Computational Photography

Week 6

Instructor: Lou Kratz

Compositing

How does Superman fly?

- Super-human powers?
  OR
- Image Matting and Compositing?

Slide from Alyosha Efros

Motivation: Compositing

Combining multiple images. Typically, paste a foreground object onto a new background

- Movie special effect
- Multi-pass CG
- Combining CG & film
- Photo retouching
  - Change background
  - Fake depth of field
  - Page layout: extract objects, magazine covers

From Cinefex

Plate 94 *A composite image created for the film* Titanic.

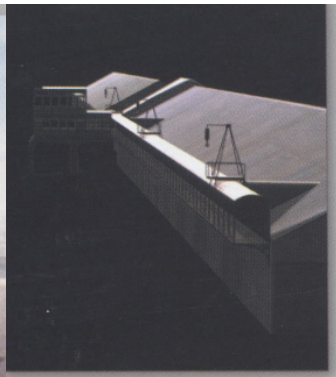


Plate 97 *A computer-generated dock element.*

Plate 98 *An element used to control the atmosphere on the dock.*

Plate 99 *An element featuring people that were on the ship.*

Plate 95 *An element that features a miniature of the ship.*

Plate 96 *An intermediate element that contains computer-generated water and an animated sky.*

Plate 100 *An element featuring a group of people on the dock.*

Plate 93 *The script used to create the image shown in Plate 94. Sections are magnified and detailed.*

From the Art & Science of Digital Compositing


Performing Songwriter
Be Heard
Annie Lennox and Dave Stewart
Eurythmics
THEY CALL ME MR. BIG
Notorious Rock Managers
MEMPHIS
Inside Soulsville, USA
STUDIO CONNECTIONS
Integrating Your Gear
+PLUS
Isaac Hayes • Lesley Gore • Chick Corea
Darden Smith • Bruce Cockburn
Sergio Mendes • Richard Julian
LOVE IS REAL
REFLECTIO
VOLUME 13, ISSUE 9
JAN./FEB. 2006


THE BIG CHILI CHALLENGE • VOTE! WIN! PAGE 24
BON APPÉTIT
MARCH 2006
Secrets to Great Soup
NEW COMBOS, FAB FLAVORS
QUICK PARTIES
Seafood & Pasta for 6
PAGE 44
St. Pat's in a Flash
PAGE 118
WARM GOOEY DESSERTS
Steakhouse Greats
BEYOND STEAK AND CREAMED SPINACH
(But we have those, too...)
Should You Give Up Foie Gras?
Build a better biscuit
page 40
(Hint: It's the cheddar)
FIT FOR WINTER
* LOW CAL
* LOW FAT
* HIGH FIBER
RECIPES, PAGE 118
$3.99
CANADA $4.99
FOREIGN $4.99
bonappetit.com

Photo Editing

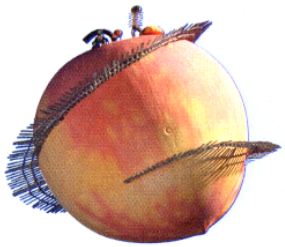
- Edit the background independently from foreground

