CPSC 4040/6040 Computer Graphics Images

Joshua Levine
levinej@clemson.edu

Lecture 18
Affine and Perspective Warping

Oct. 22, 2015

Agenda

- Quiz 3 DUE
- PA05 Questions? DUE Tues. 10/27
- Project Proposals DUE Thurs. 10/29

Refresher from Lec17

Warps are Domain Transformations

- Apply a function f from R^2 to R^2 within the image domain

- If (u,v) had color c in the input, then (x,y) = f(u,v) has color c in the output.
  - Here, f takes a pixel coordinate and returns a pixel coordinate

- 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

Destination Image Size

- Can compute the destination dimensions by transforming the bounds and using the width and height of the bounds as the destination dimensions.

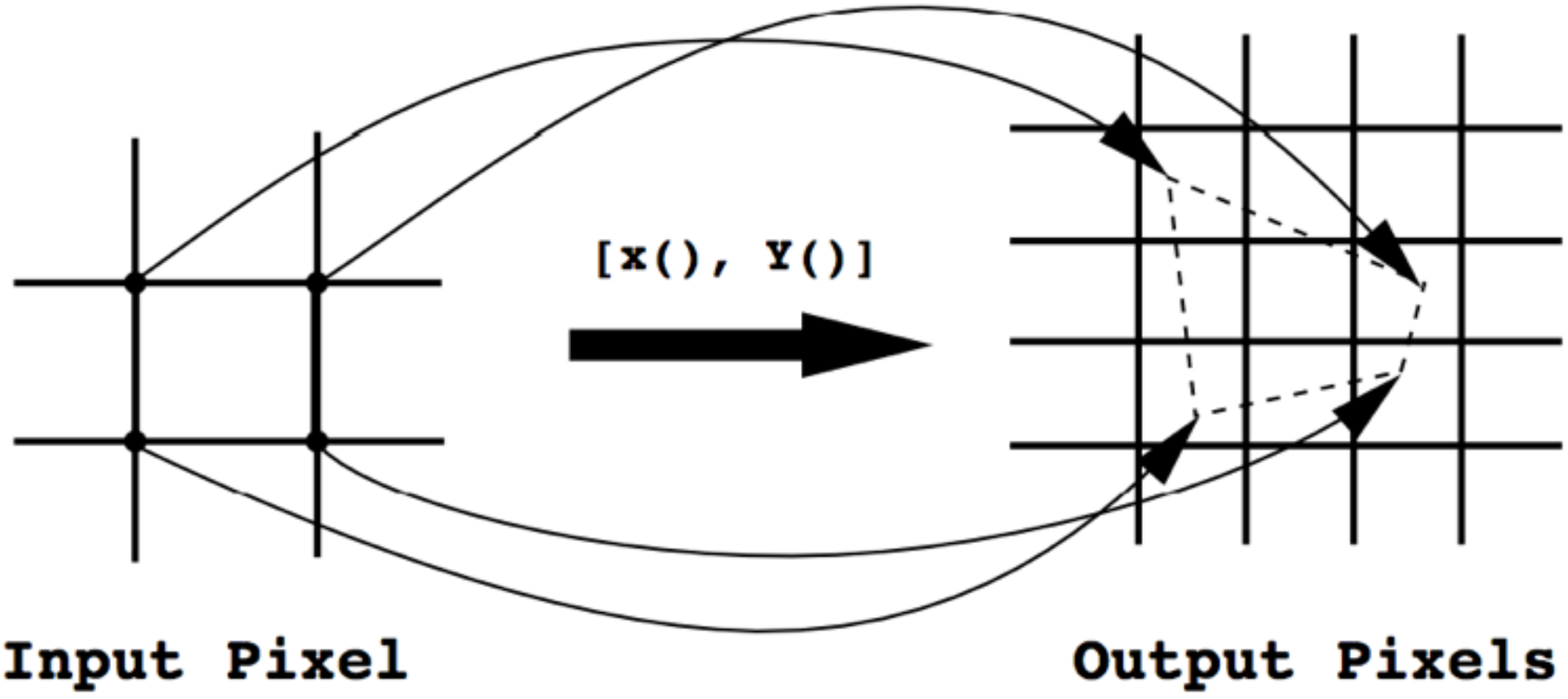
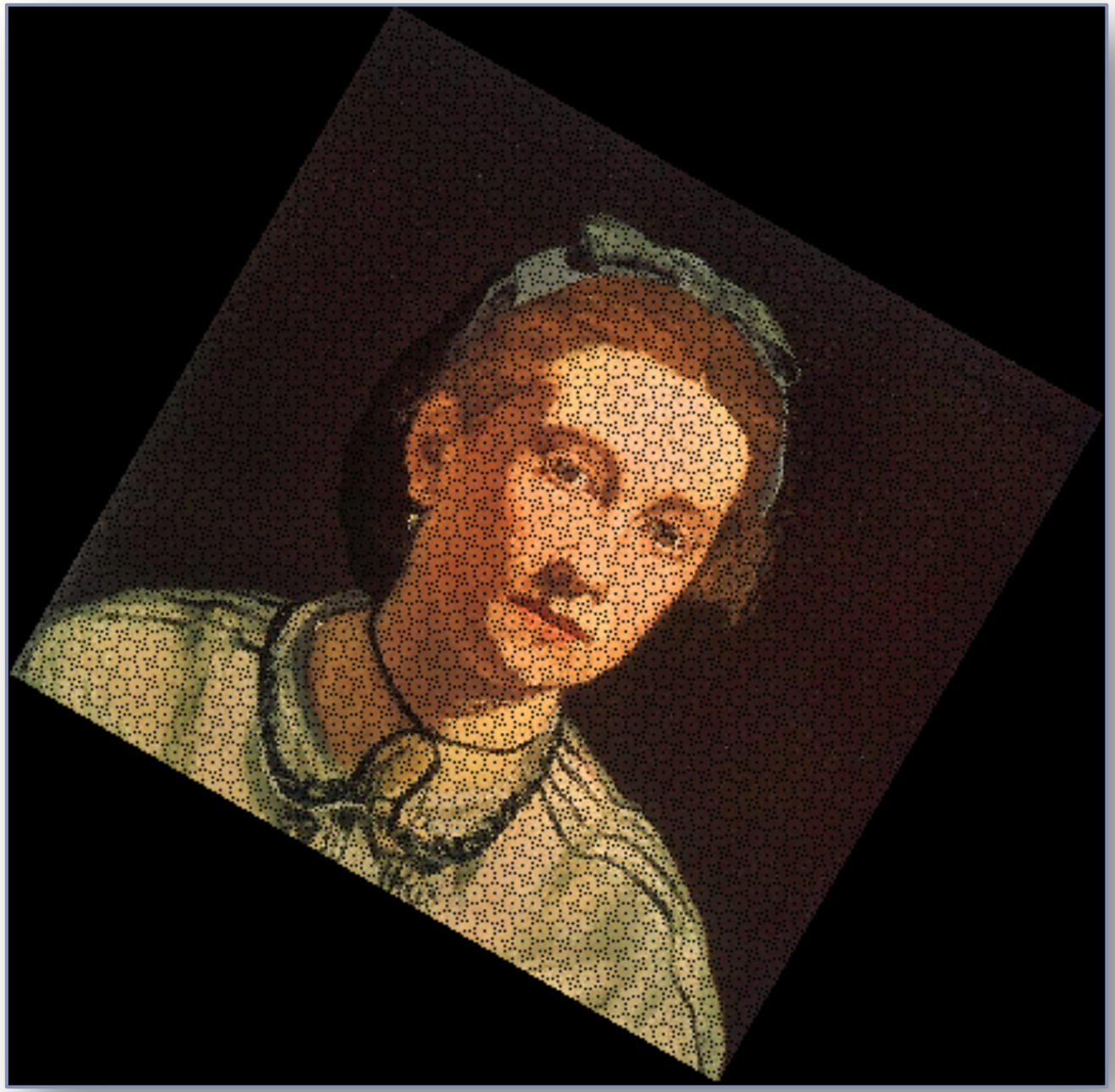


