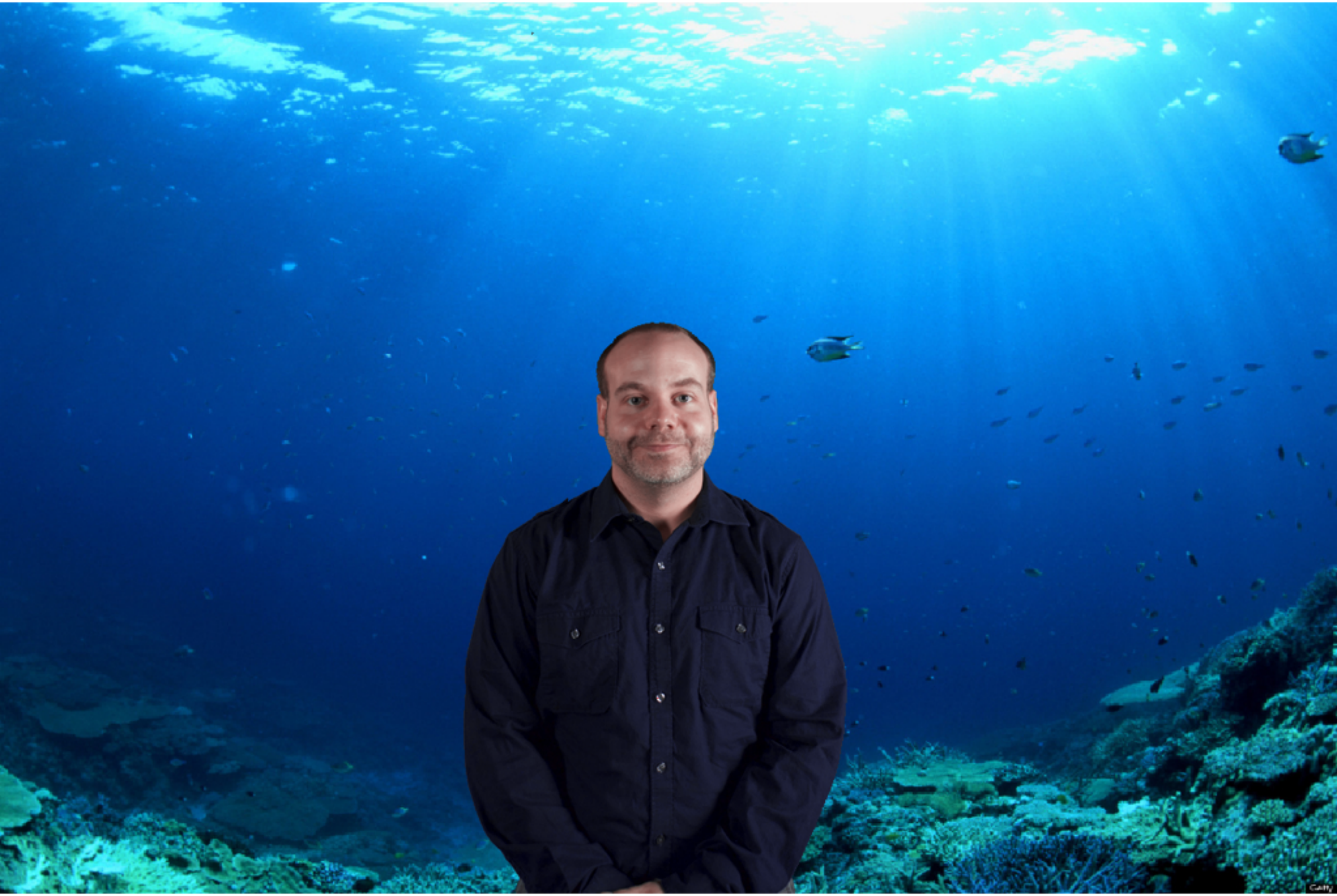
- Code structure: comments, indentation, readability
- README Structure:
    1. Separate onto multiple lines
    2. Give CLEAR instructions on how to COMPILE and EXECUTE
    3. A brief overview of functionalities, parameters
    4. An overview of the user interface, keyboard/mouse commands
- It would be helpful if q/ESC exited the program
- Please review academic integrity rules in the syllabus.

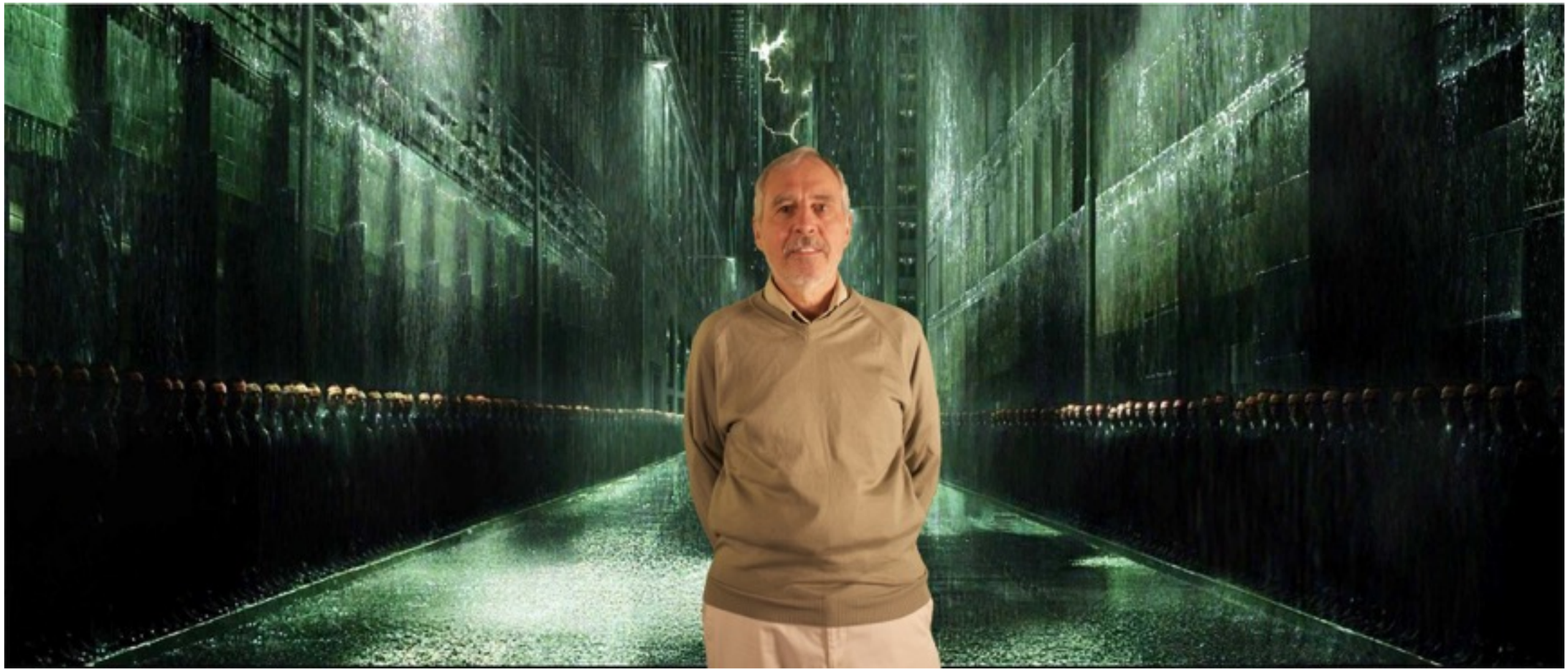
ner, Like Daughter


N




CCTV 1
综 合
19:00:16
新闻联播
XINWEN





PA05 Notes: Color Scaling

- For a color image, try to convert the input (world) luminance Lw to a target display luminance Ld

- This type of scaling works best in the log and/or exponential domains (because of Weber's law)

$$\begin{bmatrix} R_d \\ G_d \\ B_d \end{bmatrix} = \begin{bmatrix} L_d \dfrac{R_w}{L_w} \\ L_d \dfrac{G_w}{L_w} \\ L_d \dfrac{B_w}{L_w} \end{bmatrix}$$

PA05 Notes: Working in the Log Domain

- Recall: humans are sensitive to multiplicative contrast
  - This is what gamma correction works!

