CS559: Computer Graphics

Lecture 6: Edge Detection, Image Compositing, and Warping

Li Zhang

Spring 2010

Last time: Image Resampling and Painterly Rendering

- Box filter
  - Simple and cheap
- Tent filter
  - Linear interpolation
- Gaussian filter
  - Very smooth antialiasing filter
- B-spline cubic
  - Very smooth
- Catmull-rom cubic
  - Interpolating
- Mitchell-Netravali cubic
  - Good for image upsampling

Last time: Image Resampling and Painterly Rendering

- For some filters, the reconstruction process winds up implementing a simple algorithm
- Box filter (radius 0.5): nearest neighbor sampling

  box always catches exactly one input point

  it is the input point nearest the output point

  so output[*i*, *j*] = input[round(*x*(*i*)), round(*y*(*j*))]
    x(*i*) computes the position of the output coordinate *i* on the input grid
- Tent filter (radius 1): linear interpolation

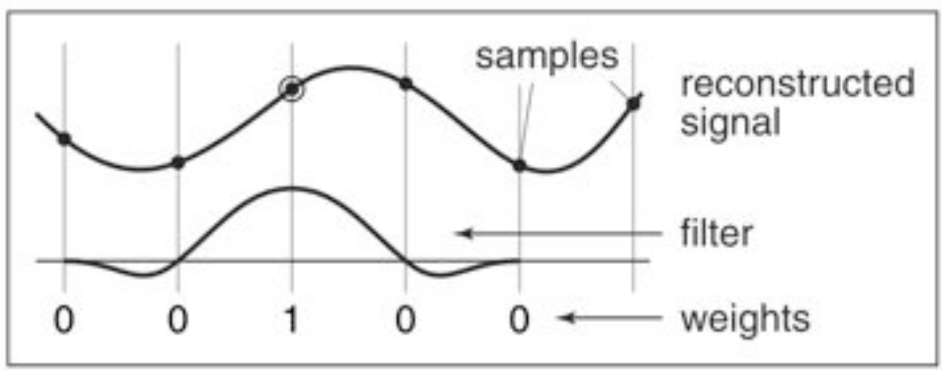
  tent catches exactly 2 input points

  weights are *a* and (1 – *a*)

  result is straight-line interpolation from one point to the next

Properties of Kernels

- Filter, Impulse Response, or kernel function, same concept but different names
- Degrees of continuity
- Interpolating or no
- Ringing or overshooting

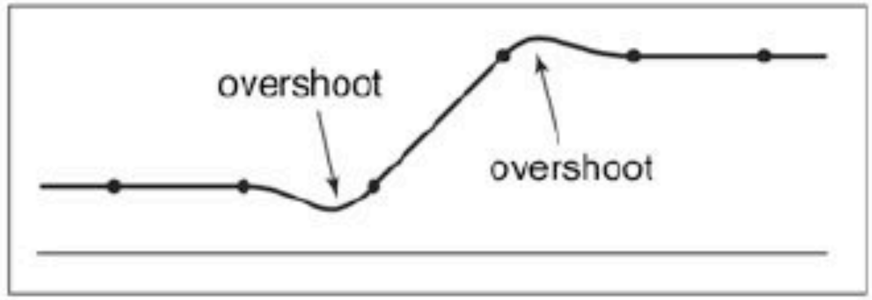


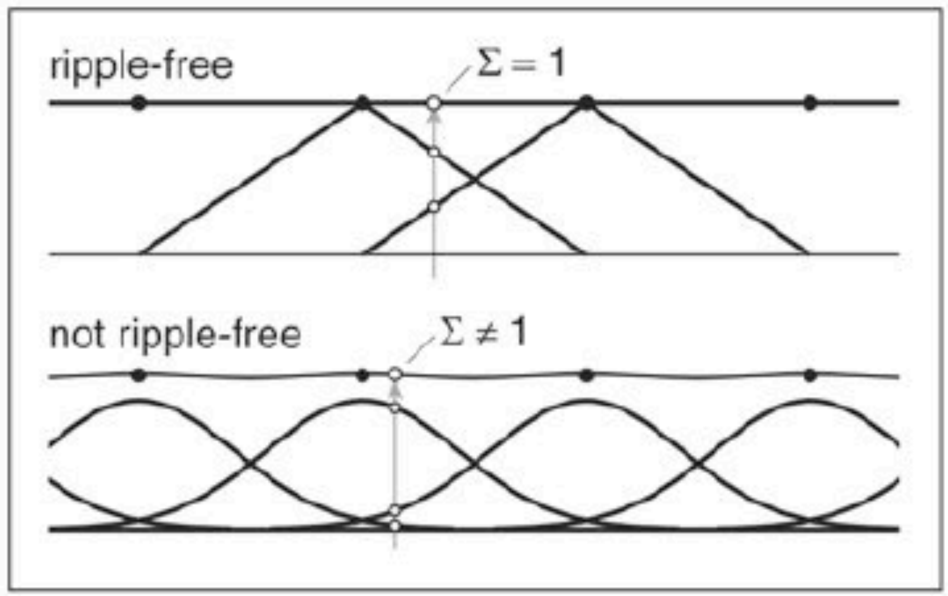
Interpolating filter for reconstruction

Ringing, overshoot, ripples

- Overshoot
  - caused by
    - negative filter values
- Ripples
  - constant in,
    - non-const. out
  - ripple free when:

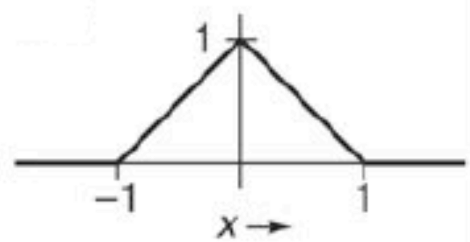
$$\sum_i f(x+i) = 1 \quad \text{for all } x.$$

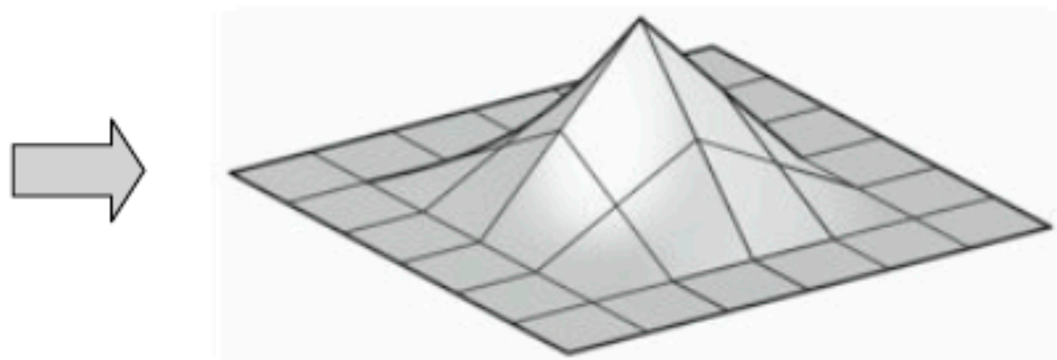


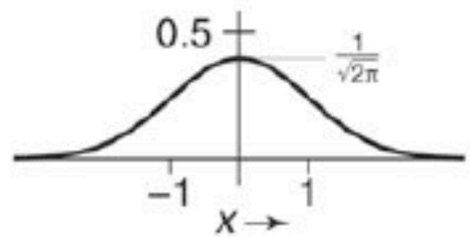


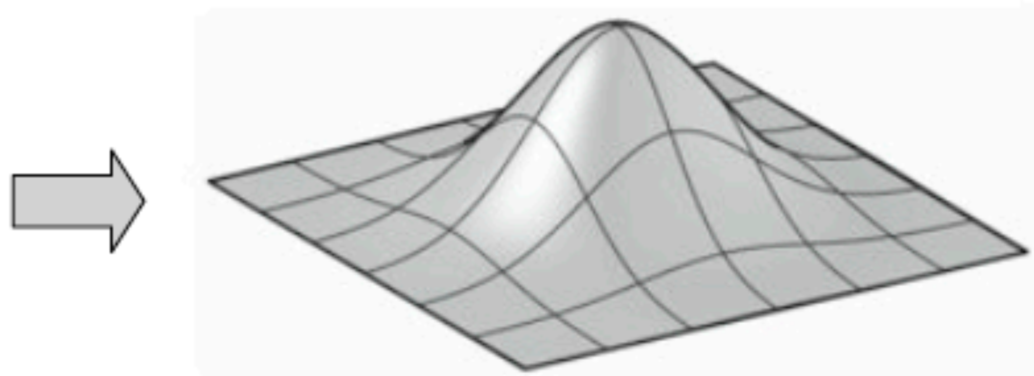
Constructing 2D filters

- Separable filters (most common approach)