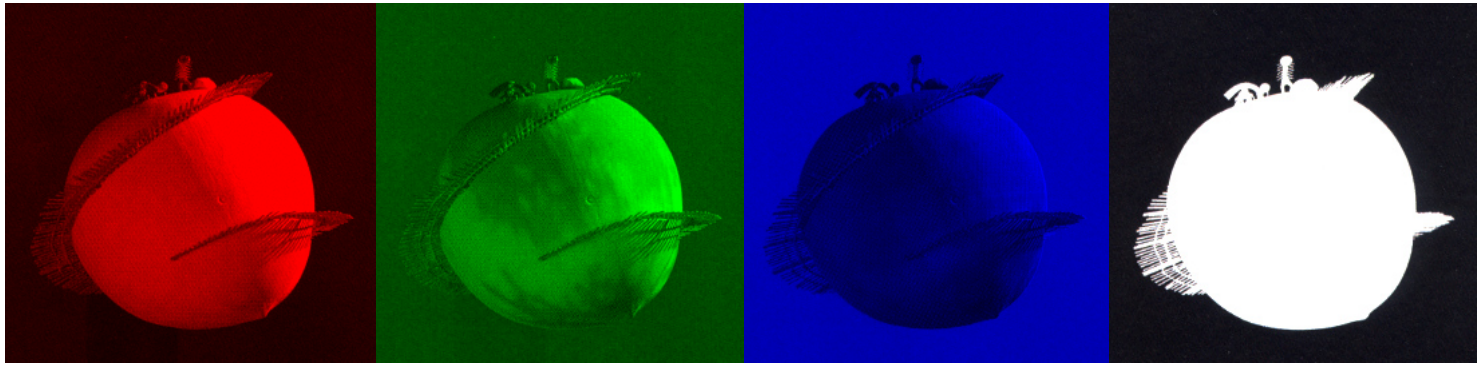
Technical Issues

- Smart selection
  - Facilitate the selection of an object
- Compositing
  - How exactly do we handle transparency?
- Matte extraction
  - Resolve sub-pixel accuracy, estimate transparency
- Smart pasting
  - Don't be smart with copy, be smart with paste
  - See also in a couple weeks (gradient manipulation)
- Extension to video
  - Where life is always harder