- In the log-domain, gamma correction is a multiplication:
  - $L_d = L_w^{\gamma}$
  - $\Rightarrow \log(L_d) = \log(L_w^{\gamma})$
  - $\Rightarrow \log(L_d) = \gamma \log(L_w)$

- Do this in your lab, but be careful if $L_w = 0$
  - In practice, if $L_w == 0$, add a tiny number (e.g. FLT_MIN)

PA05 Notes: Log Domain Review

Rule 1: $\log_b(M \cdot N) = \log_b M + \log_b N$

Rule 2: $\log_b\left(\frac{M}{N}\right) = \log_b M - \log_b N$

Rule 3: $\log_b\left(M^k\right) = k \cdot \log_b M$

Rule 4: $\log_b(1) = 0$

Rule 5: $\log_b(b) = 1$

Rule 6: $\log_b\left(b^k\right) = k$

Rule 7: $b^{\log_b(k)} = k$

Where : $b > 1$, and M, N and k can be any real numbers

but M and N must be positive!

http://www.chilimath.com/algebra/advanced/log/log-expanding.html

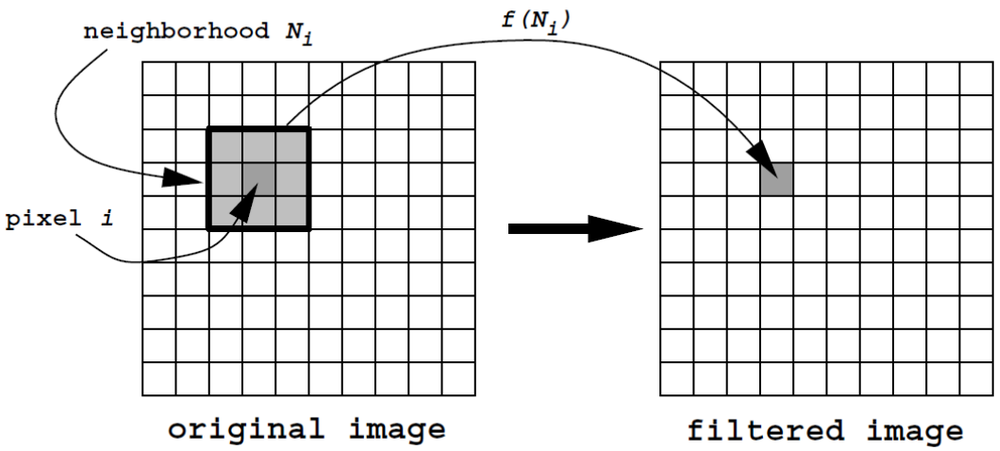
Warping

Warping vs. Filtering

- **Filters** modify the color **values** stored at pixel locations.
- **Warps** modify the **location** of each pixel's color values.

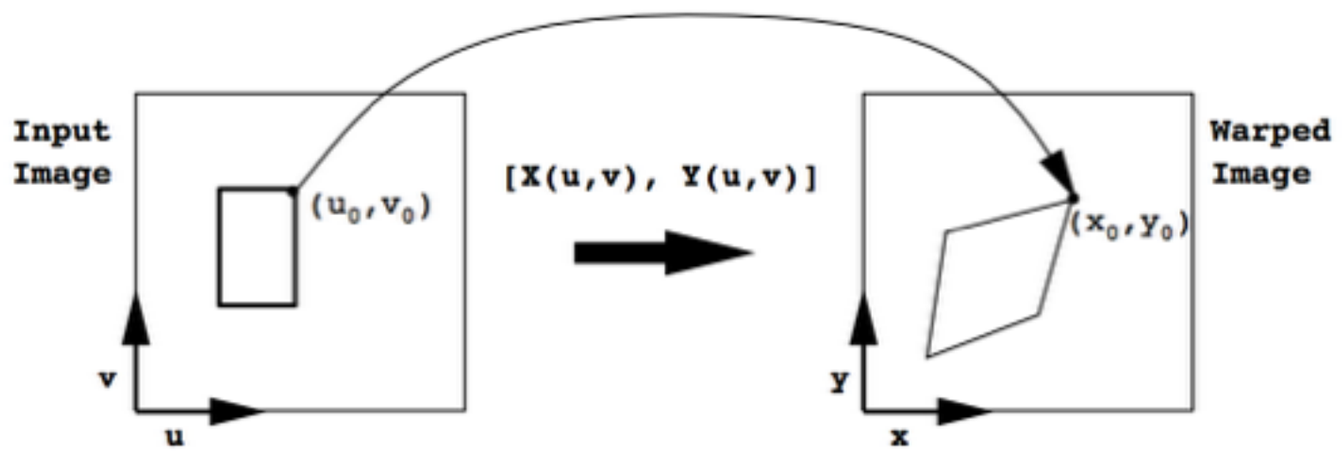
Filters

$$\hat{C}_i = f(N_i)$$



Warps

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X(u,v) \\ Y(u,v) \end{bmatrix}$$



Warps are Domain Transformations

- Apply a function f from R^2 to R^2 within the image domain
- If pixel at (u,v) had color c in the input, then pixel at (x,y) = f(u,v) will have color c in the output.
  - Idea: f takes a pixel coordinate and returns a pixel coordinate
- Compare: Filters transform the range of an image, not the domain.
  - Filters take input pixel coordinates and return color values.

Types of Transformations

- Simple parametric transformations
  - linear, affine, perspective, etc.

translation

rotation

aspect

affine

perspective

cylindrical

illustration by Rick Szeliski

Warping

- Imagine your image is made of rubber
  - Stretching the rubber == warps the image

No prairie dogs were armed when creating this image

Application of Warping: Weight Loss

- In Photoshop (e.g. liquify)

figure 9.35

figure 9.36

figure 9.37

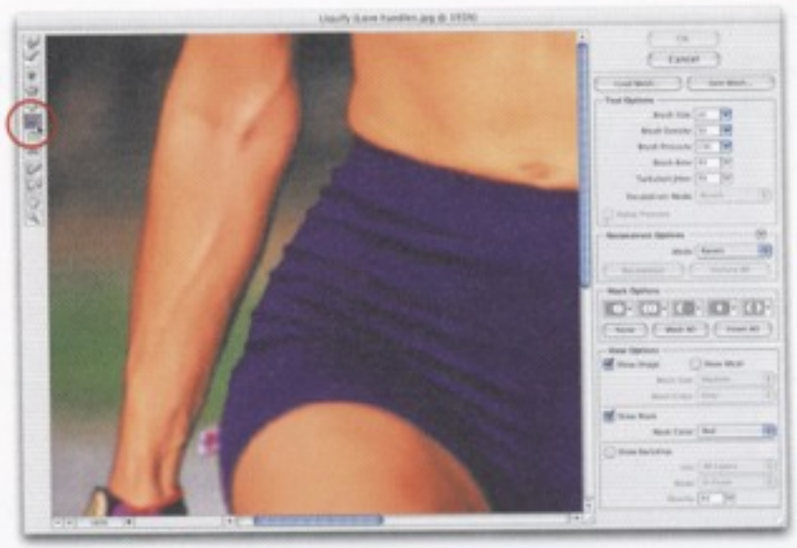
Selecting the entire left side of the image avoids potential artifacts.

figure 9.38

Dragging a Free Transform handle to narrow the selected area.

figure 9.39

The Liquify filter's Warp tool pushes pixels forward as you drag.

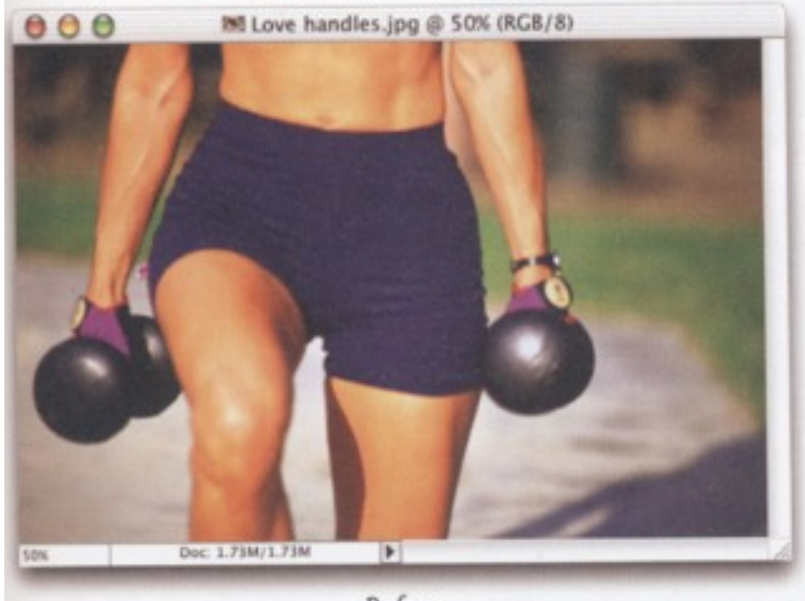


Step Three:

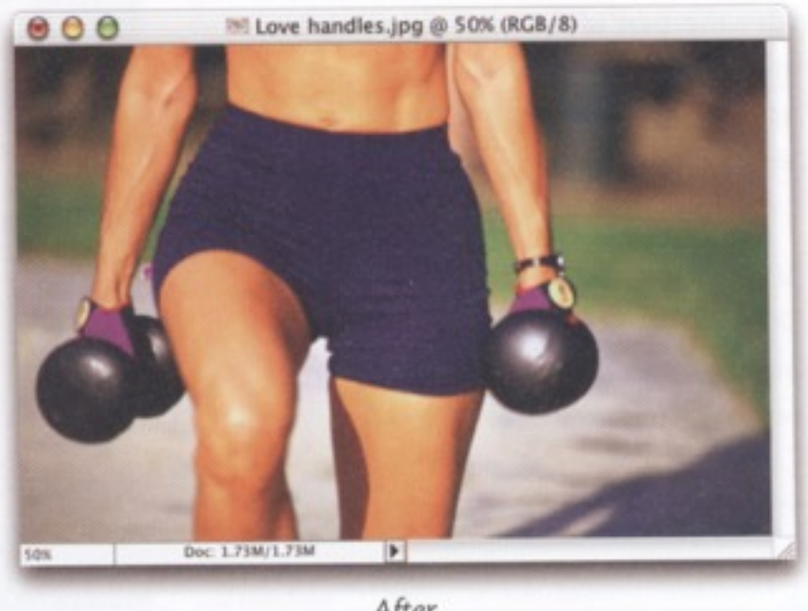
Get the Push Left tool from the Toolbar (as shown here). It was called the Shift Pixels tool in Photoshop 6 and 7, but Adobe realized that you were getting used to the name, so they changed it, just to keep you off balance.