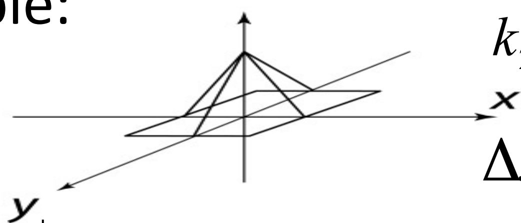


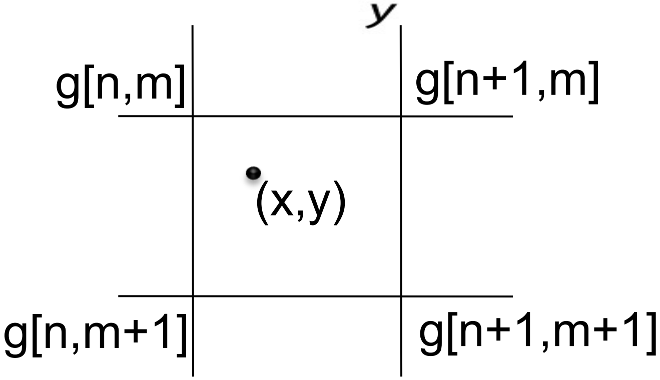




Reconstruction filter Examples in 2D

2D example:





$$k_{2D}(x,y) = \begin{cases} (1-|x|)(1-|y|) & |x|<1, |y|<1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta x = x - n, \Delta y = y - m$$

$$\begin{aligned} f(x,y) = & \ g[n,m]\cdot(1-\Delta x)\cdot(1-\Delta y) \\ & + g[n+1,m]\cdot \Delta x \cdot (1-\Delta y) \\ & + g[n,m+1]\cdot(1-\Delta x)\cdot \Delta y \\ & + g[n+1,m+1]\cdot \Delta x \cdot \Delta y \end{aligned}$$

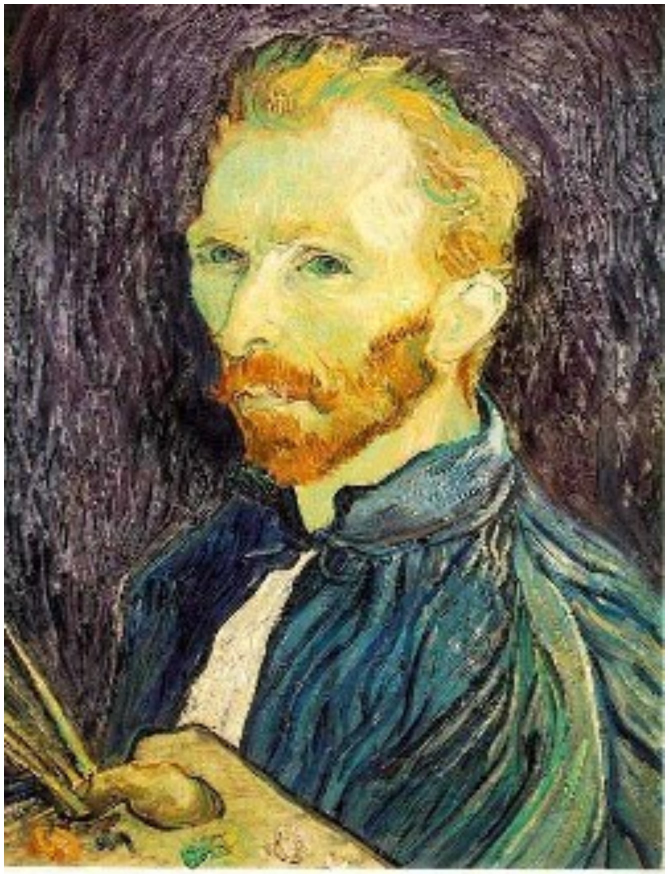
How to simplify the calculation?

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:

[Philip Greenspun]

Image Downsampling

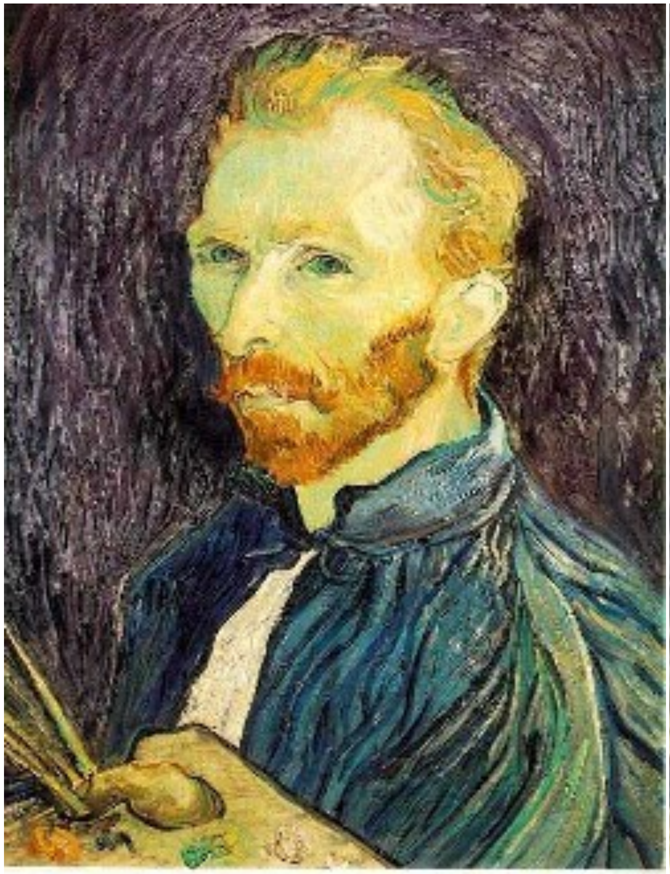


1/4

1/8

Throw away every other row and column to create a 1/2 size image
- called *image sub-sampling*

Image sub-sampling



1/2

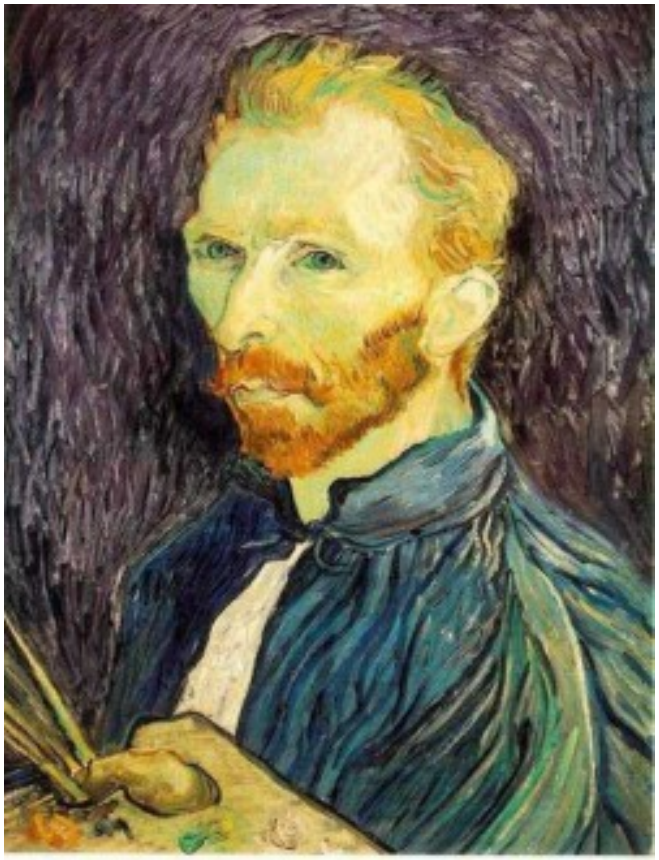
1/4 (2x zoom)

1/8 (4x zoom)

Why does this look so crufty?