Figure 9.3: Projection of pixel area onto output raster

Forward Mapping Leaves Gaps



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

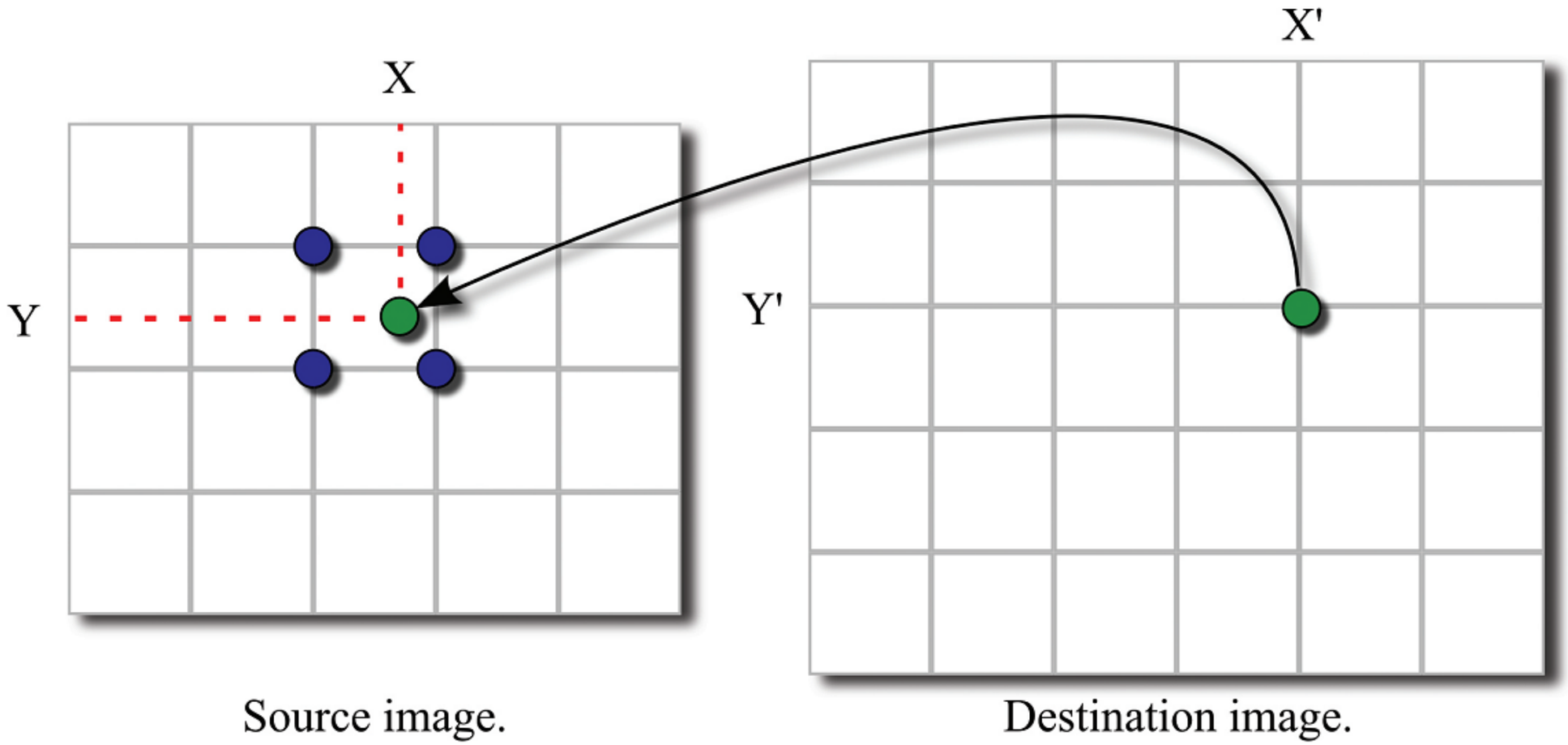


Figure 7.4. Reverse mapping.

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.

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

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

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

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

More on Affine Transformations

Types of Affine Transformations

- An affine transformation matrix is a six parameter entity controlling the coordinate mapping between source and destination image.
- The following table shows the correspondence between coefficient settings and effect.

	a_{11}	a_{21}	a_{12}	a_{22}	a_{13}	a_{23}
translation	1	0	0	1	δx	δy
rotation	$\cos(\theta)$	$-\sin(\theta)$	$\sin(\theta)$	$\cos(\theta)$	0	0
shear	1	sh_y	sh_x	1	0	0
scale	s_x	0	0	s_y	0	0
horizontal reflection	-1	0	0	1	x_c	0
vertical reflection	1	0	0	-1	0	y_c

Table 7.1. Coefficient settings for affine operations.

(a) Translation.

(b) Rotation.

(c) Shearing.

(d) Uniform scaling.

(e) Nonuniform scaling.

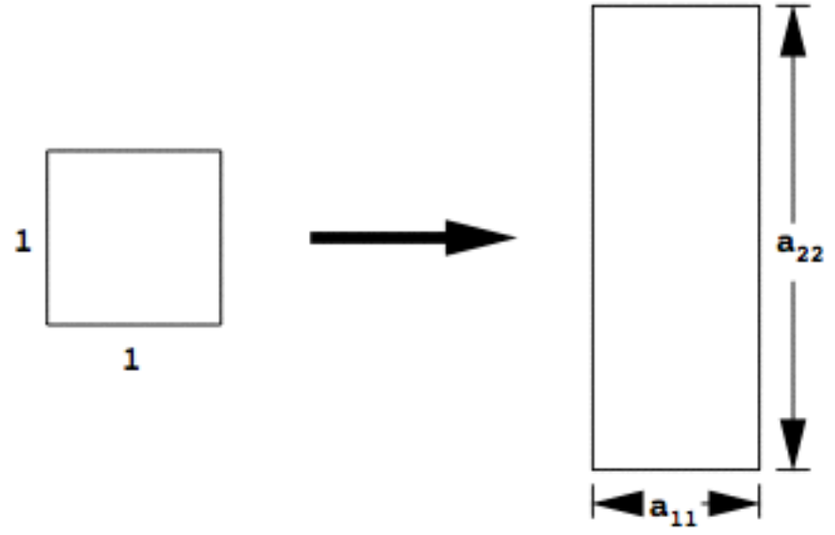
(f) Reflection.

Figure 7.1. Linear geometric transformations.

Affine Types: Scaling

$$\textbf{scale}\ (x, y) = (a_{11}u, a_{22}v)$$

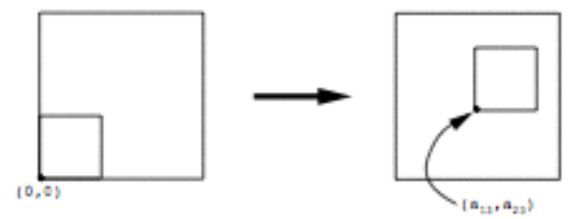
$$\begin{bmatrix} a_{11}u \\ a_{22}v \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$



Affine Types: Translation

$$\textbf{translation } (x, y) = (u + a_{13}, v + a_{23})$$

$$\begin{bmatrix} u + a_{13} \\ v + a_{23} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a_{13} \\ 0 & 1 & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$



A Note on Translation

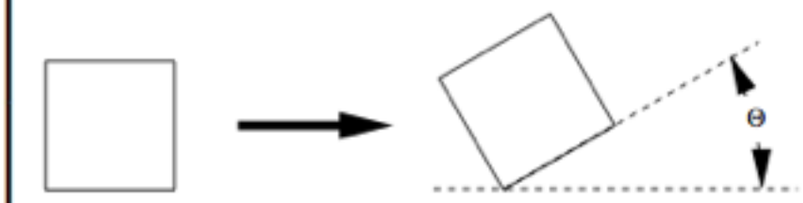
- After translating an image, what should be the new image size?
- Example, 100x200 image translated 50 horizontally and 20 vertically
- How big is the final image?
- Where is (0,0) now?
- We often crop the image in this case.
    - Translations have more interesting effects when used in combination with other types of warps.

Affine Types: Rotation

- Rotation about the origin (0,0)

$$\textbf{rotation}\ (x, y) = (u\cos\theta - v\sin\theta, u\sin\theta + v\cos\theta)$$

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

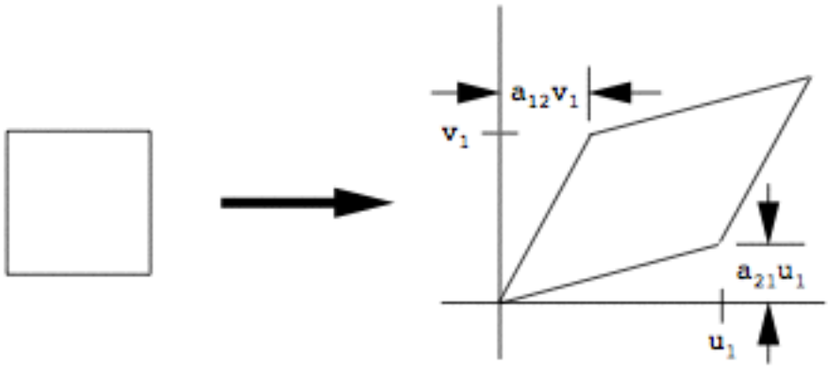


- How would we rotate about a different point?