Step Four:

Choose a relatively small brush size (like the one shown here) using the Brush Size field near the top-right of the Liquify dialog. With it, paint a downward stroke starting just above and outside the love handle and continuing downward. The pixels shifts back in toward the body, removing the love handle as you paint. (Note: If you need to remove love handles on the left side of the body, paint upward rather than downward. Why? That's just the way it works.) When you click OK, the love handle repair is complete.



Before.



After.

Digital Image Warping

Issues with Warping

- Apply a function f from R^2 to R^2 to the image
  - Looks easy enough?
- Big issues:
  - How do we deal with non-integer coordinates?
  - How big is the output image?
  - Which direction do we transform?
- Solutions will be based on how we compute and specify f.

Transformations

- The mapping between (u,v) and (x,y) can be generalized as two equations where both X and Y are transformation functions that produce output coordinates based on both of the u and v coordinates of the input pixel.

- Both functions produce real values as opposed to integer coordinates and are assumed to be well defined at all locations in the image plane R^2.

- Function X outputs the horizontal coordinate of the output pixel while function Y outputs the vertical coordinate:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X(u,v) \\ Y(u,v) \end{bmatrix}$$

Destination Image Size

- The destination image may not be large enough to contain all of the processed samples

- Consider a source image that is rotated about the origin.
  - Both the width and height of the destination image must be increased beyond that of the source.
  - Can determine the destination dimensions by transforming the input image bounds.

(a) Source image. (b) Rotation. (c) Expanded view of rotated image.

Figure 7.2. Destination dimensionality under rotation.

Destination Image Size

- Can determine the destination dimensions by transforming the input image bounds.

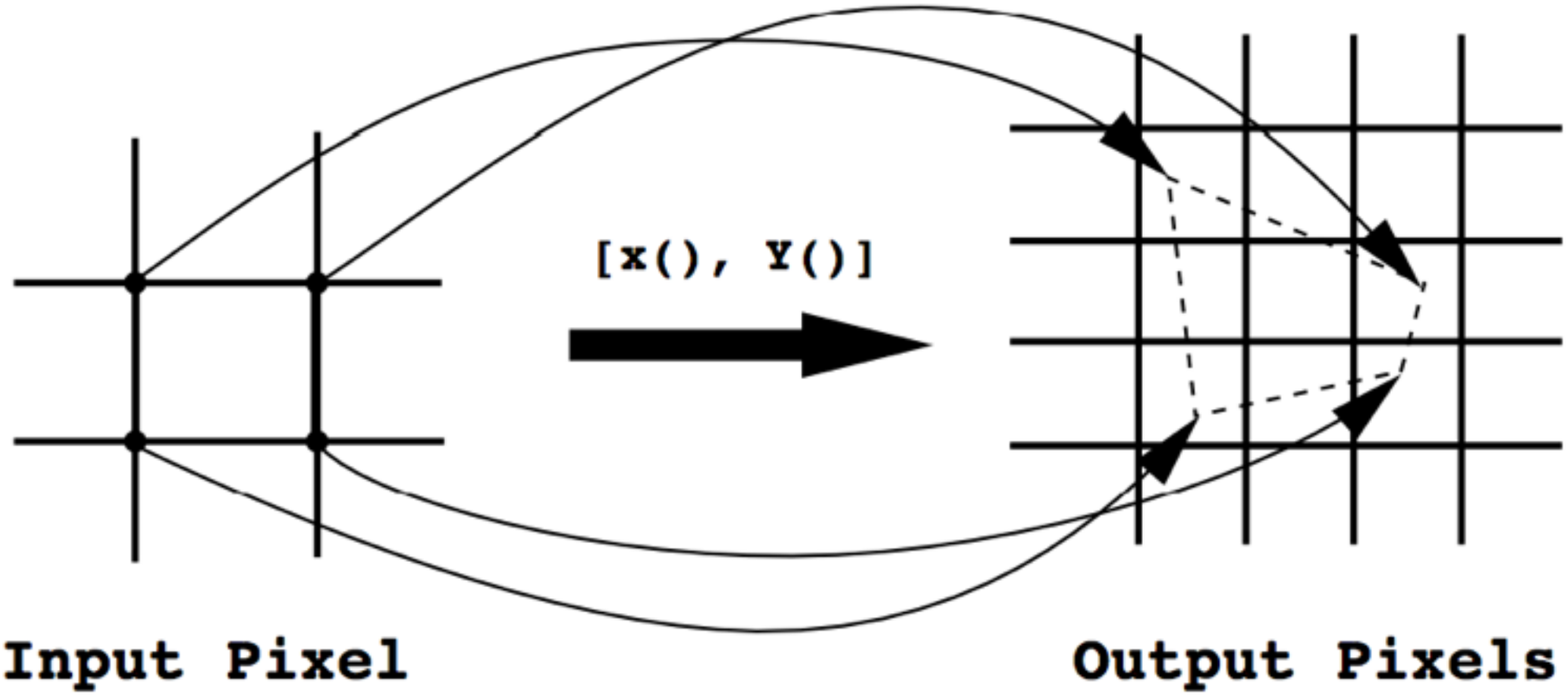
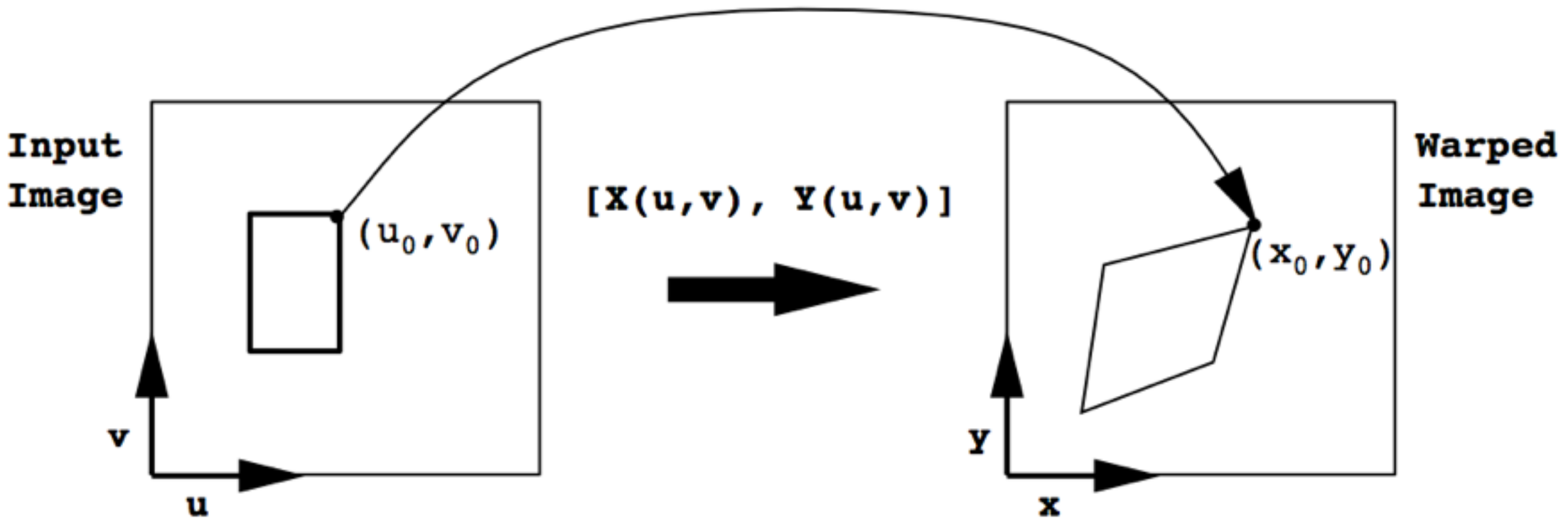