Minimum Sampling requirement is not satisfied – resulting in **Aliasing effect**

Subsampling with Gaussian pre-filtering



Gaussian 1/2

G 1/4

G 1/8

- Solution: filter the image, *then* subsample

Results in the paper

Original

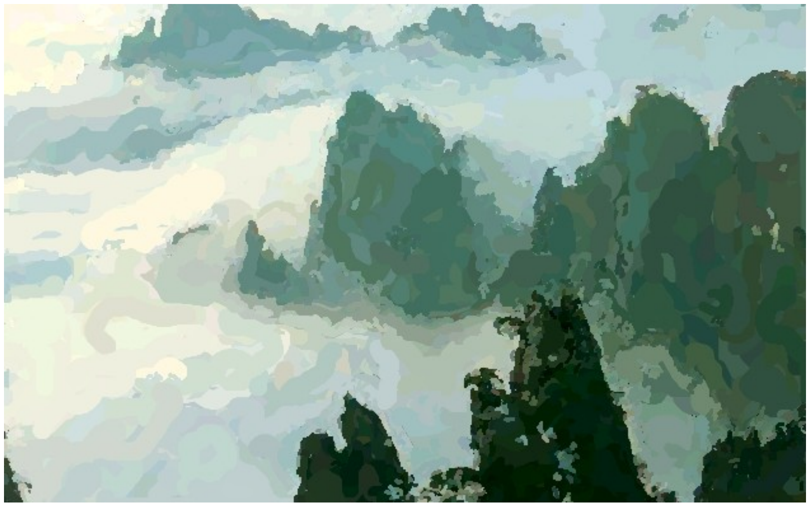
Biggest brush

Medium brush added

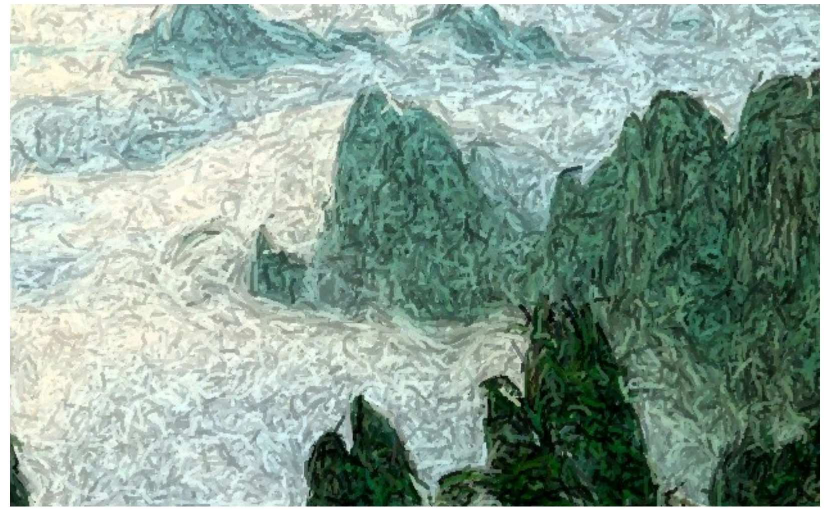
Finest brush added

Changing Parameters

Changing Parameters

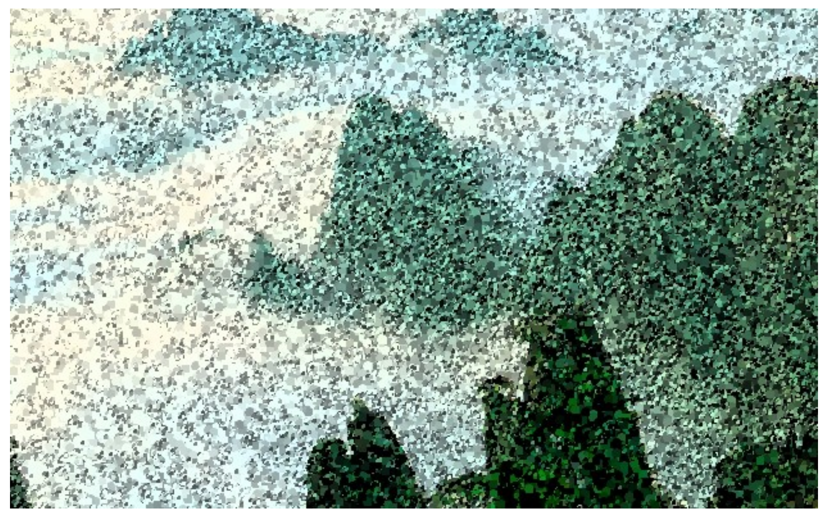


Impressionist, normal painting style



Expressionist, elongated stroke

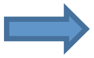
Colorist wash, semitransparent stroke with color jitter



Densely-placed circles with random hue and saturation

Another type of painterly rendering

- Line Drawing



http://www.cs.rutgers.edu/~decarlo/abstract.html

Another type of painterly rendering

- Line Drawing

http://www.cs.rutgers.edu/~decarlo/abstract.html

Another type of painterly rendering

- Line Drawing

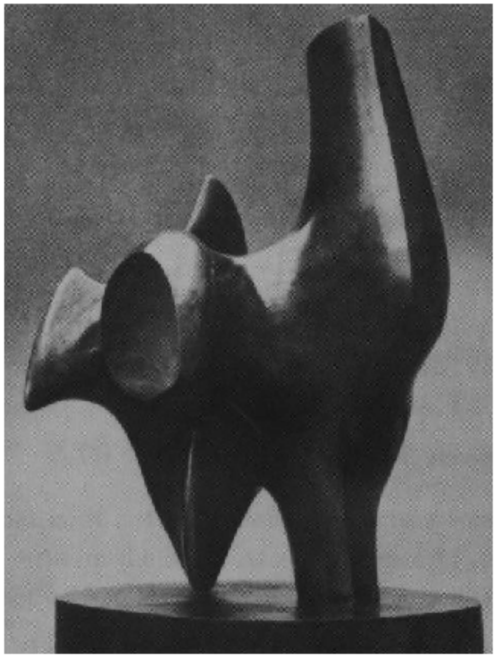
http://www.cs.rutgers.edu/~decarlo/abstract.html

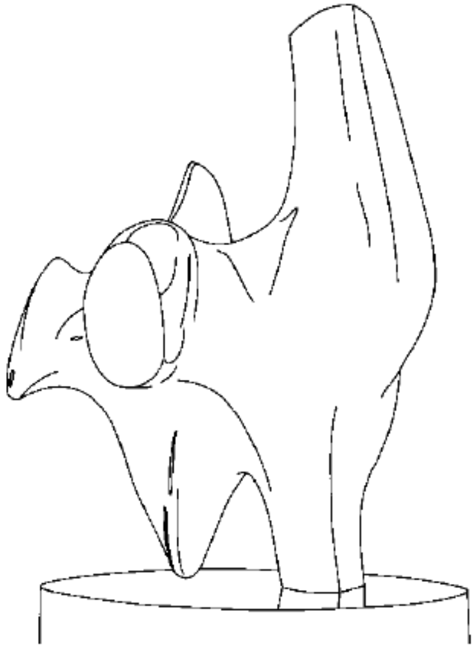
Another type of painterly rendering

- Line Drawing

http://www.cs.rutgers.edu/~decarlo/abstract.html

Edge Detection





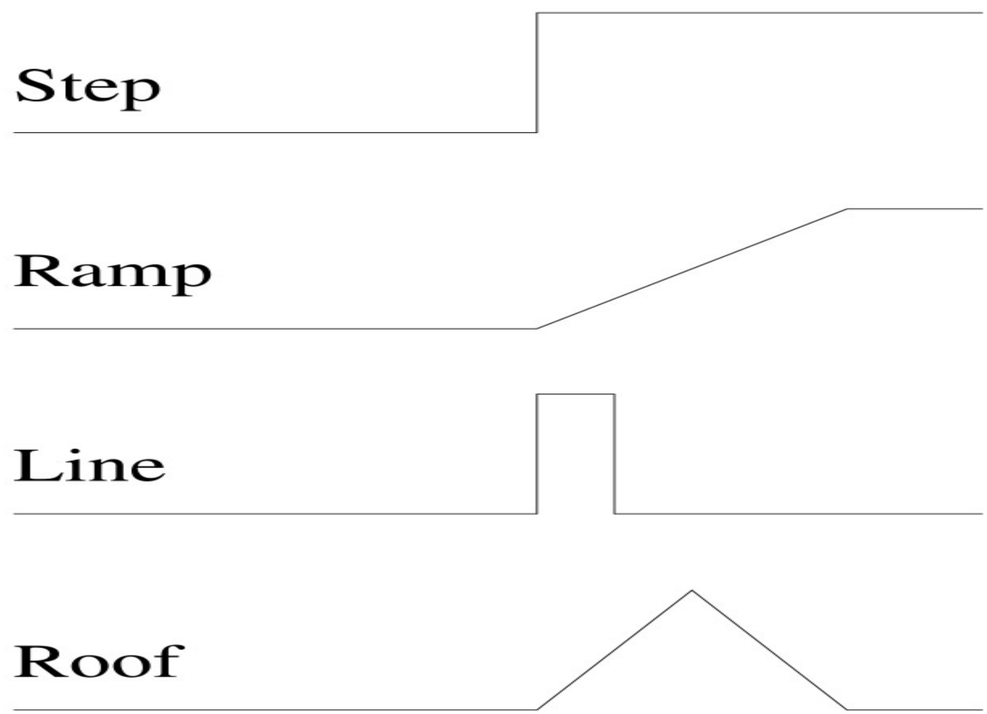
- Convert a 2D image into a set of curves
  - Extracts salient features of the scene

Edge detection

- One of the most important uses of image processing is **edge detection:**
  - Really easy for humans
  - Not that easy for computers

  - Fundamental in computer vision
  - Important in many graphics applications

What is an edge?



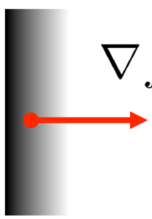
- **Q**: How might you detect an edge in 1D?

Image gradient

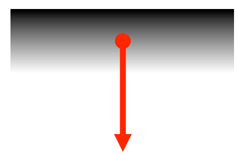
- The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

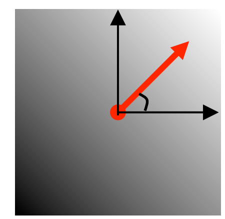
- The gradient points in the direction of most rapid change in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

  - how does the gradient relate to the direction of the edge?