Affine Types: Shear

$$\textbf{shear}\ (x, y) = (u + a_{12}v, a_{21}u + v)$$

$$\begin{bmatrix} u + a_{12}v \\ a_{21}u + v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_{12} & 0 \\ a_{21} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$



Practice

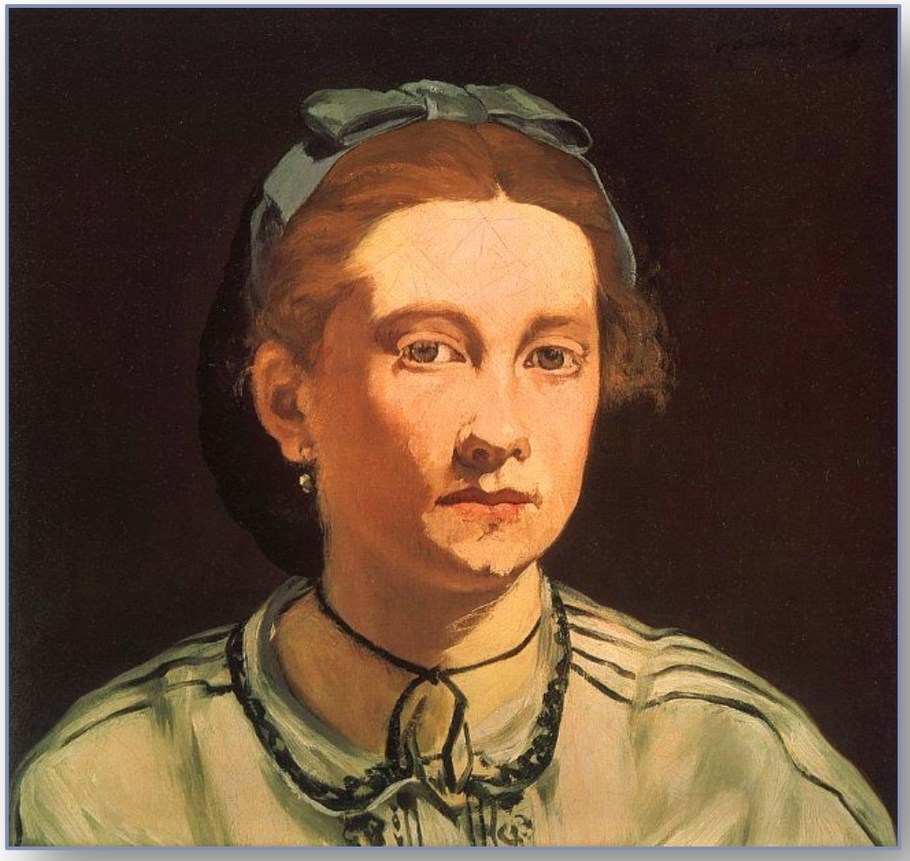
- Explain what the following transformation accomplishes:

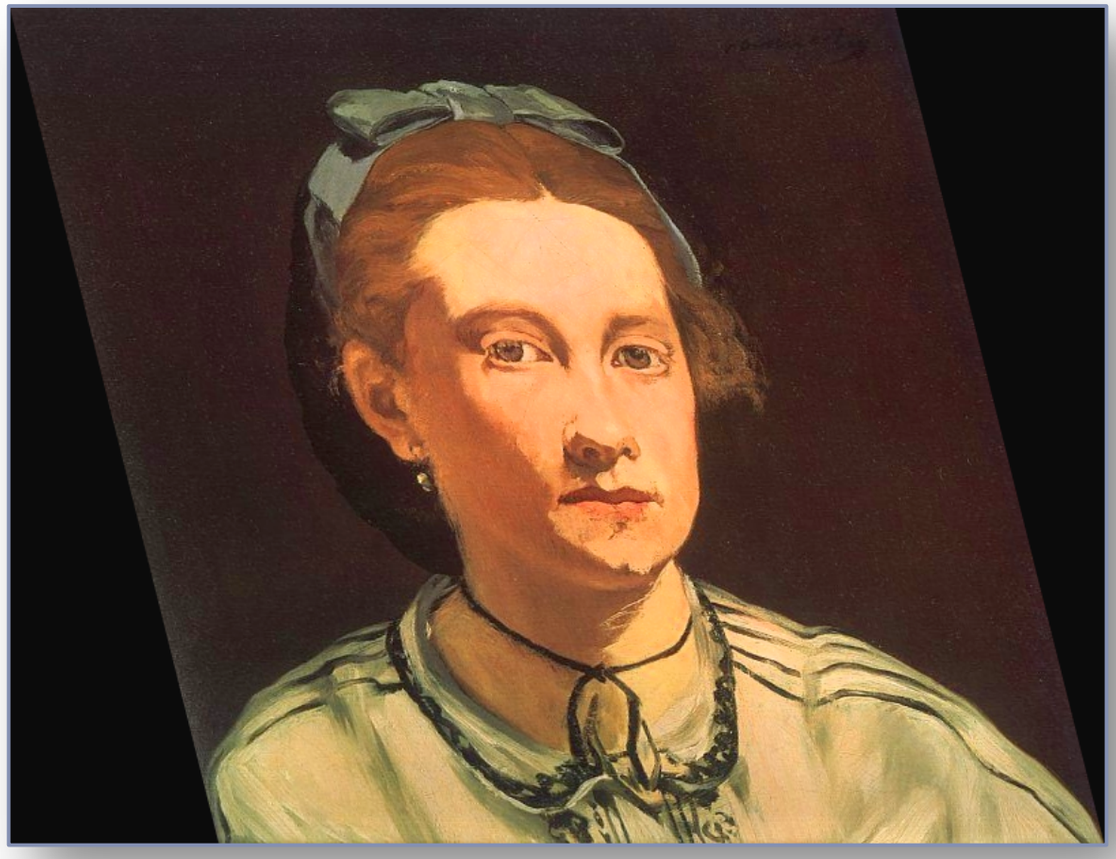
1	0.25	0
0	1	0
0	0	1

Practice

- Explain what the following transformation accomplishes:

1	0.25	0
0	1	0
0	0	1



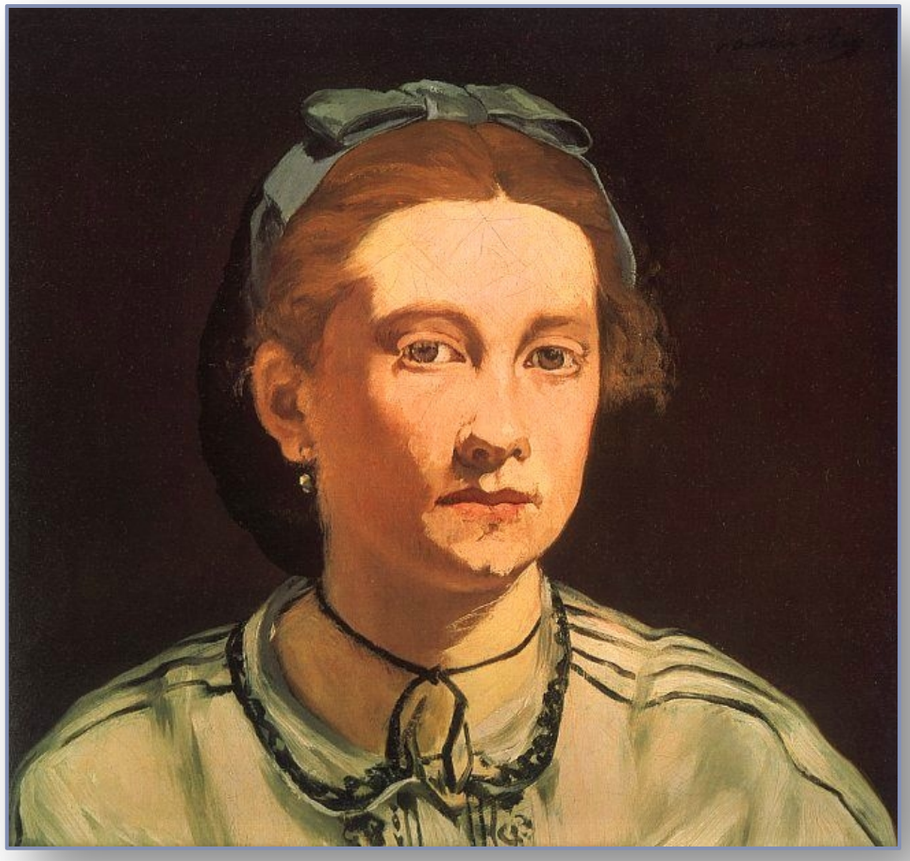


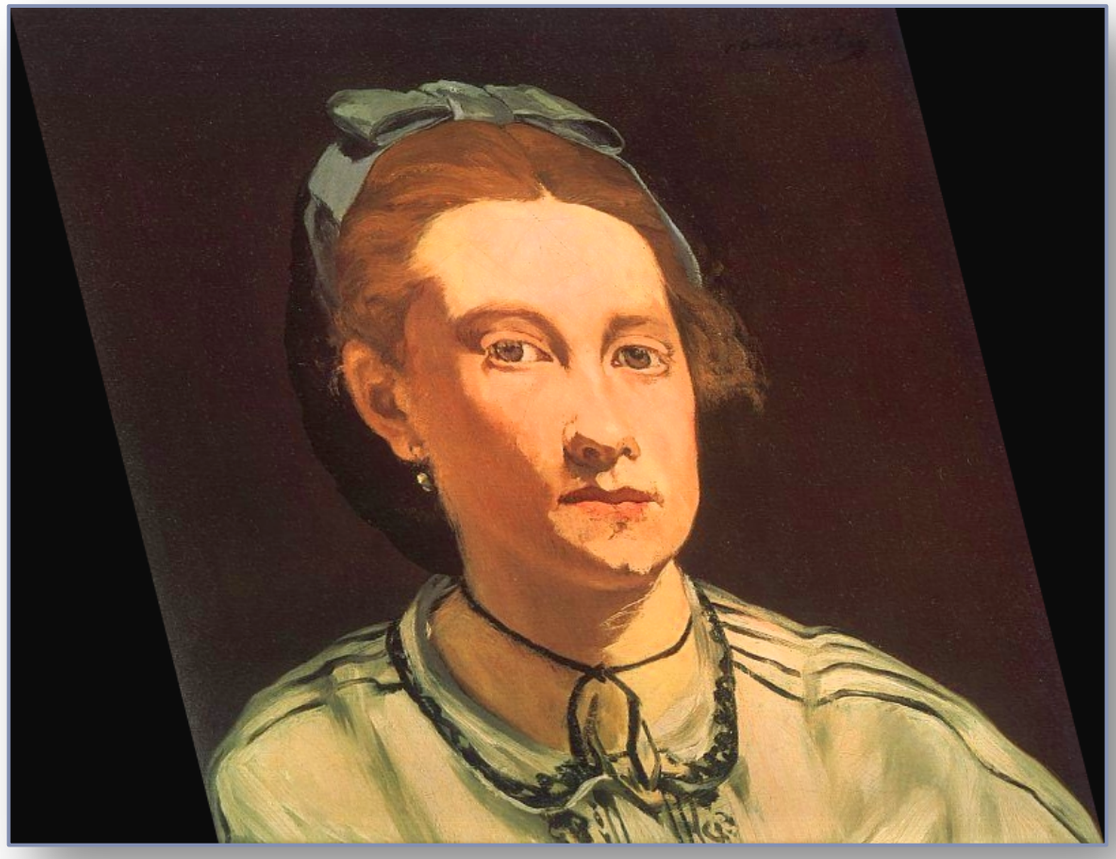
Practice

- Explain what the following transformation accomplishes:

1	0.25	0
0	1	0
0	0	1

(Note: origin is top left)





Practice

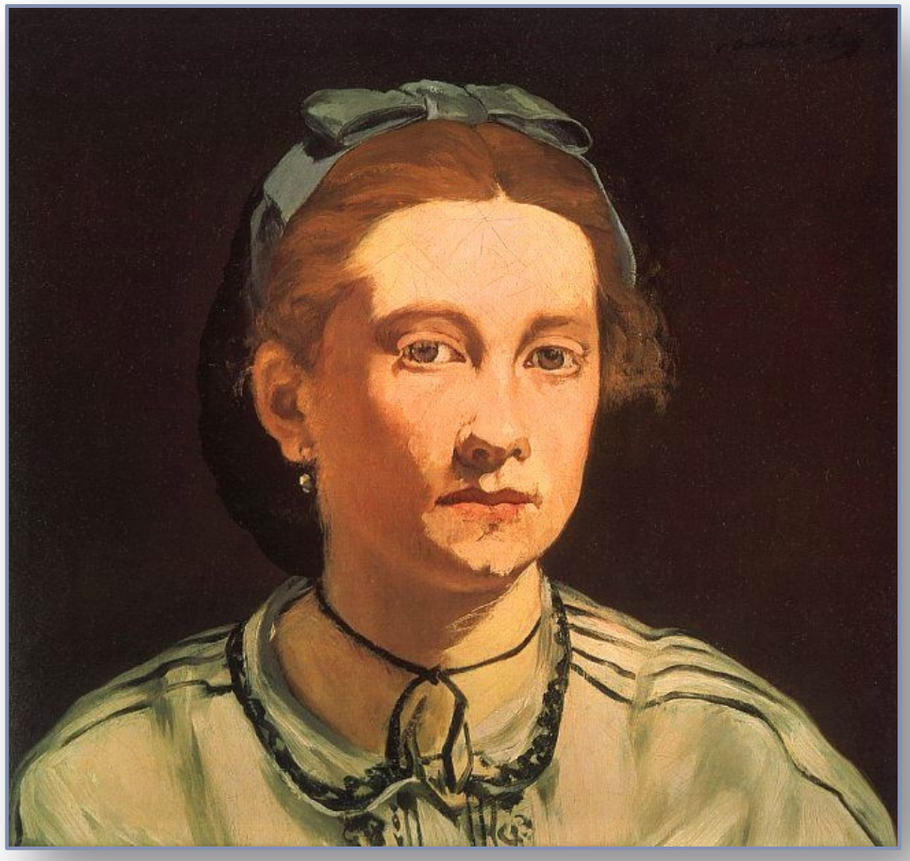
- Explain what the following transformation accomplishes:

1	0.25	0
0.5	1	0
0	0	1

Practice

- Explain what the following transformation accomplishes:

1	0.25	0
0.5	1	0
0	0	1



Practice

- Explain what the following transformation accomplishes:

0.87	-0.5	0
0.5	0.87	0
0	0	1

Practice

- Explain what the following transformation accomplishes:

0.87	-0.5	0
0.5	0.87	0
0	0	1

Practice

- Explain what the following transformation accomplishes:

0.5	0	0
0	2	0
0	0	1

Practice

- Explain what the following transforma plishes:

0.5	0	0
0	2	0
0	0	1

Sequences of Transformations

- A single homogeneous matrix can also represent a sequence of individual affine operations.
  - Let A and B represent affine transformation matrices. the affine matrix corresponding to the application of A followed by B is given as BA, a homogeneous transformation matrix.
  - The order of multiplication is important — rightmost matrix happens first.
  - Assume, for example, that matrix A represents a rotation of 30 degrees about the origin and matrix B represents a horizontal shear by a factor of .5. The affine matrix corresponding to the rotation followed by shear is given as BA.

$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} .866 & -0.5 & 0 \\ 0.5 & .866 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.116 & -0.067 & 0 \\ 0.5 & .866 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \qquad (7.4)$$

Perspective Warps

Working with Homogeneous Coordinates

- So far, we have left kept the bottom row of the transformation matrix fixed at 0,0,1

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

- Results: if we transform a u,v,1 pair we are always left with a third component (called w) which is also equal to 1.

- If a_{31}, a_{32}, or a_{33} were nonzero, the equation would be nonlinear

Normalizing Coordinates

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ a_{31}u + a_{32}v + 1 \end{bmatrix}$$

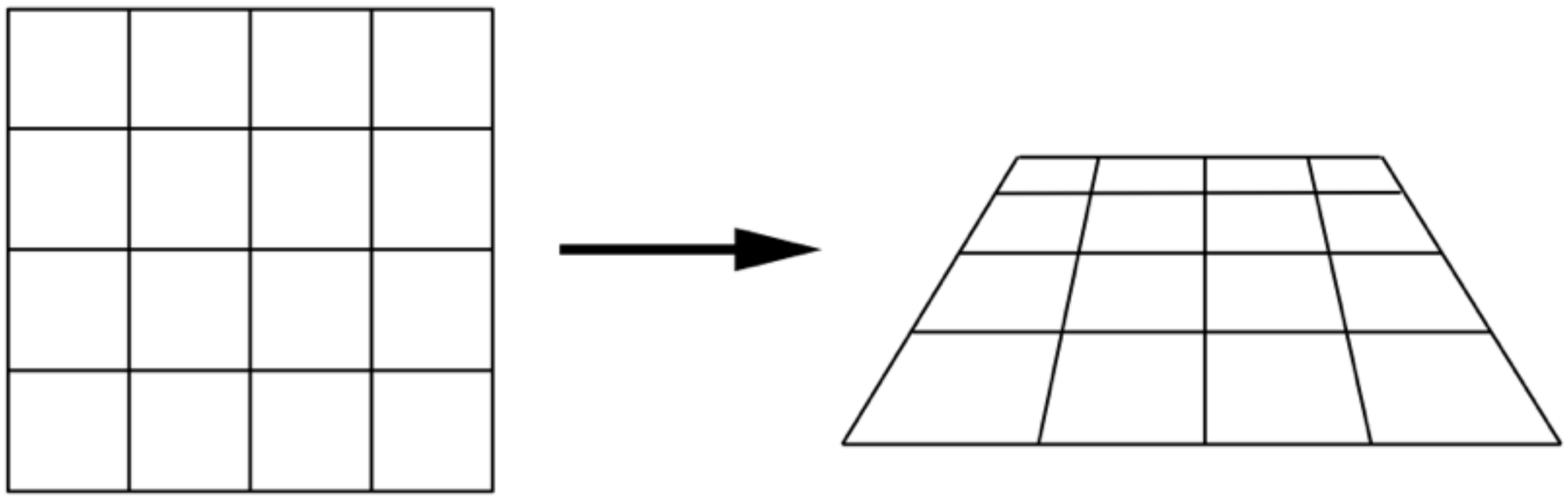
- Rewrite as:

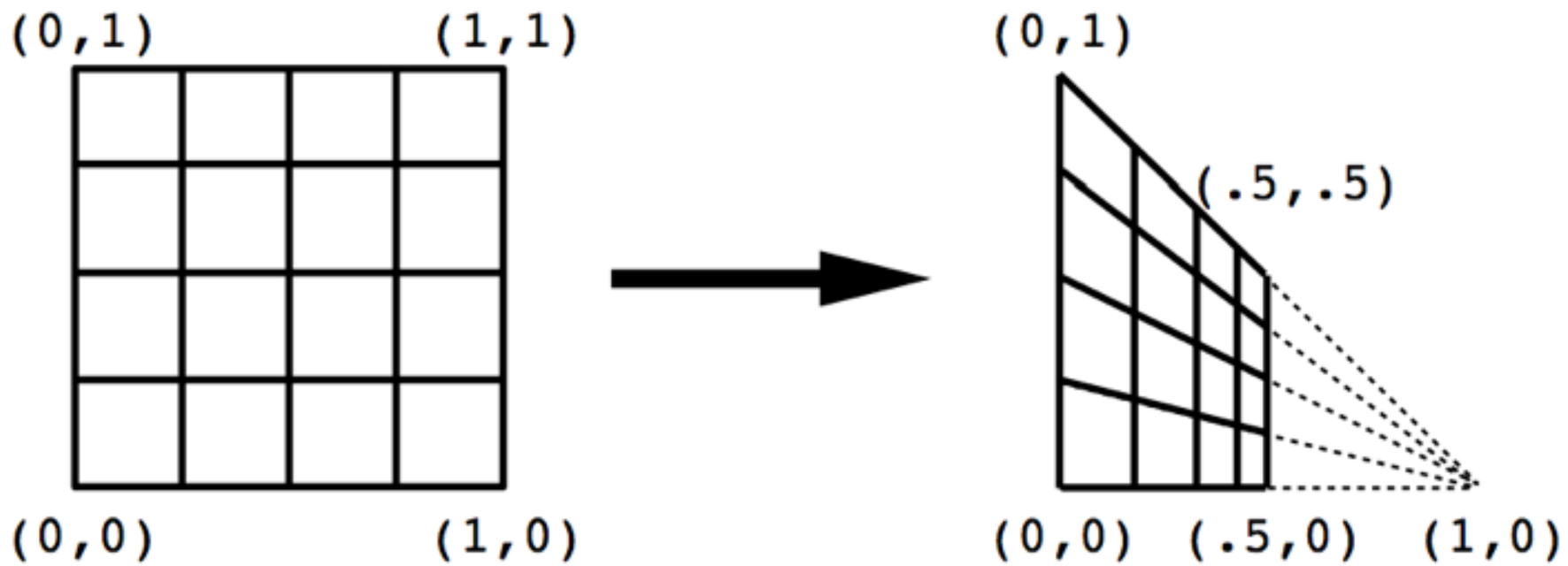
$$\frac{1}{w} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix}$$

This is no longer linear!

Perspective Warping

- This form of warping stretches and squishes in different places
- Straight lines stay straight, but not necessarily parallel





$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$P \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ u+1 \end{bmatrix} \Longrightarrow \begin{bmatrix} \frac{u}{u+1} \\ \frac{v}{u+1} \\ 1 \end{bmatrix}$$

Figure 9.11: Example Perspective Warp

What Actually Happened?

- Mathematically, working with homogenous coordinates is working in three dimensions, not two!
  - (u,v,w) to (x,y,z); not (u,v) to (x,y) (or, f goes from R^3 to R^3, not f from R^2 to R^2)
    - Not just a matter of convenience, this helps us express translations. And more.

- We're treating points as lying on the plane w = 1, as viewed from above, and transforming them back to the plane z = 1.
  - Affine warps preserve the z coordinate
  - Perspective warps do not preserve it!
    - So we normalize them back.

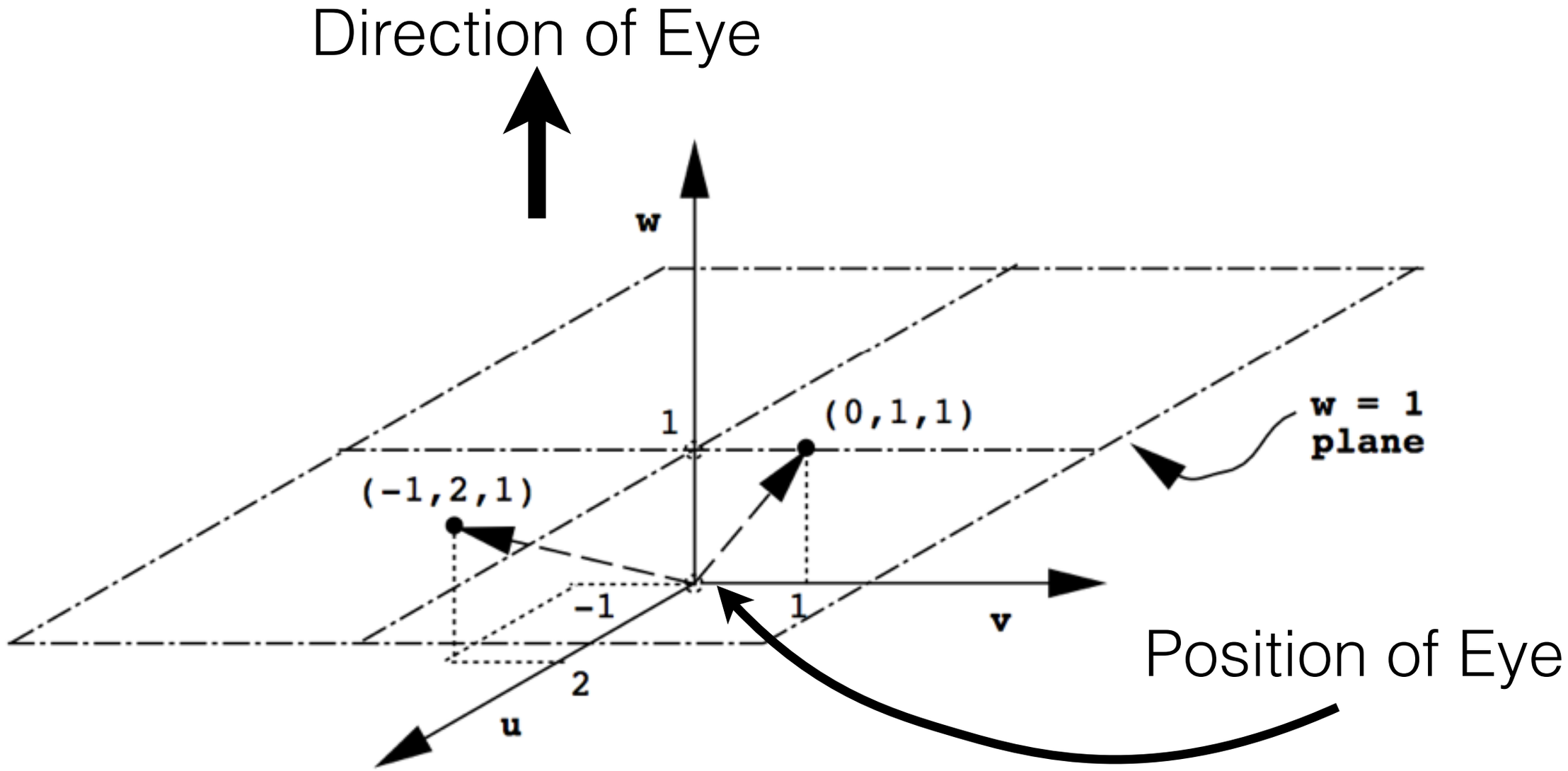


Figure 9.10: Homogeneous Coordinates Lie on the Plane $w = 1$

- When a_{31}, a_{32} do not equal zero, we tilt this plane
- But we are still looking at it along the w-axis, eye is located at (0,0,0)
- Dividing by w shortens vector to lie in w = 1 plane
- Given (x,y,w) and (x/w,y/w,1), the point still lies on the same ray exiting from the eye passing through (0,0,0) and (x,y,w)

Other Nonlinear Transformations

Nonlinear Mapping

- Lots of interesting effects can be achieved if we use nonlinear functions
- However, we have to be extremely careful with the size of the output image as well as interpolation issues.
    - This isn't a problem with perspective warps though (why?)