Figure 9.3: Projection of pixel area onto output raster

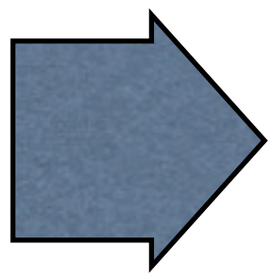
Forward Mapping

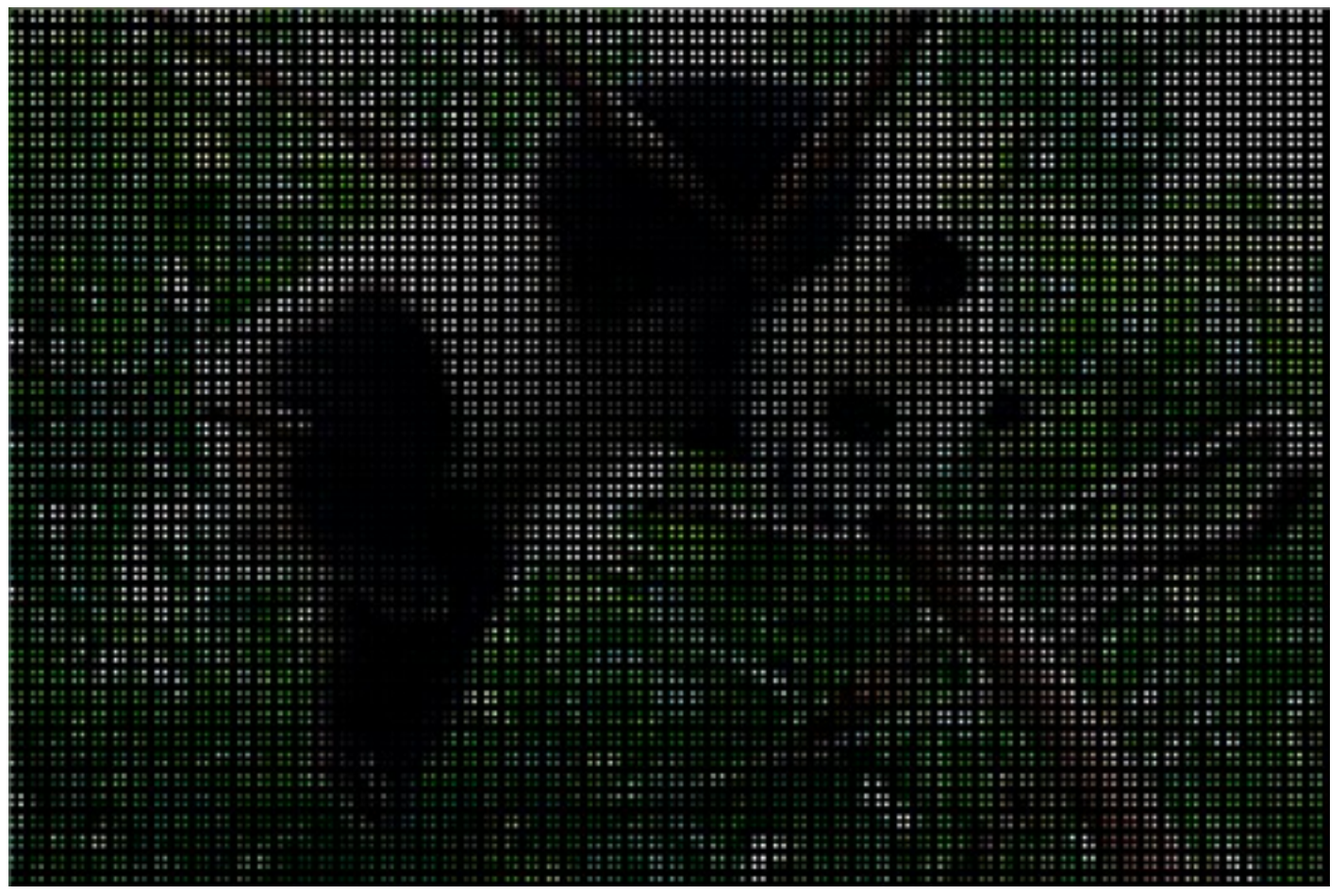



```
for(v = 0; v < in_height; v++)
  for(u = 0; u < in_width; u++)
    Out[round(X(u,v))][round(Y(u,v))] = In[u][v];
```

Recall: Naive Image Scaling

- Consider resizing an image to a large resolution
- Simple approach: Take all the pixels in input and place them in an output location.





Holes and Overlaps

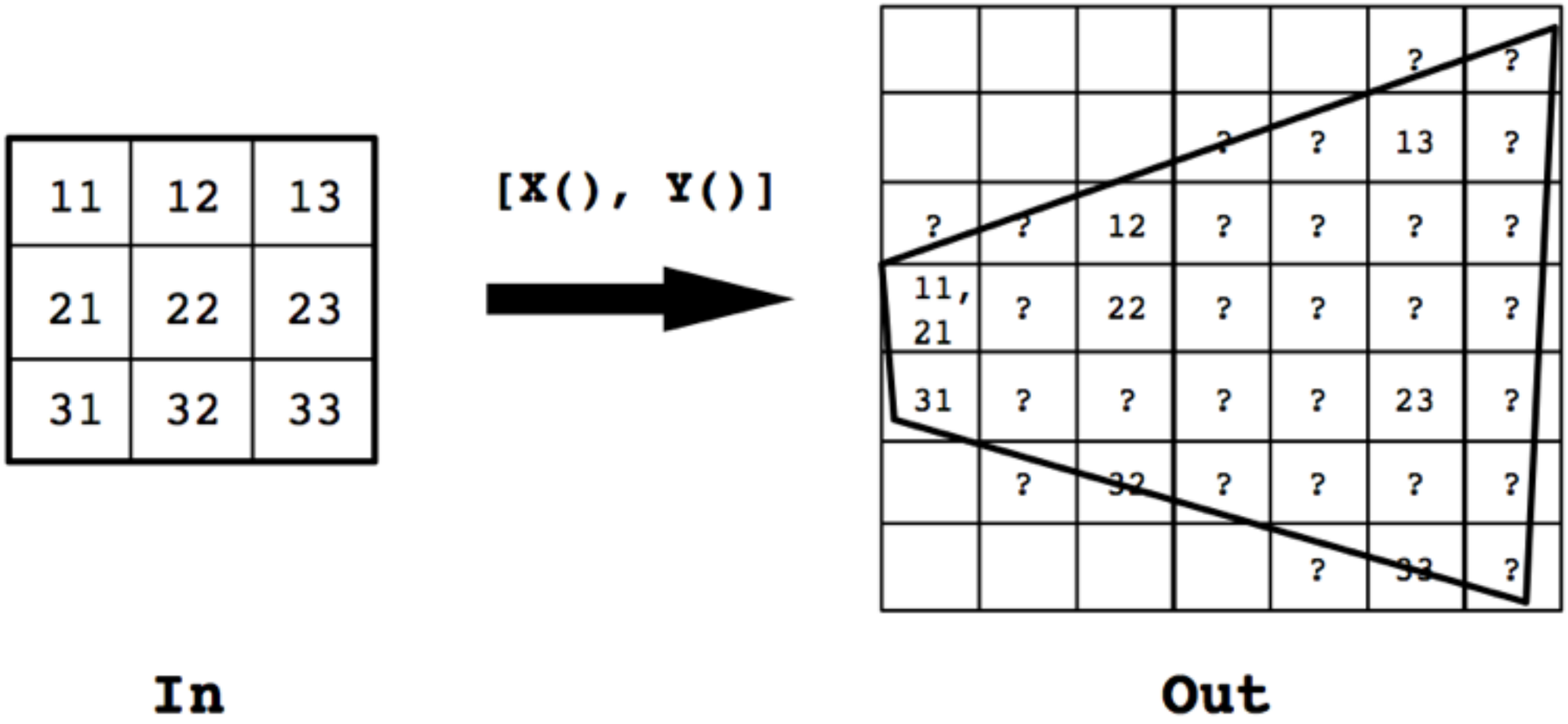


Figure 9.2: Forward Map Leaves Holes and Overlaps

Forward Mapping Rotation Also Causes Artifacts

- The issue of how integer-valued source coordinates are mapped onto integer-valued destination coordinates must also be addressed.
  - Forward mapping takes each pixel of the source image and copies it to a location in the destination by rounding the destination coordinates so that they are integer values.