Alpha

- α: 1 means opaque, 0 means transparent
- 32-bit images: R, G, B,

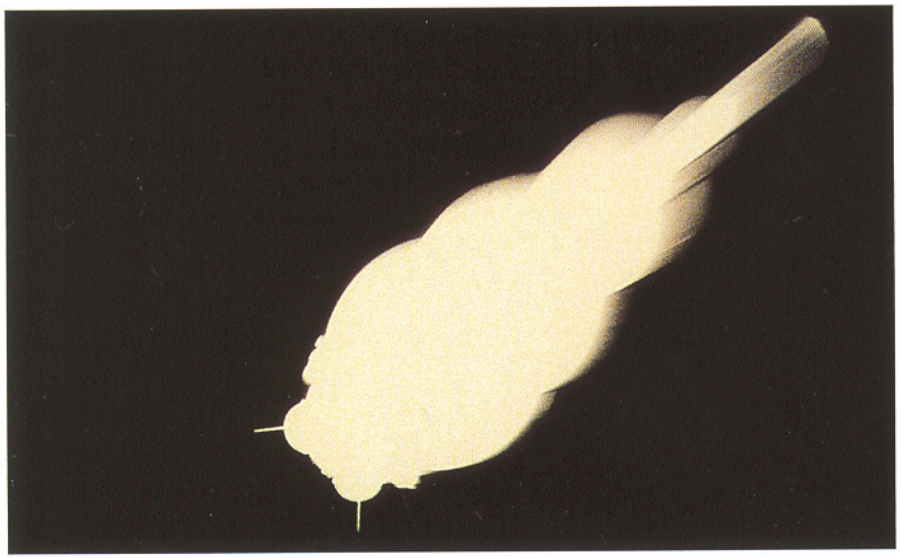


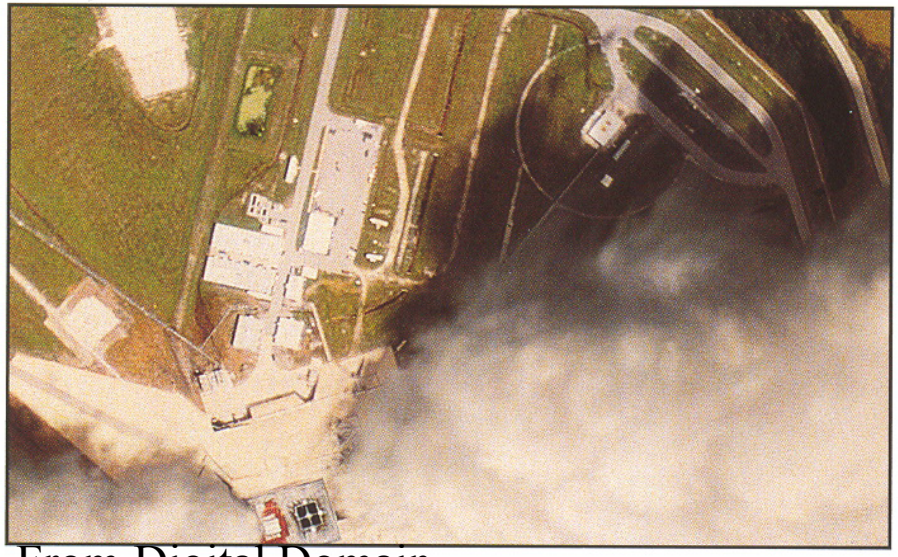


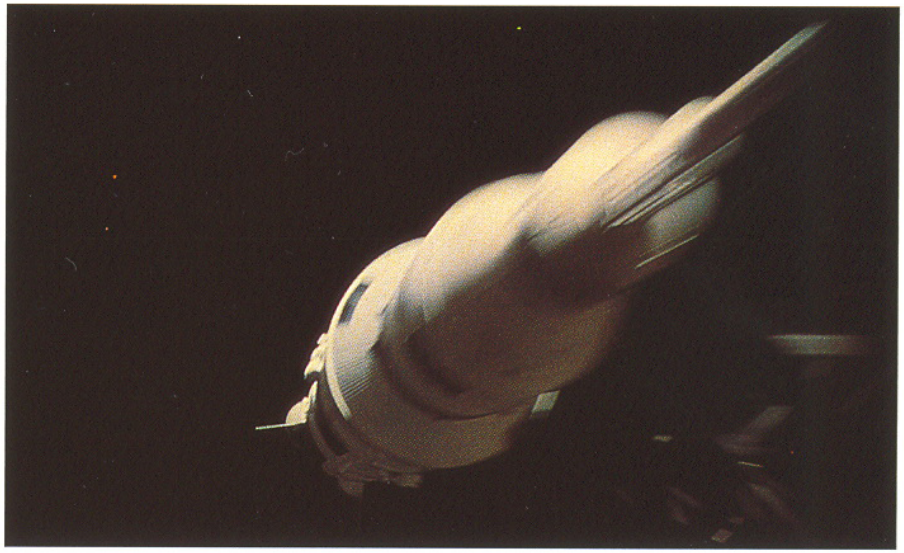
From the Art & Science of Digital Compositing

Why fractional alpha?