- The *edge strength* is given by the gradient magnitude

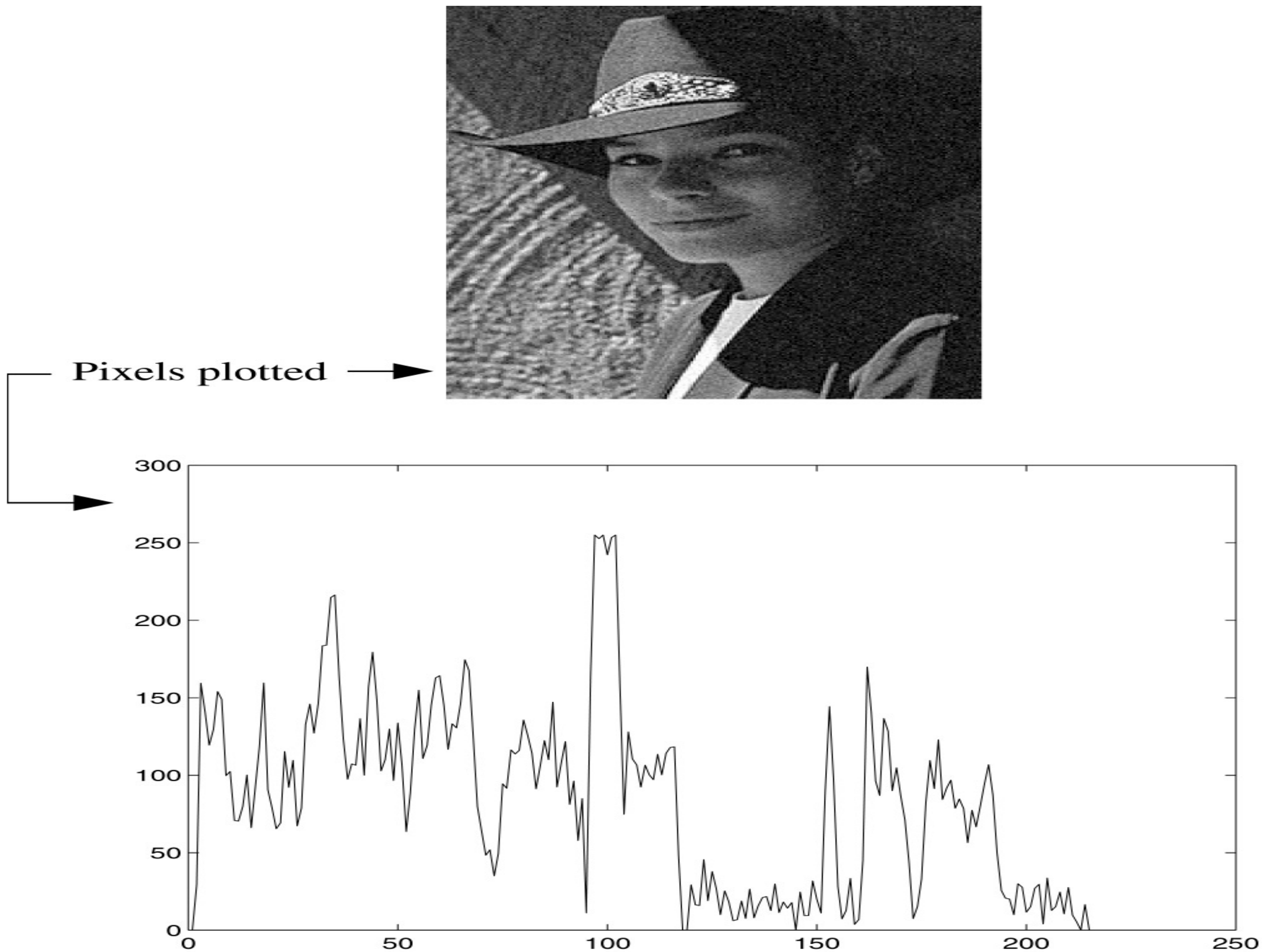
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Gradients

- How can we approximate the gradient in a discrete image?

  gx[i,j] = f[i+1,j] – f[i,j] and gy[i,j]=f[i,j+1]-f[i,j]
  Can write as mask [-1 1] and [1 –1]’

Less than ideal edges



Results of Sobel edge detection

Original

Smoothed

Edge enhancement

- A popular gradient magnitude computation is the **Sobel operator**:

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

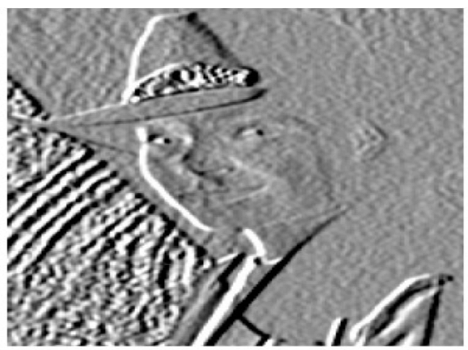
$$s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- We can then compute the magnitude of the vector (s_x, s_y).

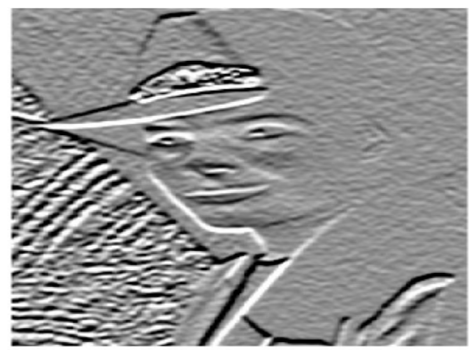
Results of Sobel edge detection

Original

Smoothed



Sx + 128



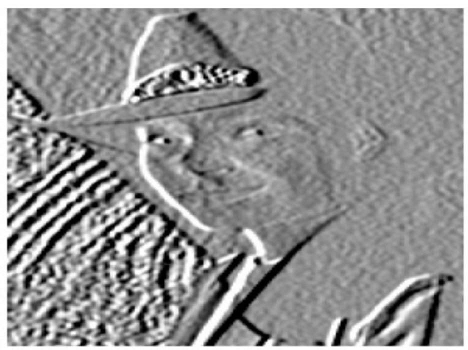
Sy + 128

Magnitude

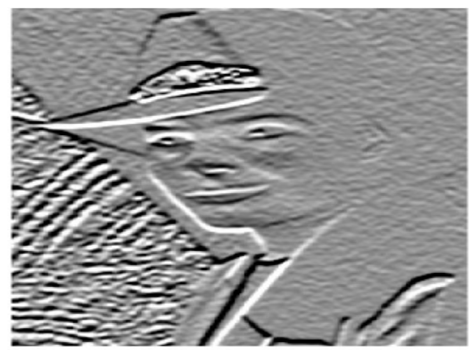
Results of Sobel edge detection

Original

Smoothed



Sx + 128



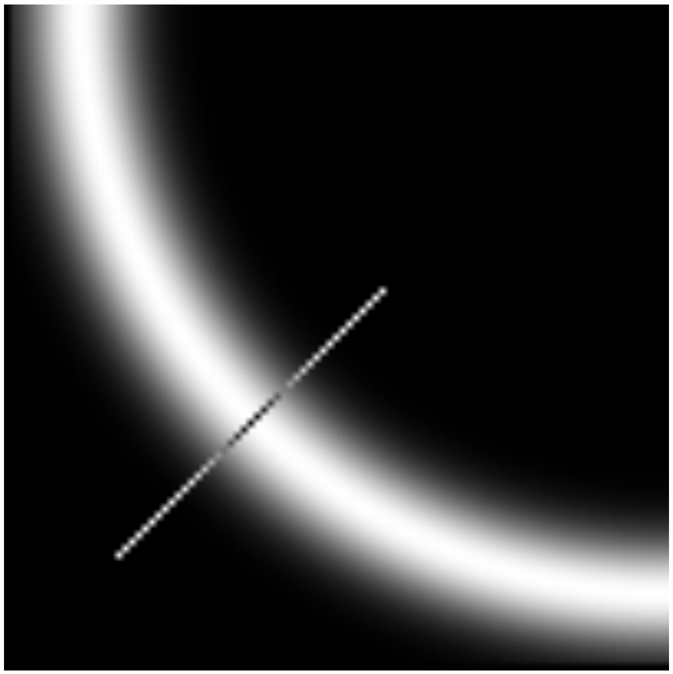
Sy + 128

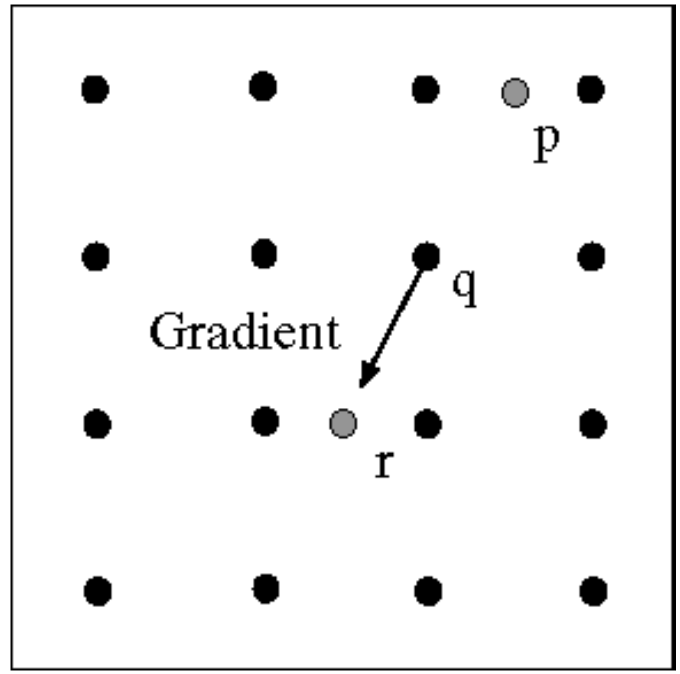
Magnitude

Threshold = 64

Threshold = 128

Non-maximum Suppression





- Check if pixel is local maximum along gradient direction
  - requires checking interpolated pixels p and r