“Twirling” (from Hunt)

- This mapper imposes a sin-wave displacement on a location in both the x and y dimensions
- The strength of the displacement and the frequency of the displacement can be controlled

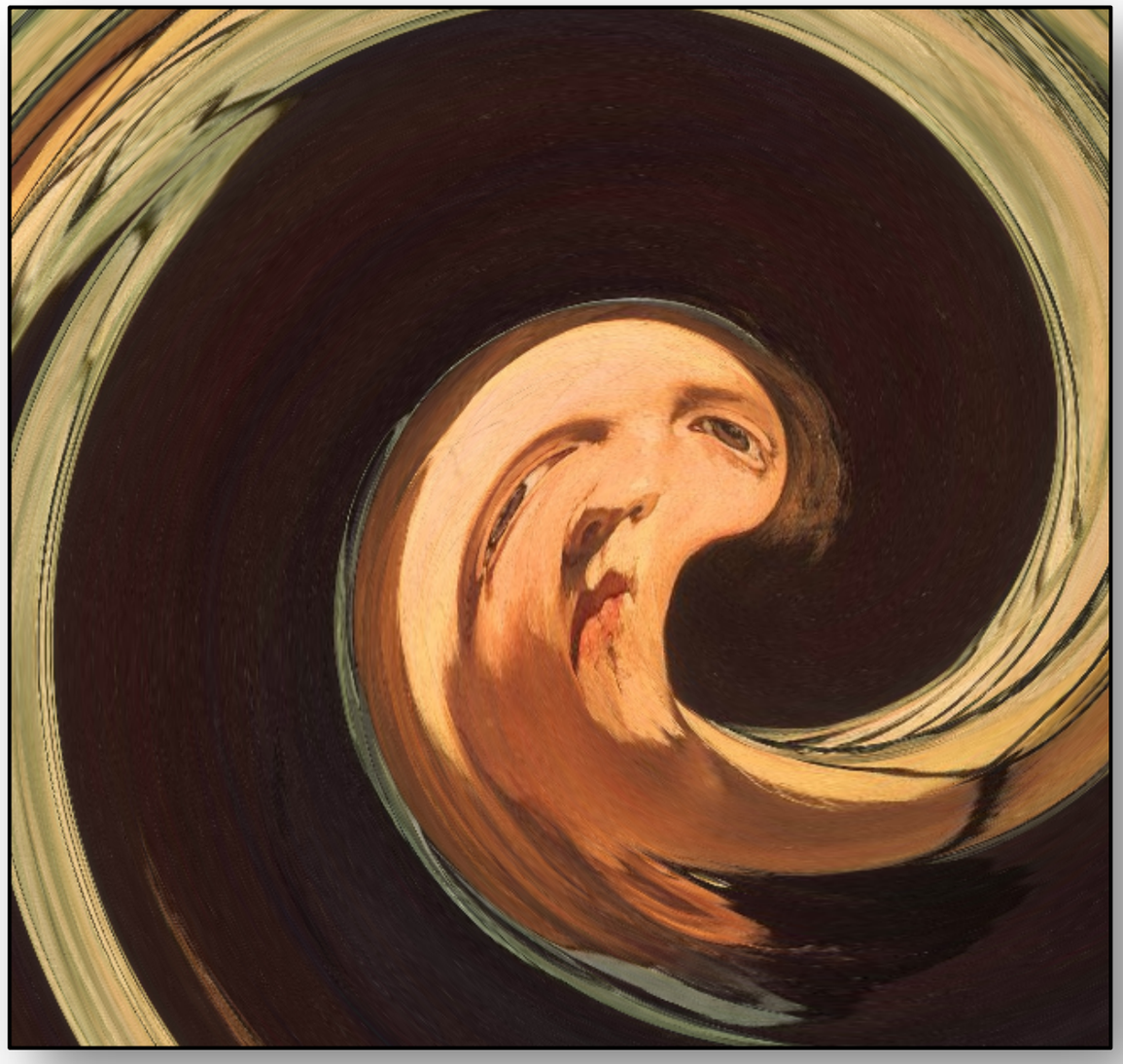
$$T_x(x,y) = r \cdot \cos(\theta + \text{strength} \cdot (r - \text{minDim})/\text{minDim}) + \text{centerX}$$

$$T_y(x,y) = r \cdot \sin(\theta + \text{strength} \cdot (r - \text{minDim})/\text{minDim}) + \text{centerY}$$

- Where:
  - r is the distance between the pixel and the center of the twirl (located at pixel (centerX,centerY))
  - θ is the angle of rotation between the point and the center coordinate
  - minDim = min(WIDTH,HEIGHT)

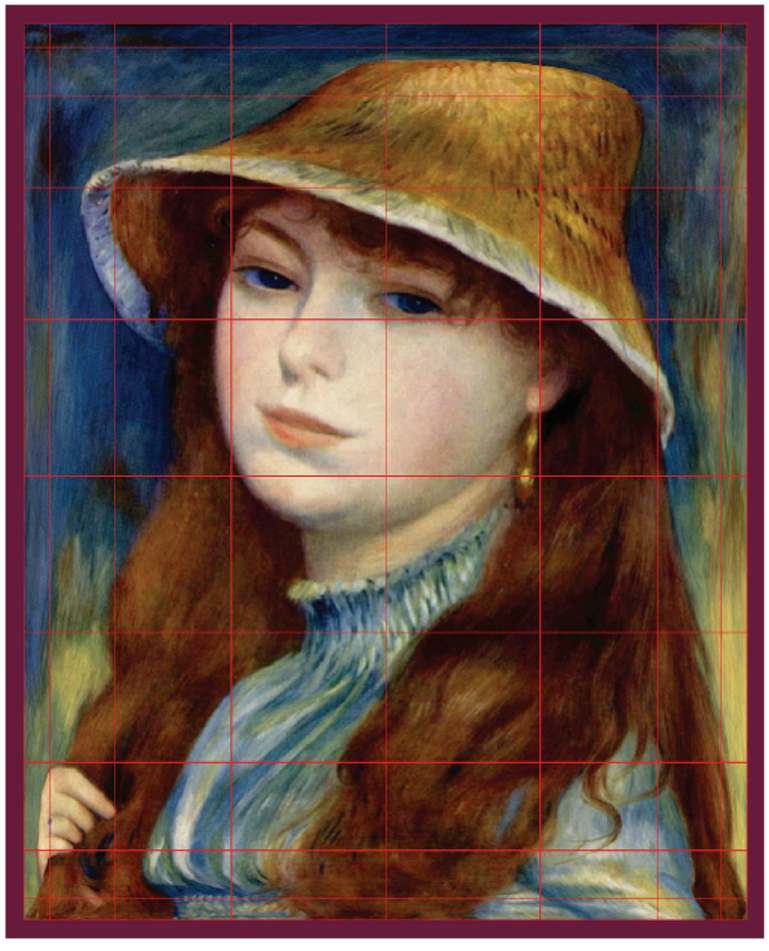
Twirl Mapper

center = (0.5,0.5)
strength = 3.75



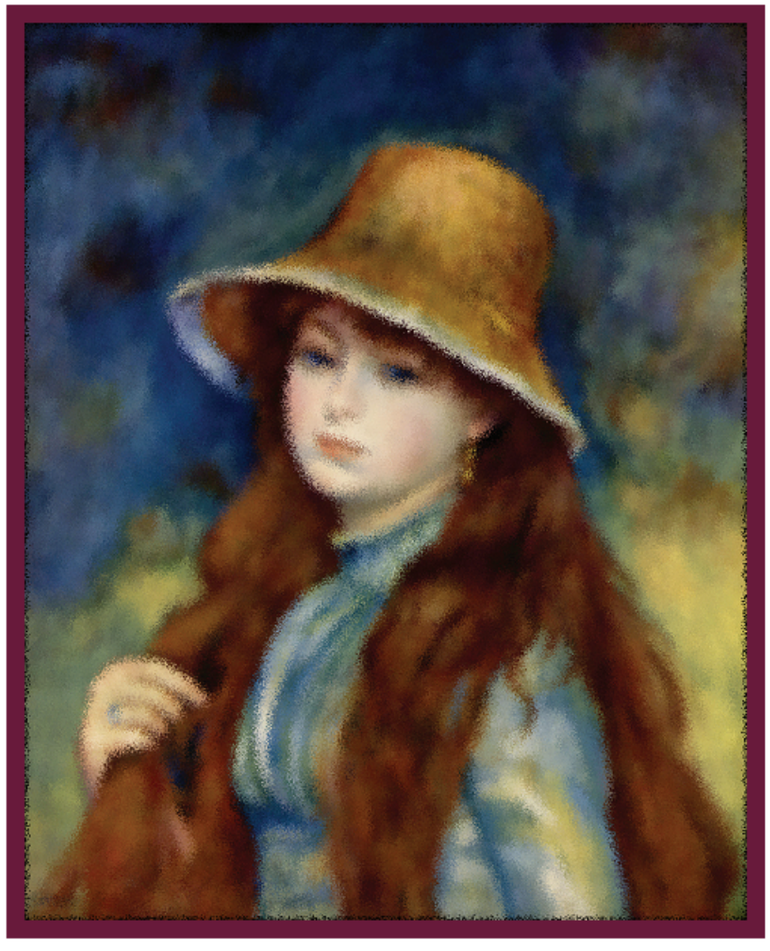
center = (0.65,0.5)
strength = 7.5

More Nonlinear Examples



(a) Warped.