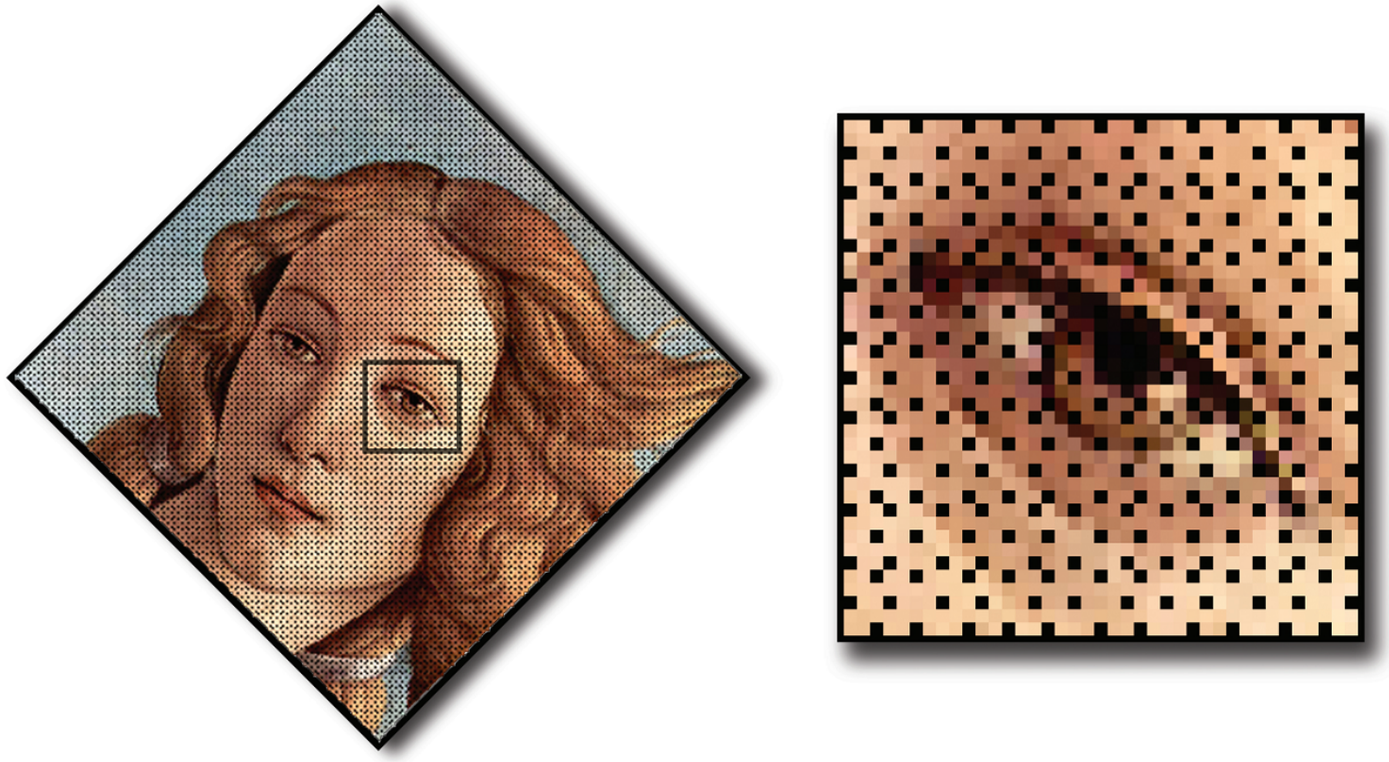


Figure 7.3. Rotation by forward mapping.

Forward Mapping And Gaps

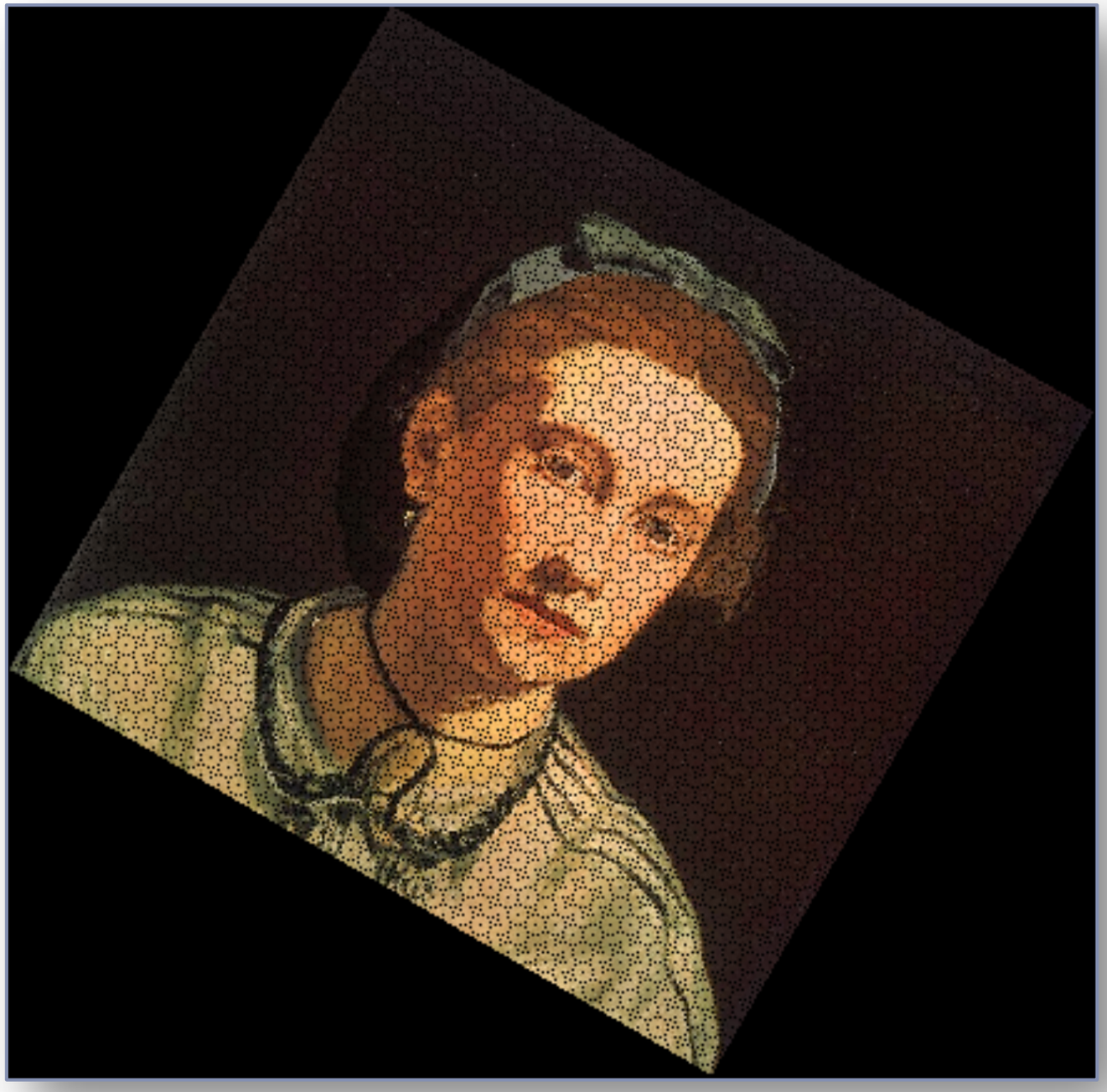


Image Rotation Example

```
algorithm rotate(Image, theta)
   INPUT: an MxN image and angle theta
   OUTPUT: the original image rotated by theta

   Let rotatedImage be an MxN "empty" image
   for every pixel P at location X,Y in the image
      compute rotated X,Y coordinates X',Y'
      place P at X',Y' in rotatedImage

   return rotatedImage
```

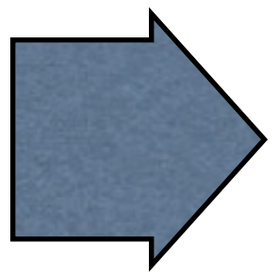
X	Y	X	Y	Round(x, y)	
0	0	0.000	0.000	0	0
1	0	0.866	0.500	1	1
2	0	1.732	1.000	2	1
0	1	-0.500	0.866	0	1
1	1	0.366	1.366	0	1
2	1	1.232	1.866	1	2
0	2	-1.000	1.732	-1	2
1	2	-0.134	2.232	0	2
2	2	0.732	2.732	1	3
0	3	-1.500	2.598	-1	3
1	3	-0.636	3.098	-1	3
2	3	0.232	3.598	0	4

	0	1	2
0	0	10	15
1	20	25	30
2	35	40	45
3	50	55	60

	-1	0	1	2
0		0		
1		20/25	10	15
2	35	40	30	
3	50/55		45	
4		60		

Recall: Using the “Inverse” Transform

- The problem with the previous approach is that we skip many pixels in the output
- We can adjust the loop so that we do one copy for each output pixel instead of each input pixel



Inverse Map

- Forward map operates on input pixel (u,v):

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X(u,v) \\ Y(u,v) \end{bmatrix}$$

- Inverse map operates on output pixels (x,y):

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} U(x,y) \\ V(x,y) \end{bmatrix}$$

Backward Mapping

- When using inverse mapping, the source location (u,v) corresponding to destination (x,y) may not be integer values.
- Determine which pixel you lie in by rounding (alternatives?)