The Canny Edge Detector

Steps in edge detection

- Edge detection algorithms typically proceed in three or four steps:
  - **Filtering**: cut down on noise
  - **Enhancement**: amplify the difference between edges and non-edges
  - **Detection**: use a threshold operation
  - **Localization** (optional): estimate geometry of edges, which generally pass between pixels

The Canny Edge Detector

original image (Lena)

The Canny Edge Detector

magnitude of the gradient

The Canny Edge Detector

After non-maximum suppression

Canny Edge Detector

original　　Canny with $\sigma = 1$　　Canny with

: Gaussian filter parameter

- The choice of depends on desired behavior
  - large detects large scale edges
  - small detects fine features

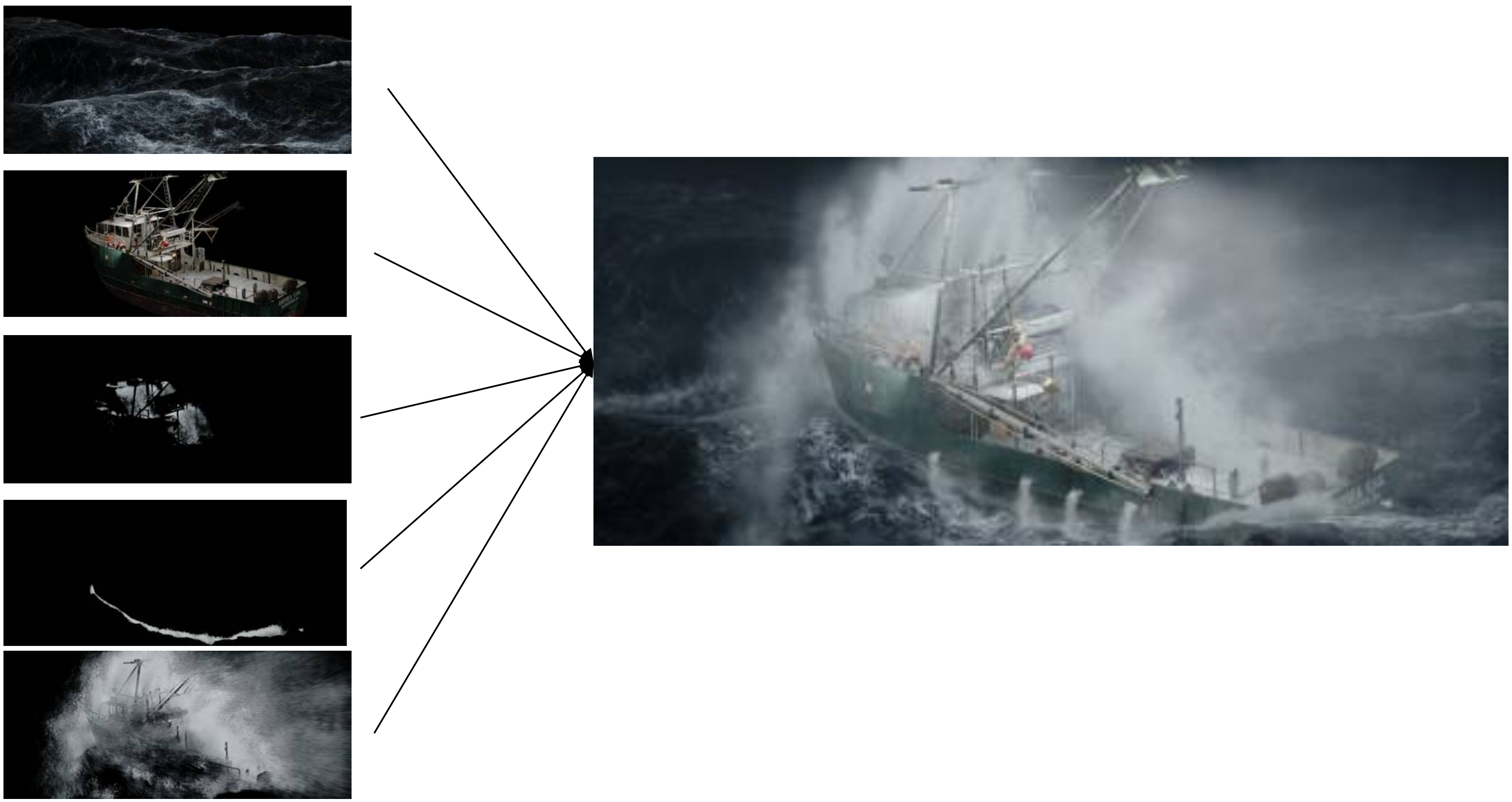
Compositing

- Compositing combines components from two or more images to make a new image
  - Special effects are easier to control when done in isolation
  - Even many all live-action sequences are more safely shot in different layers

Compositing

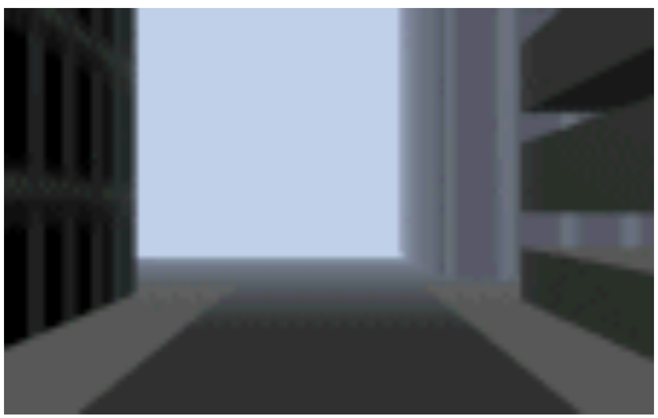
- Compositing combines components from two or more images to make a new image
  - Special effects are easier to control when done in isolation
  - Even many all live-action sequences are more safely shot in different layers

Perfect Storm

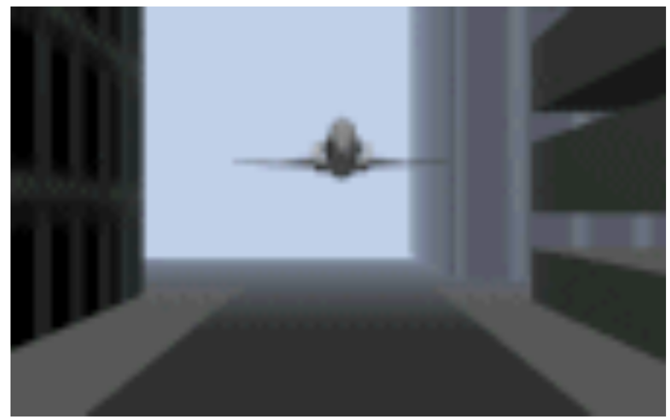


Animated Example

over



=



Mattes

- A *matte* is an image that shows which parts of another image are foreground objects
- Term dates from film editing and cartoon production
- How would I use a matte to insert an object into a background?
- How are mattes usually generated for television?

Working with Mattes

- To insert an object into a background
  - Call the image of the object the source
  - Put the background into the destination
  - For all the source pixels, if the matte is white, copy the pixel, otherwise leave it unchanged

Working with Mattes

- To insert an object into a background
  - Call the image of the object the source
  - Put the background into the destination
  - For all the source pixels, if the matte is white, copy the pixel, otherwise leave it unchanged
- To generate mattes:
  - Use smart selection tools in Photoshop or similar
    - They outline the object and convert the outline to a matte
  - **Blue Screen:** Photograph/film the object in front of a blue background, then consider all the blue pixels in the image to be the background

Compositing

- Compositing is the term for combining images, one over the other
  - Used to put special effects into live action
    - Or live action into special effects

Alpha

- Basic idea: Encode opacity information in the image
- Add an extra channel, the *alpha* channel, to each image
  - For each pixel, store R, G, B and Alpha
  - alpha = 1 implies full opacity at a pixel
  - alpha = 0 implies completely clear pixels
- Images are now in RGBA format, and typically 32 bits per pixel (8 bits for alpha)
- All images in the project are in this format

Pre-Multiplied Alpha

- Instead of storing (R,G,B,α), store $(\alpha R,\alpha G,\alpha B,\alpha)$
- The compositing operations in the next several slides are easier with pre-multiplied alpha
- **To display and do color conversions, must extract RGB by dividing out α**
  - $\alpha=0$ is always black
  - Some loss of precision as α gets small, but generally not a big problem

Why do we need pre-multiplied alpha?

$$c_g = \begin{bmatrix} R_g \\ G_g \\ B_g \\ 1 \end{bmatrix} \qquad c_f = \begin{bmatrix} R_f \\ G_f \\ B_f \\ \alpha_f \end{bmatrix} \qquad c_o = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$$c_o = 1 \cdot c_f + (1 - \alpha_f) \cdot c_g$$

"Over" Operator

Why do we need pre-multiplied alpha?

$$c_g = \begin{bmatrix} R_g \\ G_g \\ B_g \\ 1 \end{bmatrix} \quad c_f = \begin{bmatrix} R_f \\ G_f \\ B_f \\ \alpha_f \end{bmatrix} \quad c_o = \begin{bmatrix} \alpha_f R_f + (1-\alpha_f) R_g \\ \alpha_f G_f + (1-\alpha_f) G_g \\ \alpha_f B_f + (1-\alpha_f) B_g \\ \end{bmatrix}$$

$$c_o = 1 \cdot c_f + (1-\alpha_f) \cdot c_g$$

"Over" Operator

Why do we need pre-multiplied alpha?