(b) Rippled.

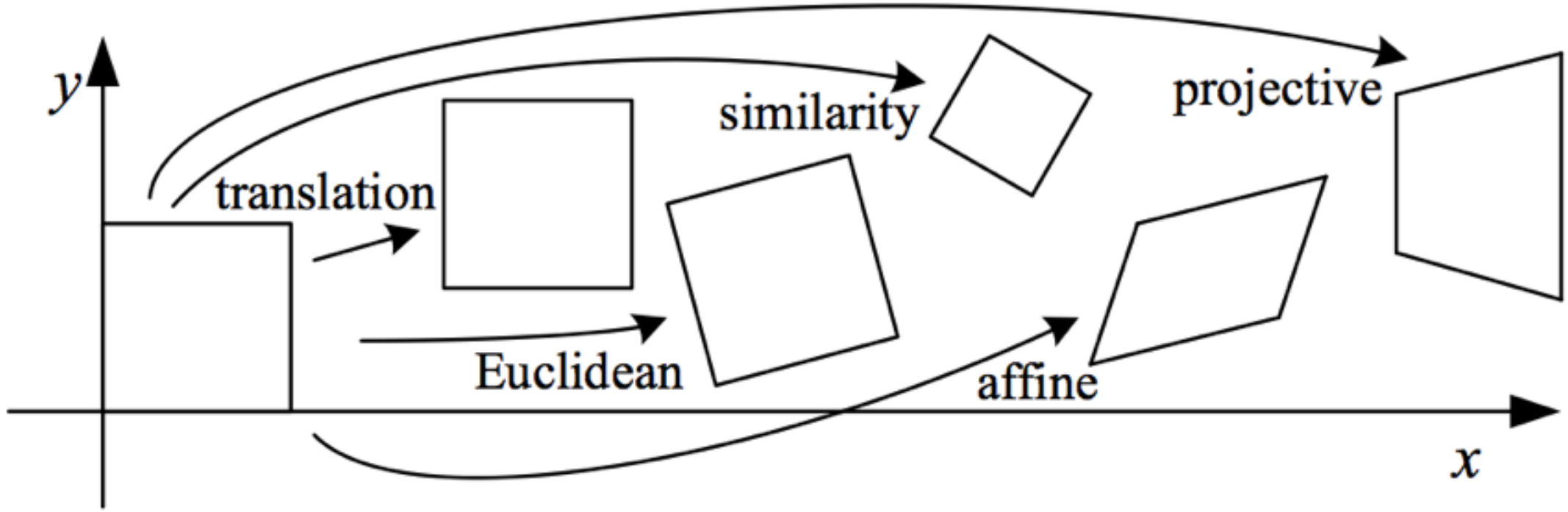


(c) Diffused.

Figure 7.12. Nonlinear geometric transformations.

Summary

- Which components of the transformation matrix are needed for each type?



From Szeliski, Ch. 2

Vector and Matrix Source Classes

Vector Classes

Can be created as 2d (Vector2d), 3d (Vector3d), 4d (Vector4d), or Nd (Vector)

Instantiation:

```
Vector2d vec1(1,0);
Vector3d vec2(-1,5.0,0.2);
```

Access data:

```
double xcoord = vec1[0];
double ycoord = vec1.y;
vec2[2] = 2.5;
```

Operators (+,-; scalar *,/; dot *; cross %; and more)

```
Vector2d vec3(0,1);
Vector2d vec4 = vec1 + vec3;
```

Functions:

```
double length = vec2.norm();
Vector3d vec5 = vec3.normalize();
```

Matrix Classes

Can be created as 2d (Matrix2x2), 3d (Matrix3x3), 4d (Matrix4x4), or Nd (Matrix)

Instantiation:

```
Matrix2x2 mat1(1,0,0,1);
Matrix3x3 mat2(3,0,0,0,2,0,0,0,1);
```

Access data:

```
Vector2d row = mat2[1];
mat1[1][0] = 1;
```

Operators (element-wise +,-,*,/; Mat-Vec and Vec-Mat *, outer product as &)

```
Matrix2x2 mat3(0,1,1,0);
Matrix2x2 mat4 = mat1 + mat3;
Vector2d vec6 = mat2 * vec2;
Vector2d vec7 = vec2 * mat2;
```

Functions:

```
Matrix3x3 invMat4 = mat4.inv();
Matrix3x3 transposeMat4 = mat4.transpose();
```

Warping Algorithm

Review: Inverse Map Algorithm

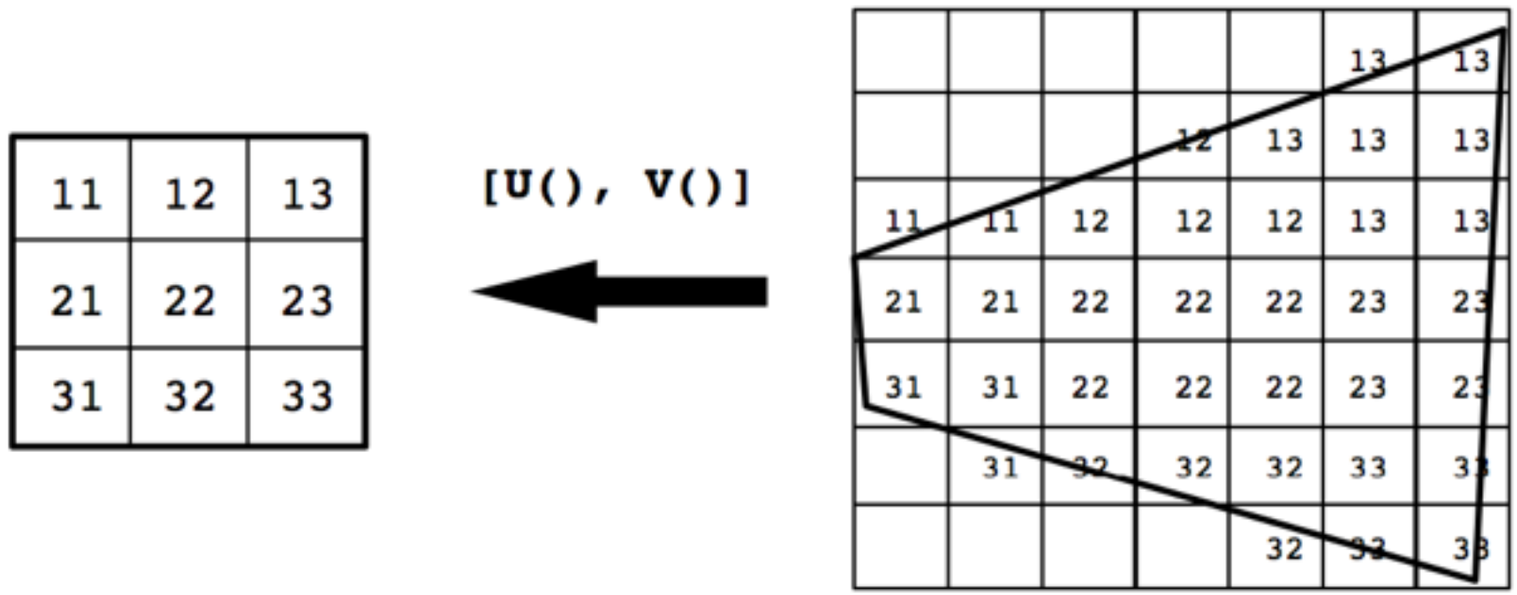


Figure 9.4: Inverse Map gives Complete Covering

```
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))];
```

Matrix-Based Perspective Warping

- Given an input image, IN, of width W and height H as well as a 3x3 matrix M:

    1. Using the forward map, M, compute W2, H2 to store the width and height of the bounding box of OUT

    2. Allocate output image OUT

    3. Compute inverse matrix invM

    4. Use the inverse map (invM) to fill in all of pixels of OUT from their respective pixels in IN

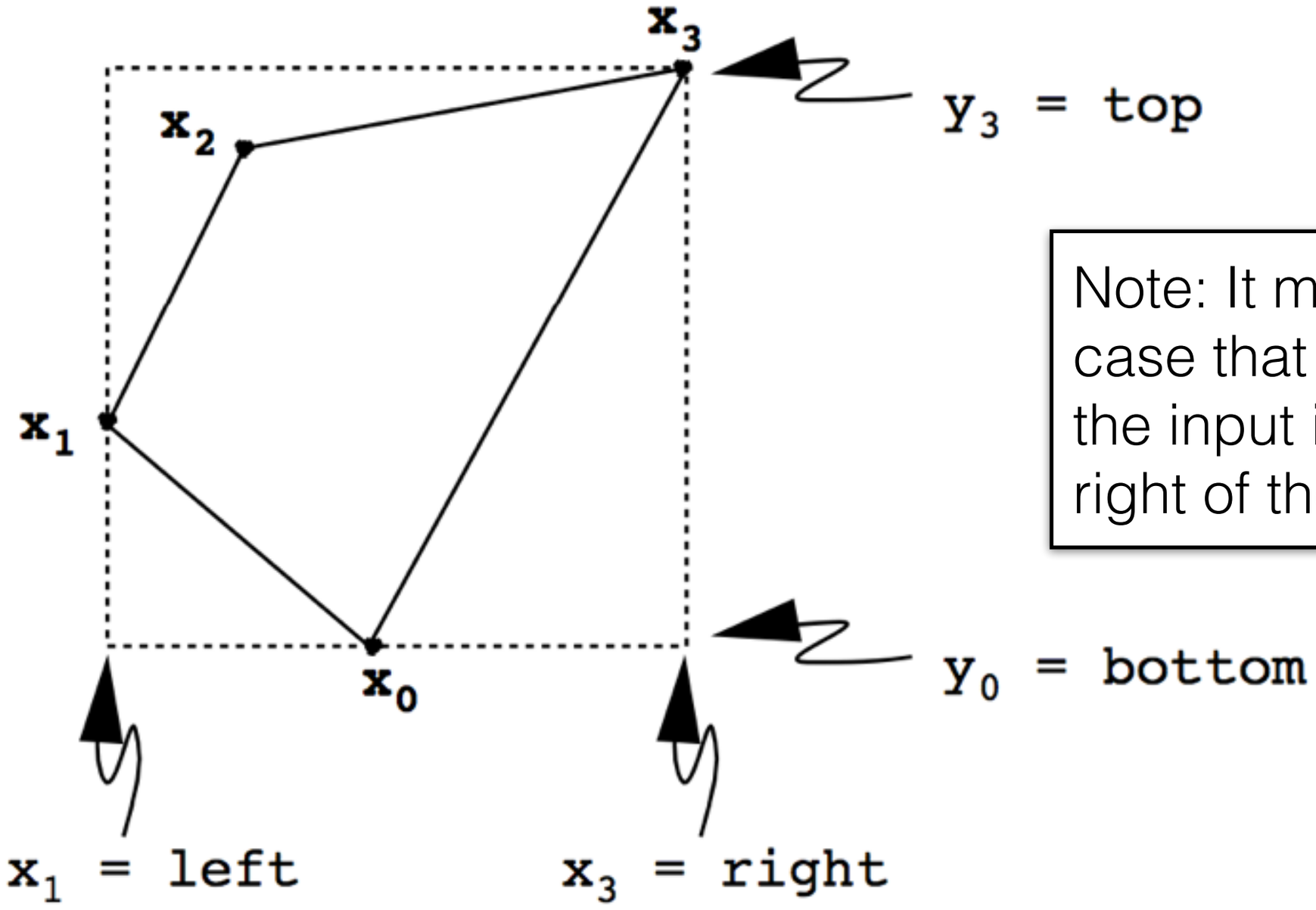
Step 1: Bounding Box

- Apply the forward map to each (u,v) for the corners of the input image
  - (u,v) for (0,0), (W,0), (0,H), (W,H)
- Make sure to normalize by w:

$$M \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad x = u'/w', \;\; y = v'/w'$$

- Produces four positions: (x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4)

The Bounding Box is computed using the minimum and maximum x and y



Note: It may not be the case that the top right of the input image is the top right of the output image

Figure 10.9: Constructing a Bounding Box

Step 2: Allocate Output Image

```
W2 = right - left

H2 = top - bottom

OUT = new pixmap*[H2]

OUT[0] = new pixmap[H2*W2]
```

Step 3: Compute invM

Determinant

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$|M| = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{32}a_{21} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{32}a_{23}$$

$$\mathcal{A}(M) = \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}$$

Adjoint

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

`invM`

Step 4: Apply the Inverse Map

```
Vector3d origin(left, bottom, 0);

OUT = new pixmap[H2][W2];
Matrix3x3 invM = M.inv();

for (y = 0; y < H2; y++) {
  for (x = 0; x < W2; x++) {
    //map the pixel coordinates
    Vector3d pixel_out(x, y, 1);
    pixel_out = pixel_out + origin;
    Vector3d pixel_in = invM * pixel_out;

    //normalize the pixmap
    float u = pixel_in[0] / pixel_in[2];
    float v = pixel_in[1] / pixel_in[2];

    //Could use better interpolation
    OUT[y][x] = IN[round(v)][round(u)];
  }
}
```

More on using a vector and matrix class next week!

Projective Warping

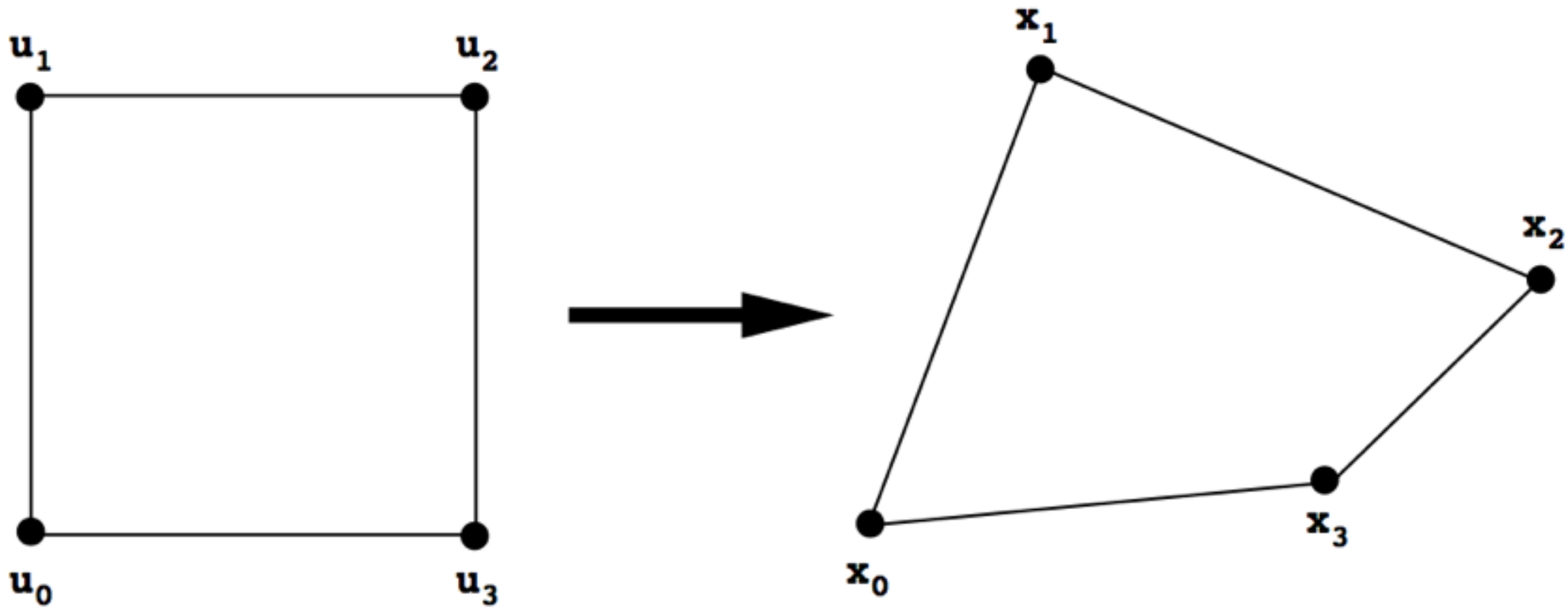
User Driven Warping

- Specifying a matrix can be non-intuitive
- Instead of specifying a matrix to warp, a more useful mechanism would be allowing the user to move the corners

- We can express this as a matrix warp!

Projective Warps

- What is the matrix?
- What the knowns? Unknowns? How many?



Algebra

- A general 3x3 matrix can express this entire class of warps (9 unknowns)
  - a_{33} acts as a global scale parameter, so we can always set it to 1 without losing generality
- The remaining 8 unknowns can be solved by 4 pairs of equations using the 8 known x_i, y_i values and the 8 known u_i, v_i values
- Solving these 8 equations gives the 8 remaining a_{ij} unknowns

$$x_i = \frac{a_{11}u_i + a_{12}v_i + a_{13}}{a_{31}u_i + a_{32}v_i + 1}$$

$$y_i = \frac{a_{21}u_i + a_{22}v_i + a_{23}}{a_{31}u_i + a_{32}v_i + 1}$$

Lec19 Required Reading

- House, Ch. 10