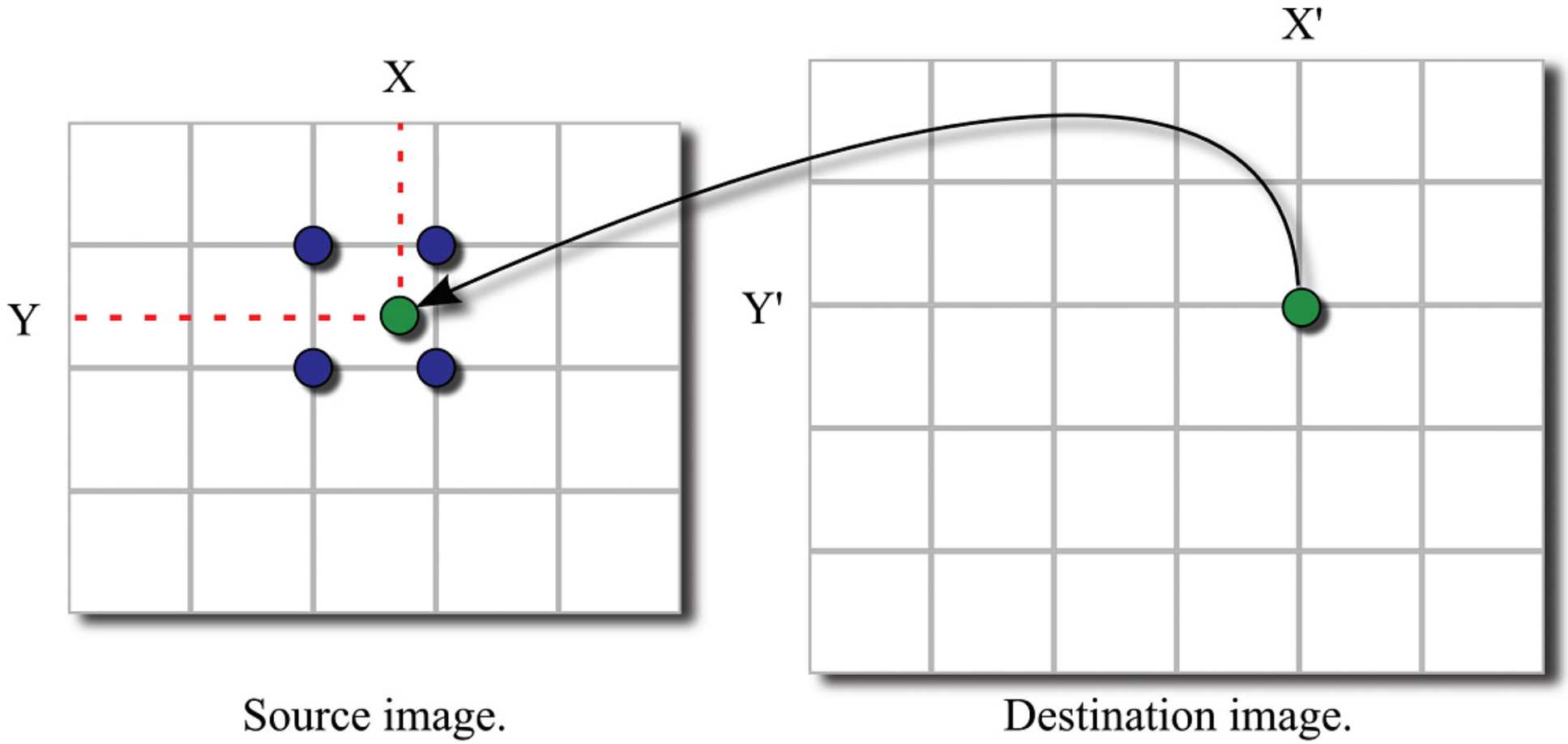


Figure 7.4. Reverse mapping.

Inverse Map Algorithm

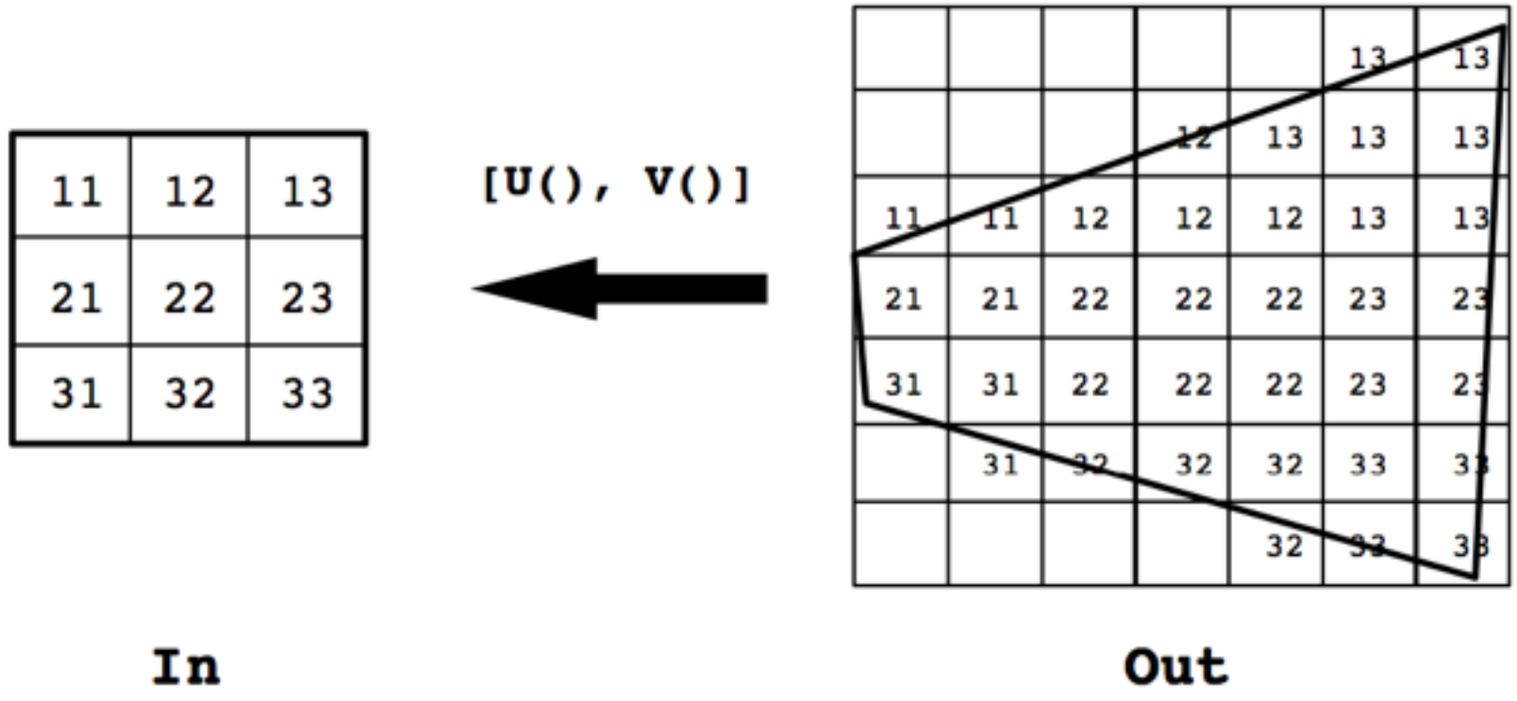


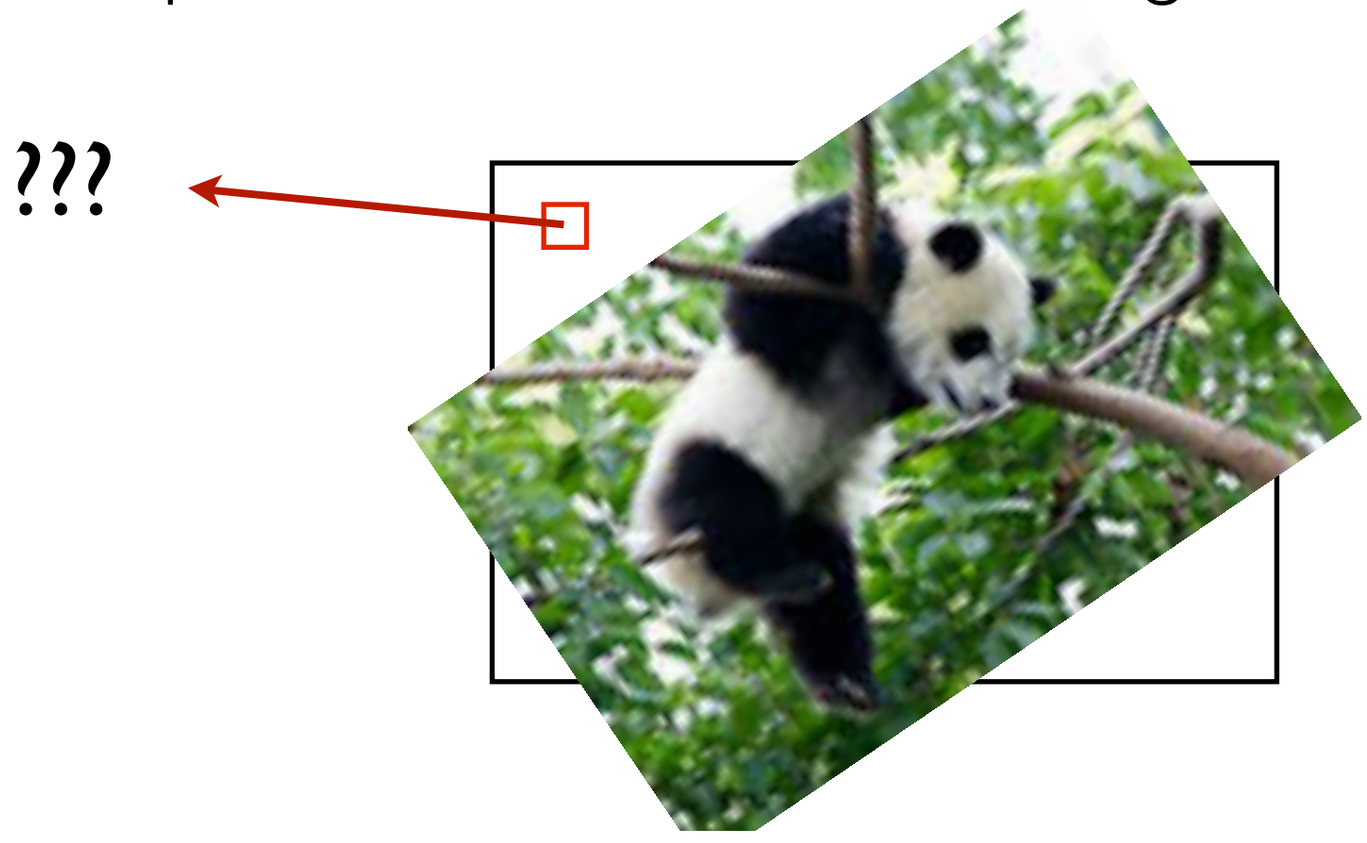
Figure 9.4: Inverse Map gives Complete Covering

```
for(y = 0; y < out_height; y++)
  for(x = 0; x < out_width; x++)
    Out[x][y] = In[round(U(x,y))][round(V(x,y))];
```