$$c_g = \begin{bmatrix} R_g \\ G_g \\ B_g \\ 1 \end{bmatrix} \qquad c_f = \begin{bmatrix} R_f \\ G_f \\ B_f \\ \alpha_f \end{bmatrix} \qquad c_o = \begin{bmatrix} \alpha_f R_f + (1-\alpha_f)R_g \\ \alpha_f G_f + (1-\alpha_f)G_g \\ \alpha_f B_f + (1-\alpha_f)B_g \\ 1 \end{bmatrix}$$

$$c_o = 1 \cdot c_f + (1-\alpha_f) \cdot c_g$$

“Over” Operator

Why do we need pre-multiplied alpha?

$$c_g = \begin{bmatrix} R_g \\ G_g \\ B_g \\ \alpha_g \end{bmatrix} \qquad c_g = \begin{bmatrix} R_f \\ G_f \\ B_f \\ \alpha_f \end{bmatrix} \qquad c_o = \begin{bmatrix} \alpha_f R_f + (1-\alpha_f) R_g \\ \alpha_f G_f + (1-\alpha_f) G_g \\ \alpha_f B_f + (1-\alpha_f) B_g \\ 1 \end{bmatrix}$$

$$c_o = 1 \cdot c_f + (1-\alpha_f) \cdot c_g$$

“Over” Operator

Why do we need pre-multiplied alpha?

$$c_g = \begin{bmatrix} R_g \\ G_g \\ B_g \\ \alpha_g \end{bmatrix} \quad c_g = \begin{bmatrix} R_f \\ G_f \\ B_f \\ \alpha_f \end{bmatrix} \quad c_o = \begin{bmatrix} \alpha_f R_f + (1-\alpha_f)\alpha_g R_g \\ \alpha_f G_f + (1-\alpha_f)\alpha_g G_g \\ \alpha_f B_f + (1-\alpha_f)\alpha_g B_g \\ \end{bmatrix}$$

$$c_o = 1 \cdot c_f + (1-\alpha_f) \cdot c_g$$

"Over" Operator

Why do we need pre-multiplied alpha?

$$c_g = \begin{bmatrix} R_g \\ G_g \\ B_g \\ \alpha_g \end{bmatrix} \qquad c_g = \begin{bmatrix} R_f \\ G_f \\ B_f \\ \alpha_f \end{bmatrix} \qquad c_o = \begin{bmatrix} \alpha_f R_f + (1-\alpha_f)\alpha_g R_g \\ \alpha_f G_f + (1-\alpha_f)\alpha_g G_g \\ \alpha_f B_f + (1-\alpha_f)\alpha_g B_g \\ \alpha_f + (1-\alpha_f)\alpha_g \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_f R_f \\ \alpha_f G_f \\ \alpha_f B_f \\ \alpha_f \end{bmatrix} + (1-\alpha_f) \begin{bmatrix} \alpha_g R_g \\ \alpha_g G_g \\ \alpha_g B_g \\ \alpha_g \end{bmatrix}$$

$$c_o = 1 \cdot c_f + (1-\alpha_f) \cdot c_g$$

“Over” Operator

Basic Compositing Operation

- At each pixel, combine the pixel data from *f* and the pixel data from *g* with the equation:

$$c_f = [\alpha_f R_f, \alpha_f G_f, \alpha_f B_f, \alpha_f]$$

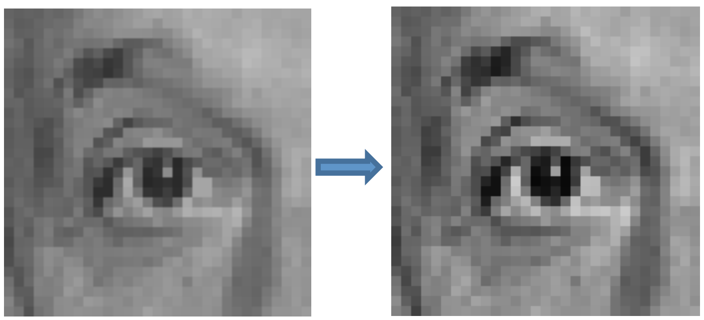
$$c_g = [\alpha_g R_g, \alpha_g G_g, \alpha_g B_g, \alpha_g]$$

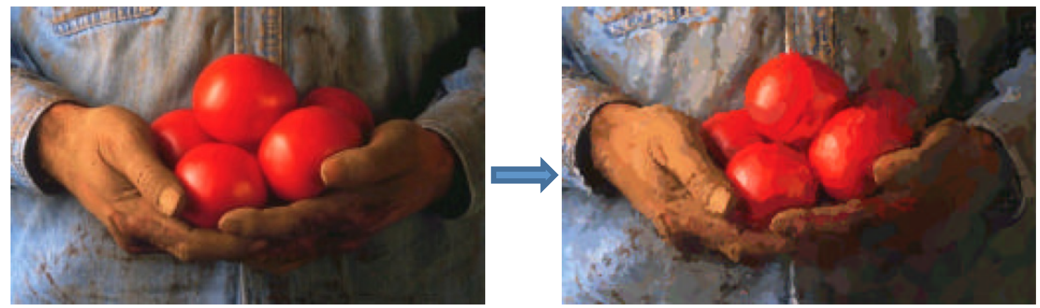
$$c_o = 1 \cdot c_f + (1 - \alpha_f) \cdot c_g$$

“Over” Operator

Image Manipulation

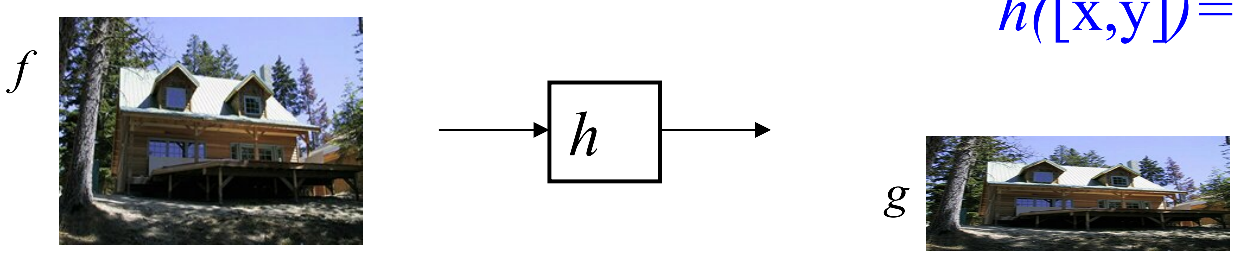
- Changing pixel values





- Moving pixels around

$$h([x,y])=[x,y/2]$$



Parametric (global) warping

Examples of parametric warps:

translation

rotation

aspect

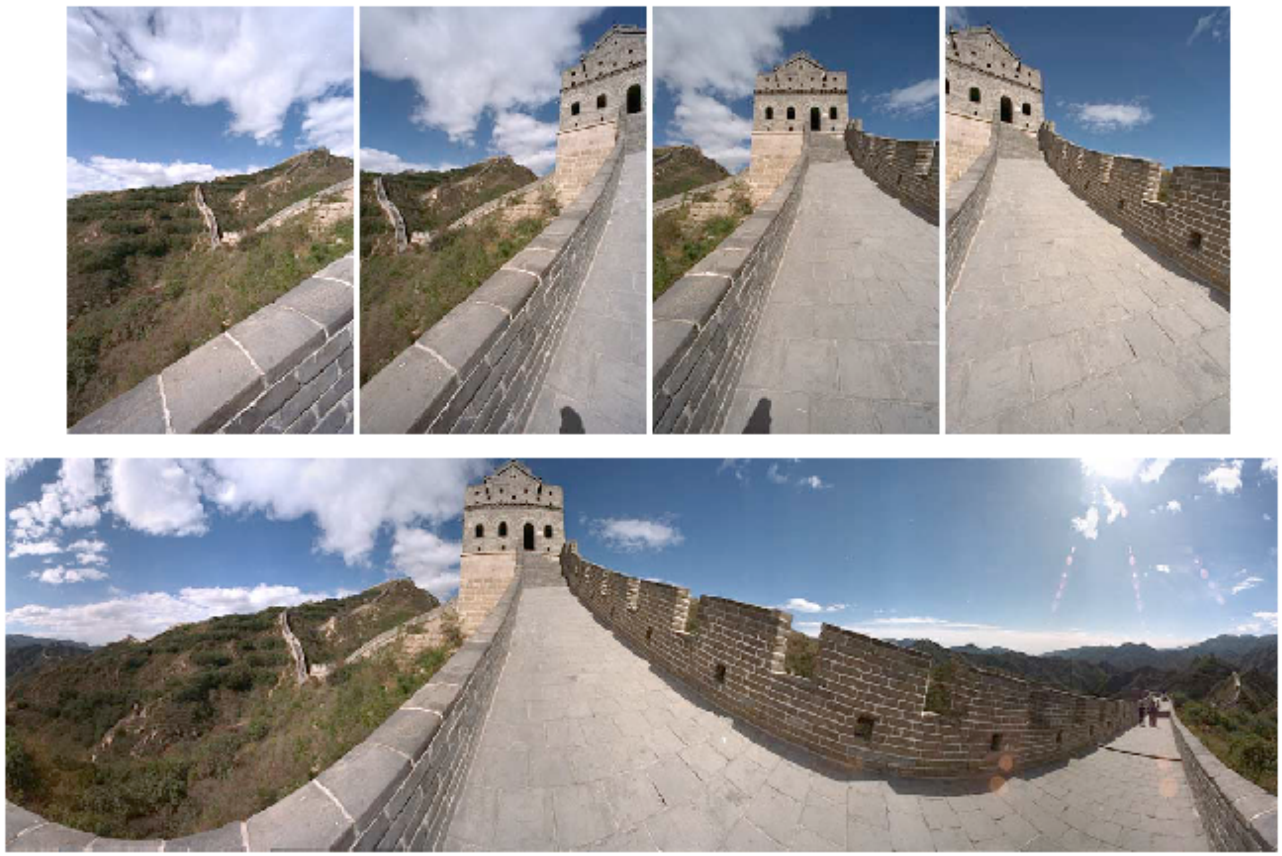
affine

perspective

cylindrical

Application of Image Warp

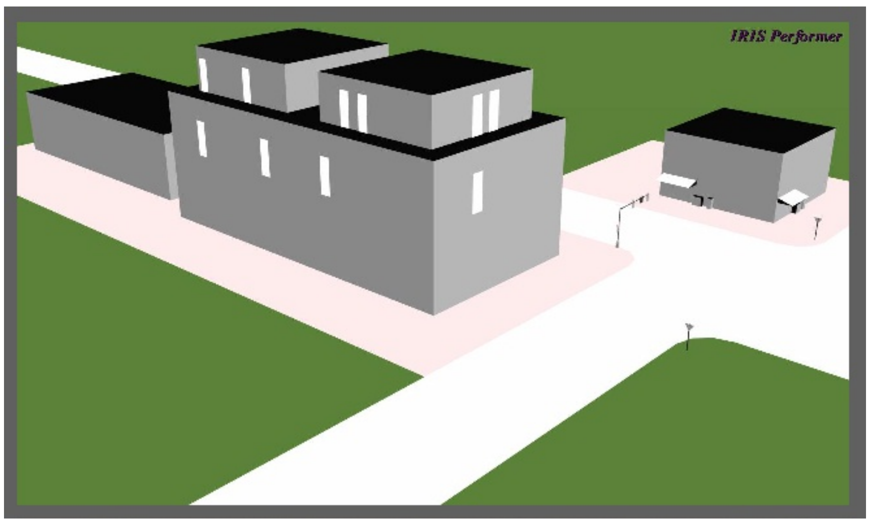
Mosaics: stitching images together

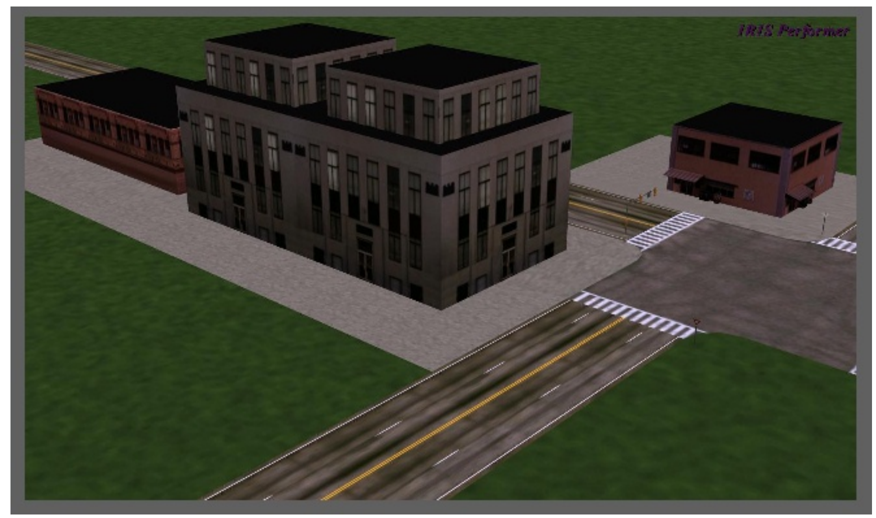


Creating virtual wide-angle camera

Application of Image Warp

Texture mapping





Creating realistic surface appearance

http://www.futuretech.blinkenlights.nl/tex.html

Application of Image Warp

- Morphing

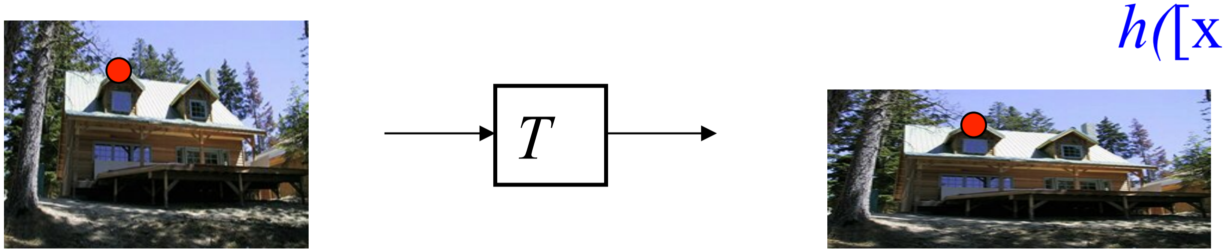
image #1

morphing

image #2

Parametric (global) warping

$h([x,y])=[x,y/2]$



p = (x,y) **p'** = (x',y')

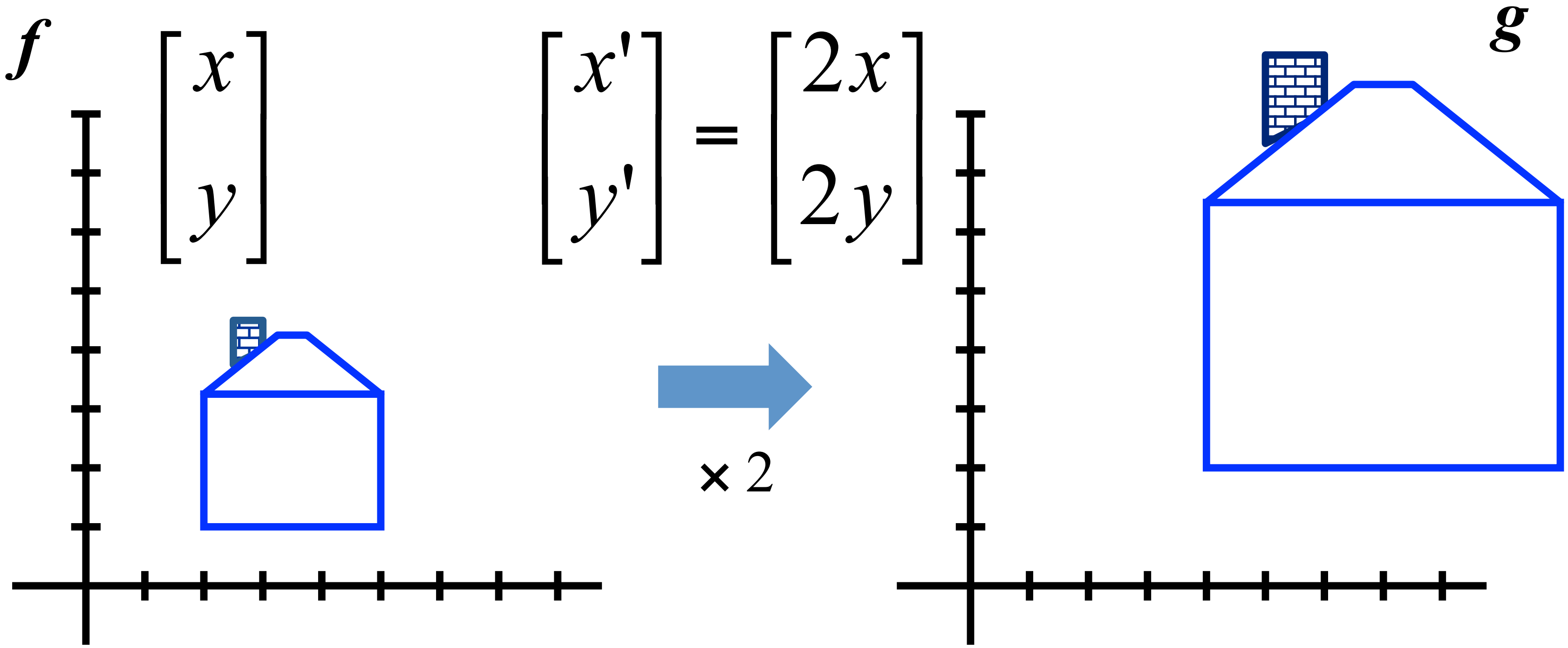
- Transformation T is a coordinate-changing machine: p' = *T*(p)
- What does it mean that *T* is global?
  - can be described by just a few numbers (parameters)
  - the parameters are the same for any point p
- Represent *T* as a matrix: p' = **M**p

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:

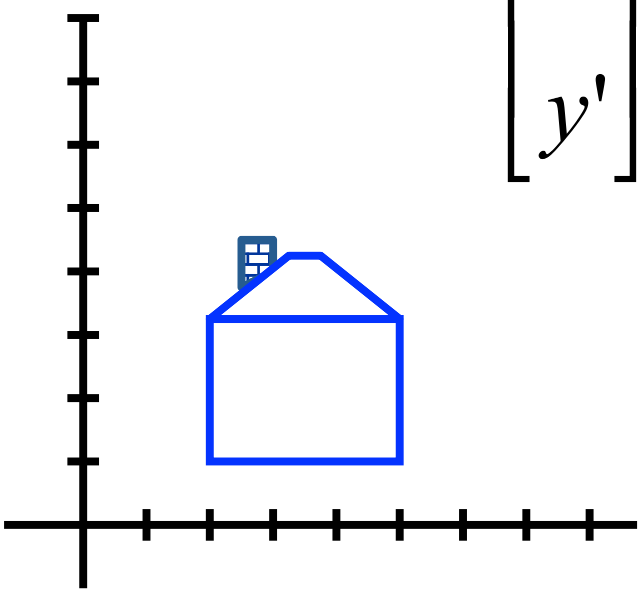
$$\begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$



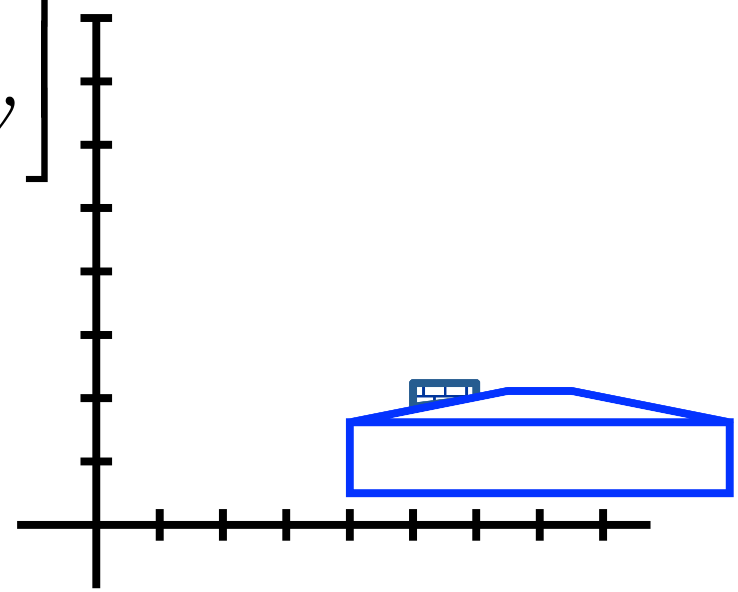
Scaling

- *Non-uniform scaling*: different scalars per component:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2x \\ 0.5y \end{bmatrix}$$



x × 2,
y × 0.5



Scaling

- Scaling operation: $x' = ax$

  $y' = by$

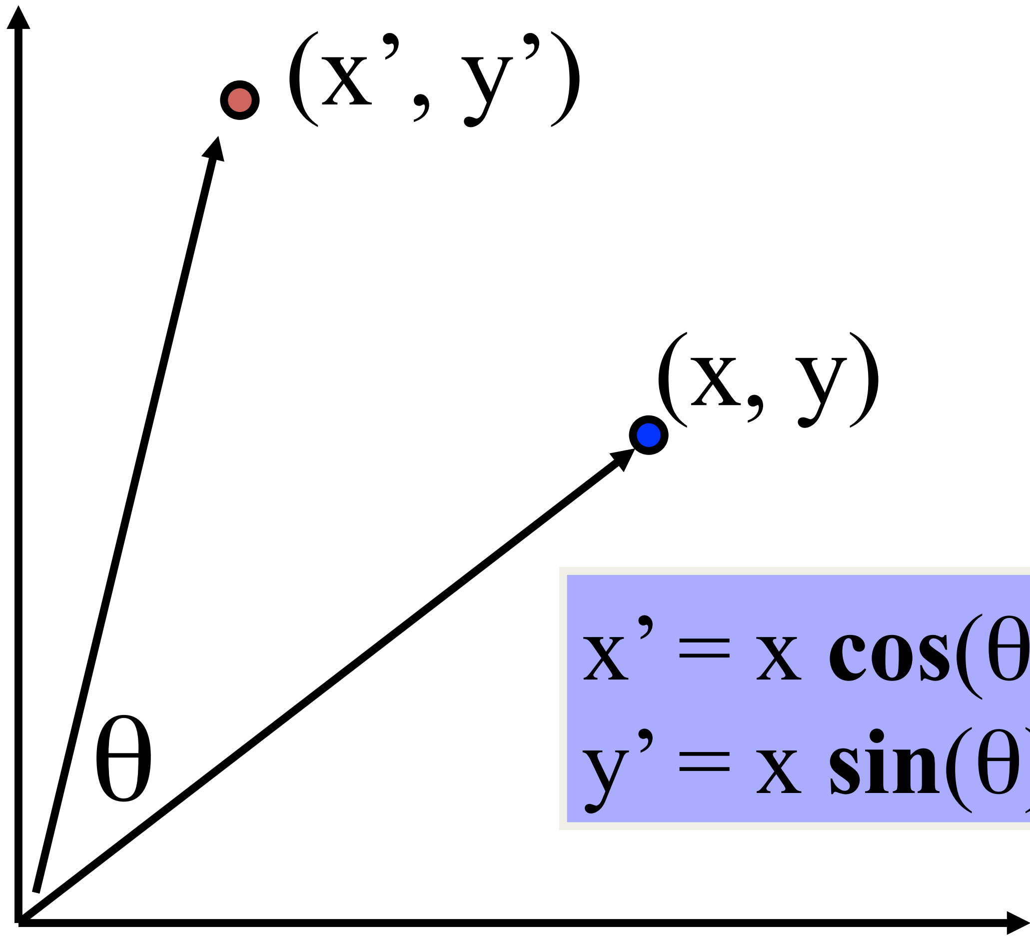
- Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\textit{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

What's inverse of S?

$$S^{-1} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/b \end{bmatrix}$$

2-D Rotation



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

2-D Rotation

- This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

x' = x **cos**(θ) - y **sin**(θ)
y' = x **sin**(θ) + y **cos**(θ)

- How can I remember this?
- Even though sin(θ) and cos(θ) are nonlinear to θ,
  - ***x' is a linear combination of x and y***
  - ***y' is a linear combination of x and y***
- What is the inverse transformation?
  - Rotation by –θ
  - For rotation matrices, det(R) = 1

$$\mathbf{R}^{-1} = \mathbf{R}^{T}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Identity?

$$x' = x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Scale around (0,0)?

$$\boldsymbol{x' = s_x * x}$$
$$\boldsymbol{y' = s_y * y}$$

$$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s_x} & 0 \\ 0 & \boldsymbol{s_y} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$x' = \cos\theta * x - \sin\theta * y$$
$$y' = \sin\theta * x + \cos\theta * y$$

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

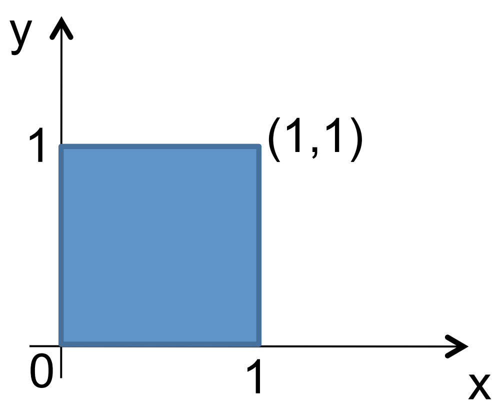
2D Shear?

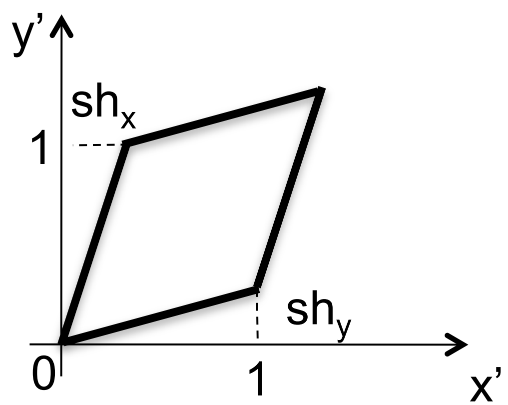
$$\boldsymbol{x' = x + sh_x * y}$$
$$\boldsymbol{y' = sh_y * x + y}$$

$$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh_x} \\ \boldsymbol{sh_y} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?





2D Shear?

$$x' = x + sh_x * y$$
$$y' = sh_y * x + y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Mirror about Y axis?

$$x' = -x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror over (0,0)?

$$x' = -x$$
$$y' = -y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Any 2D transform can be decomposed into the product of a rotation, scale, and a rotation

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = rotate(31.7^\circ) \cdot scale(1.618, 0.618) \cdot rotate(-58.3^\circ)$$

All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- A symmetric 2D transform can be decomposed into the product of a rotation, scale, and the inverse rotation

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = rotate(31.7^\circ) \cdot scale(2.618, 0.382) \cdot rotate(-31.7^\circ)$$