- Motion blur, small features (hair) cause partial occlusion

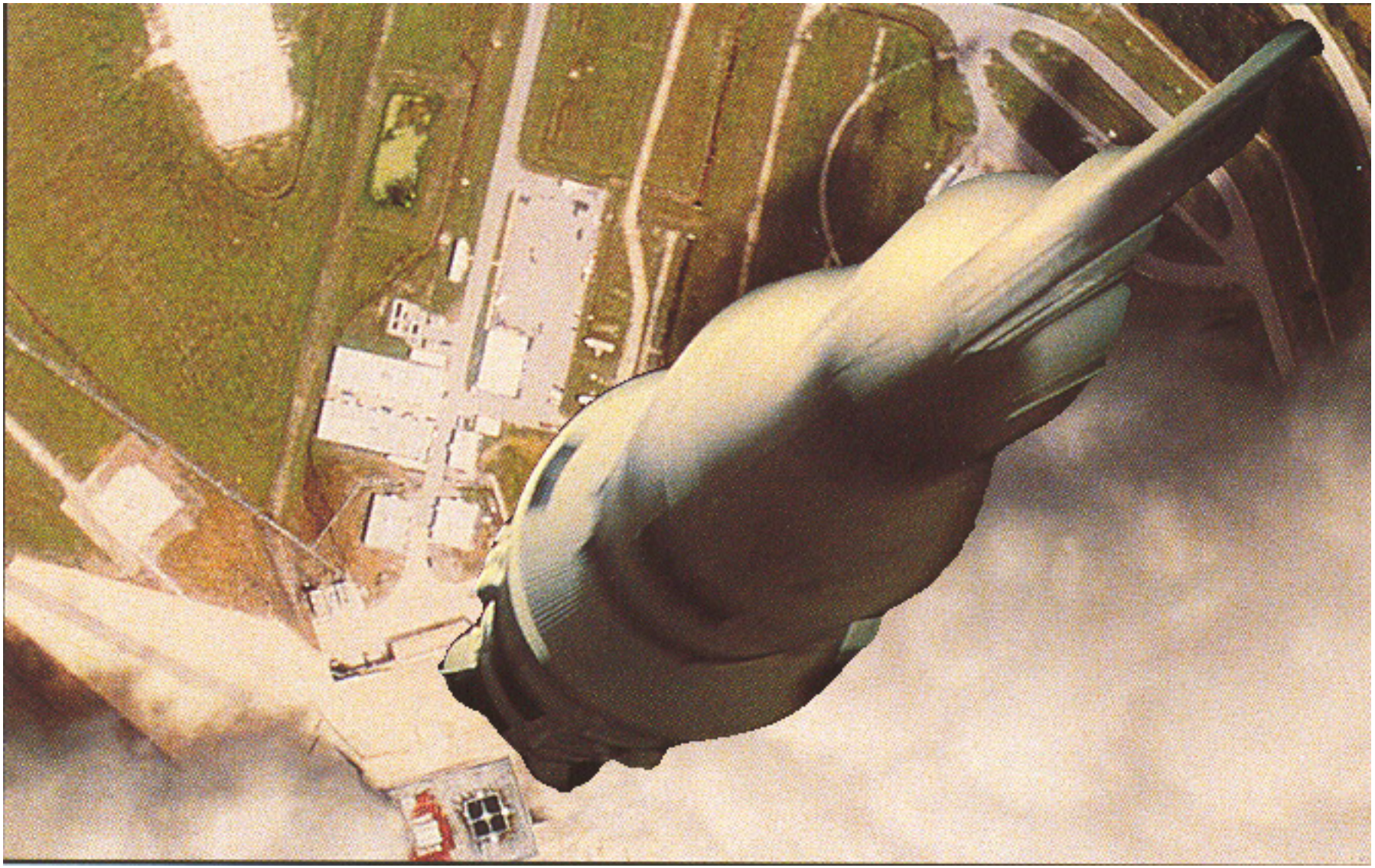






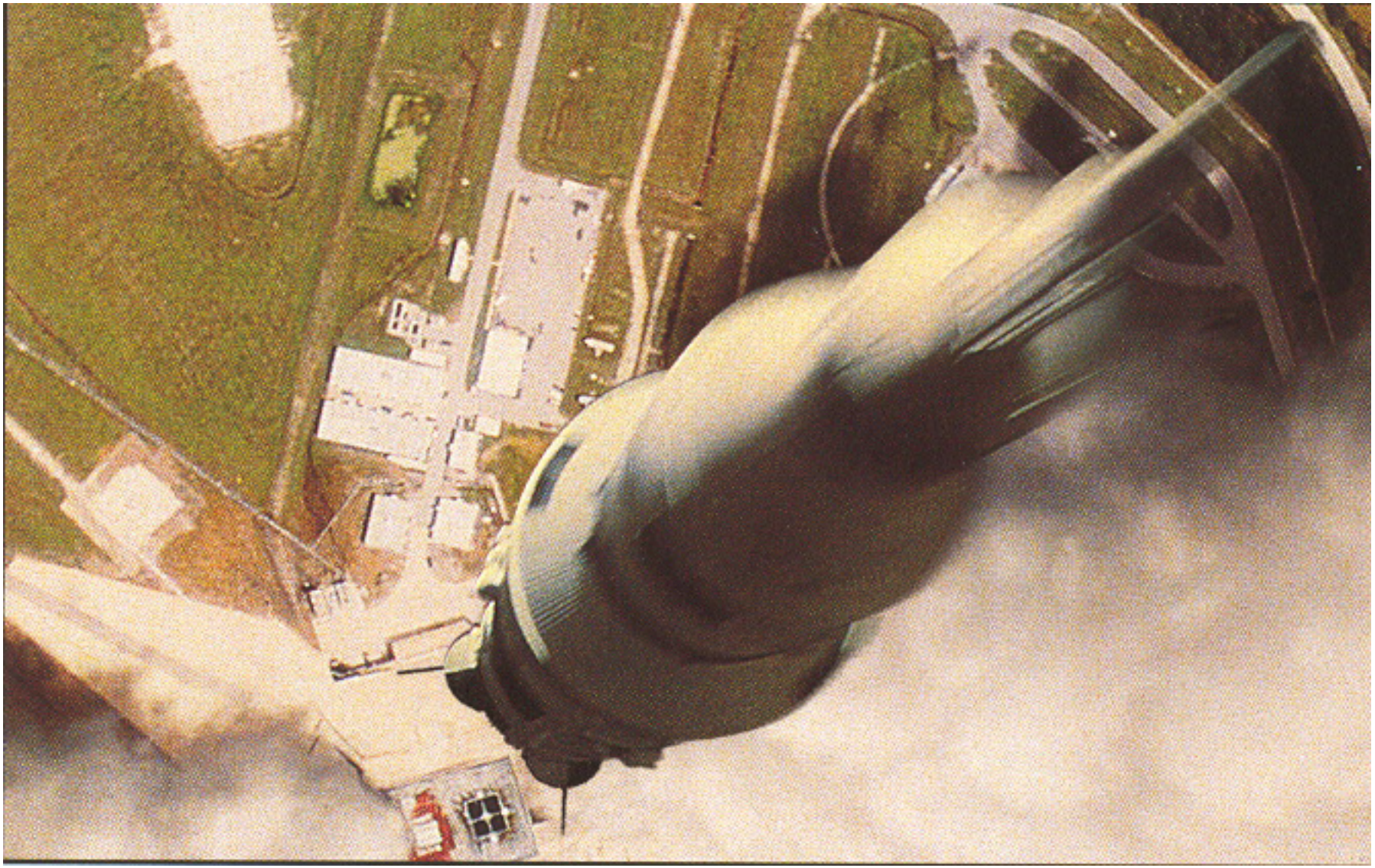
From Digital Domain

With Binary Alpha



From Digital Domain

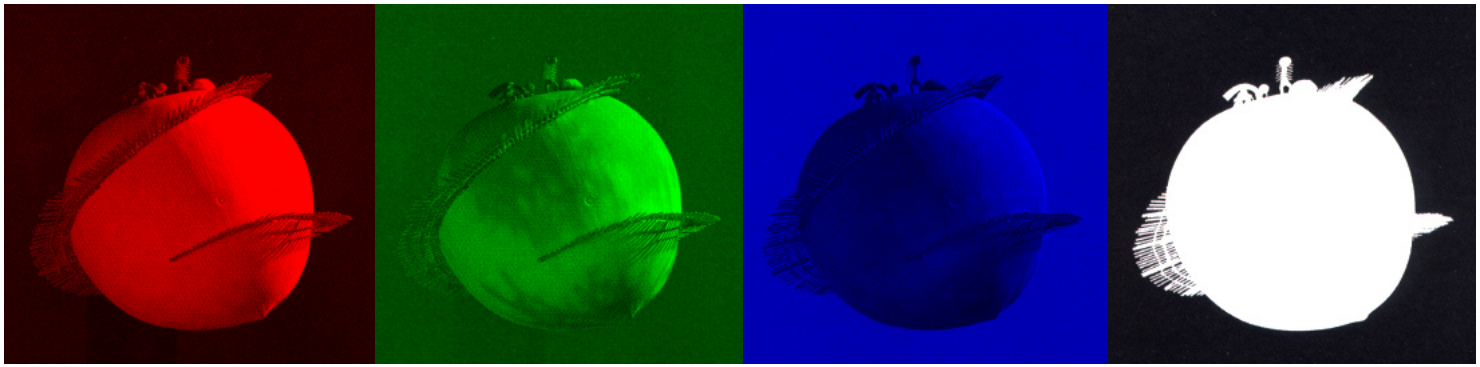
With Fractional Alpha



From Digital Domain

What does R, G, B, α represent?

- **α:** 1 means opaque, 0 means transparent
- But what about R, G, and B?

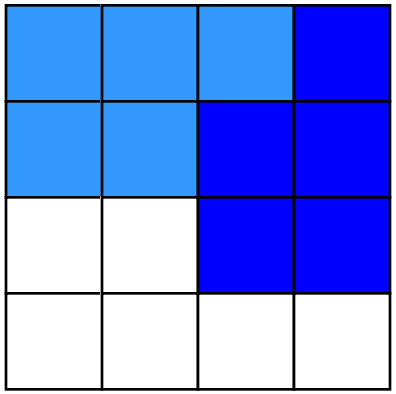


Two possible answers:

- *Premultiplied*
  the real color of the object is R/α, G/α, B/α
- or not
  the color of the object is R, G, B, and these values need to be multiplied by α for compositing