Coding Nuances

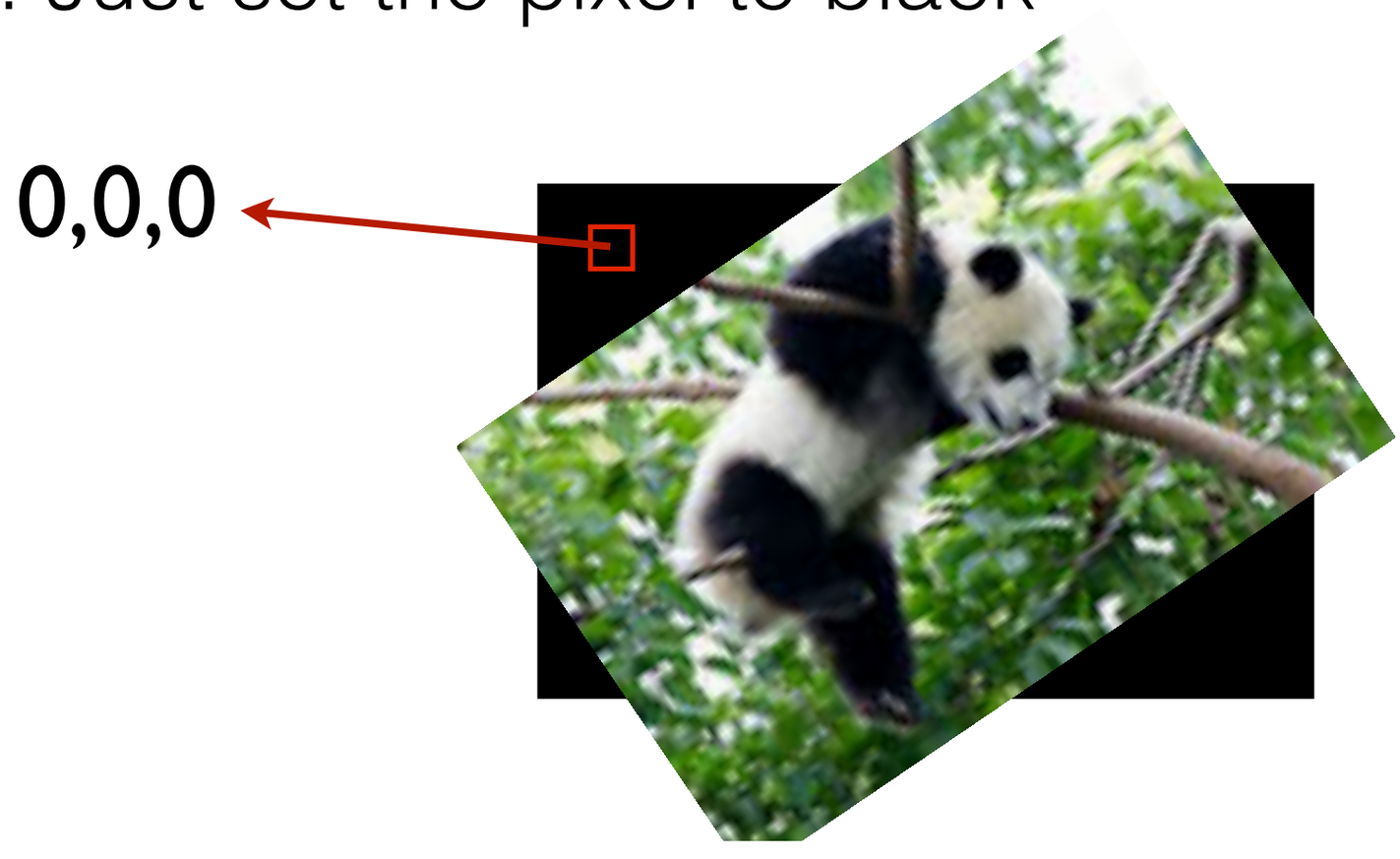
Image Padding

- Sometimes, we try to read outside the image
  - e.g. x,y are negative
- For example, when we rotate an image



Black Padding

- If u < 0 or u > WIDTH
- If v < 0 or v > HEIGHT
- Fix: Just set the pixel to black



Warping Artifacts

Types of Artifacts

- Even with inverse mapping, two big problems occur with warping

- Have to do with the fact we are resampling color information at a different rate.

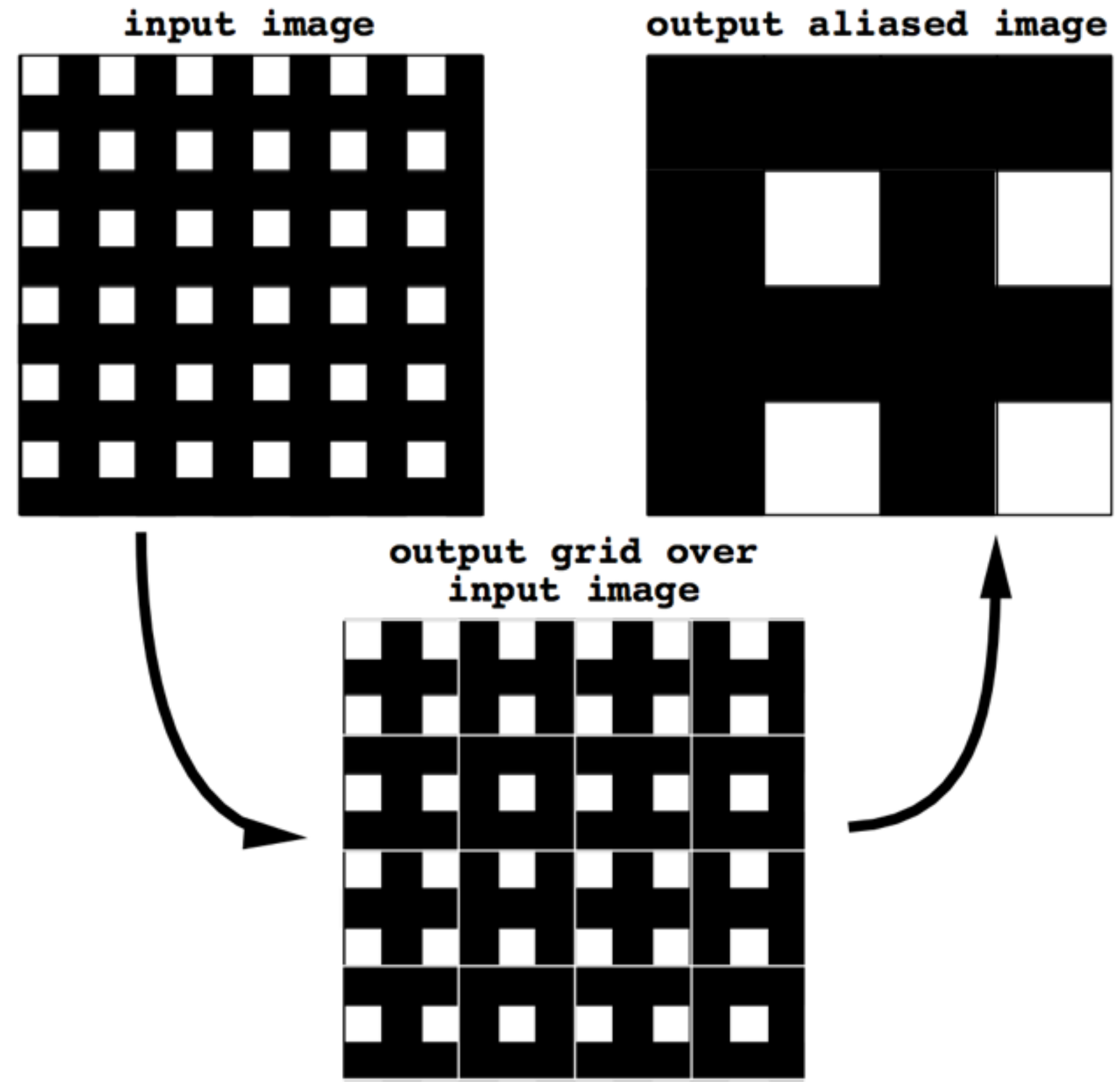
- Two types:

  - **Aliasing Artifacts**: caused by sampling the input image too coarsely. Occur when the image is being **minified**.

  - **Reconstruction Artifacts**: caused when using too crude a method (i.e. the psf) to reconstruct a pixel. Occur when the image is **magnified**.

- Aliasing leads to missing and/ or unwanted features

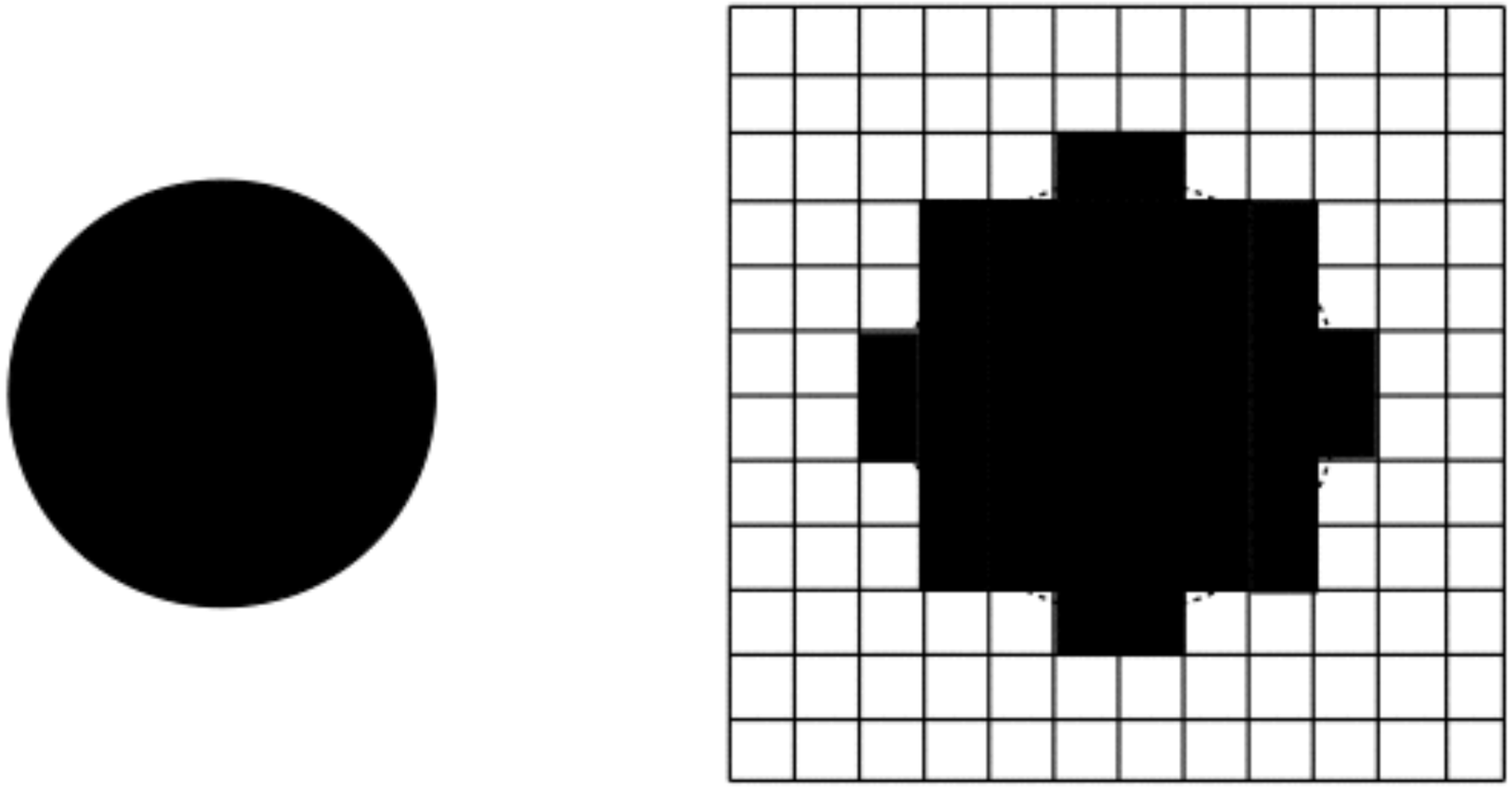
- Example: 12x12 images scaled to a 4x4 image.



a) aliasing artifacts

Reconstruction Artifacts

- Leads to staircasing or “jaggies”



b) reconstruction artifacts

Fixing Warping Artifacts

- Fixing aliasing artifacts: By carefully smoothing the input first.
    - Idea: We know we are going to lose information, so choose the right information to lose
- Fixing reconstruction artifacts: Do a better job of interpolation color values (instead of just rounding).
    - Idea: We know we are going to create information, so try to use as much data from the image as possible.
- More on both of these ideas in the coming lectures!

Affine Transformations

Affine Maps

- The simplest kind of transformations are linear
  - x and y are linearly related to (u, v)
  - All linear transformations are known as affine transformations
- Properties of affine transformations:
  - A straight line in the source is straight in the destination.
  - Parallel lines in the source are parallel in the destination.
  - They always have an inverse.
- The class of affine transformation functions is given by two equations, where six coefficients determine the exact effect of the transform.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}u + a_{12}v + a_{13} \\ a_{21}u + a_{22}v + a_{23} \end{bmatrix}$$

Remarks on Affine Maps

- Note that x is a linear combination of weighted u, v (by a_{11} and and a_{12}) and offset a_{13}
- Note that y is a linear combination of weighted u, v (by a_{21} and and a_{22}) and offset a_{23}
- The values of these six coefficients determine the specific effect
  - Four of the coefficients are sufficient for rotation/ scaling/shearing (which?)
  - Two of the coefficients are necessary for translation (which?)

Affine Maps as Matrices

- These two equations are often augmented by a third equation that may initially seem frivolous but allows for convenient representation and efficient computation.
- Equation 1=0x+0y+1 is obvious but including within our system allows us to write the affine system in the matrix form.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Homogeneous Form

- The 3x3 matrix of coefficients is known as the **homogeneous** transformation matrix.

- Using this augmented system, a two-dimensional point is represented as a 3x1 column vector where the first two elements correspond to the column and row while the third coordinate is constant at 1.

- When a two-dimensional point is represented in this form it is referred to as a **homogeneous coordinate**.

- When using homogeneous coordinates, the transformation of a point **u** with a transformation matrix A is given by the matrix product **x** = A**v** and is itself a point.

x = A**u** $\longrightarrow$ $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$

Sidebar:

Linear Algebra Review

Matrix-Vector Multiplication

- 3x3 matrix stores 3 equations in 3 variables (x,y,z) with 9 coefficients

- Each row of the matrix is multiplied by the column vector

  - Written **y** = A*__x__, where **y** is the vector **x** transformed by A

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

http://polymathprogrammer.com/2008/08/11/matrices-for-programmers/

Matrix-Matrix Multiplication

- Each row of first matrix is multiplied by each column of second matrix

- AB = C. C is also a matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} l & m & n \\ o & p & q \\ r & s & t \end{bmatrix} = \begin{bmatrix} al+bo+cr & am+bp+cs & an+bq+ct \\ dl+eo+fr & dm+ep+fs & dn+eq+ft \\ gl+ho+ir & gm+hp+is & gn+hq+it \end{bmatrix}$$

http://polymathprogrammer.com/2008/08/11/matrices-for-programmers/

Order of Operations

- Given matrices A, B, and a vector **x**
- AB**x** is a new vector, **y**
  - **y** is the vector representing **x** as first transformed by B (B**x**) and then transformed by A.
  - Rightmost matrix is applied first.

Inverse Matrices

- If A is a transformation, what if we want to undo A?
- This is another matrix, A^{-1}, where $A^{-1}A = \mathtt{I}$, where $\mathtt{I}$ is the **identity matrix**
- $\mathtt{I}\mathbf{x} = \mathbf{x}$. $\mathbf{x}$ is unchanged by applying I (why?)

$$\mathtt{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

http://polymathprogrammer.com/2008/08/11/matrices-for-programmers/

Inverse Matrices

- If $A = BC$, then A^{-1} takes a special form:
- $A^{-1} = C^{-1}B^{-1}$
- Why? Proof:
  - $A^{-1}A = \mathtt{I}$
  - $C^{-1}B^{-1}BC = \mathtt{I}$ (plug in for A and A^{-1})
  - $C^{-1}\mathtt{I}C = \mathtt{I}$ (since $B^{-1}B = \mathtt{I}$, by definition)
  - $C^{-1}C = \mathtt{I}$ (since $\mathtt{I}C = C$)
  - $\mathtt{I} = \mathtt{I}$ (since $C^{-1}C = \mathtt{I}$, by definition)

http://polymathprogrammer.com/2008/08/11/matrices-for-programmers/

Computing Inverse Matrices

- If M is an affine matrix, this is not so hard:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

- Then the inverse is:

$$M^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} & a_{12}a_{23} - a_{13}a_{22} \\ -a_{21} & a_{11} & -a_{11}a_{23} + a_{13}a_{21} \\ 0 & 0 & a_{11}a_{22} + a_{12}a_{21} \end{bmatrix}$$

- In general, the inverse is expressed as:
  - A(M) is the adjoint
  - |M| is the determinant of M

$$M^{-1} = \frac{\mathcal{A}(M)}{|M|}$$

Lec18 Required Reading

- Szeliski 3.6.1 (warping and morphing)
- Hunt, Ch. 7.3 (nonlinear warping)

- House, 9.4, 9.5, 9.7