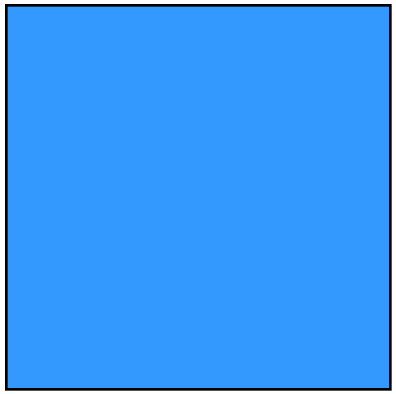
Pre-multiplied Alpha

- (R, G, B, α) means that the real object color is (R/α , G/α, B/α) and transparency is α.
- Motivated by supersampling for antialiasing in CG



$\{R_i, G_i, B_i, \alpha_i\}$

supersampled pixel



$1/n \quad R_i,$
$1/n \quad G_i,$
$1/n \quad B_i,$
$1/n \quad \alpha_i$

resampled (averaged value)

- If I combine multiple subpixels, the same operations apply to the four channels
  - In particular if I transform the image for scale/rotate

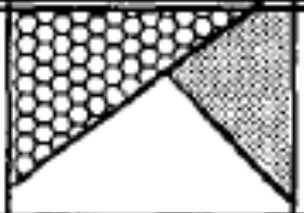
The Compositing Equation

Porter & Duff Siggraph 1984

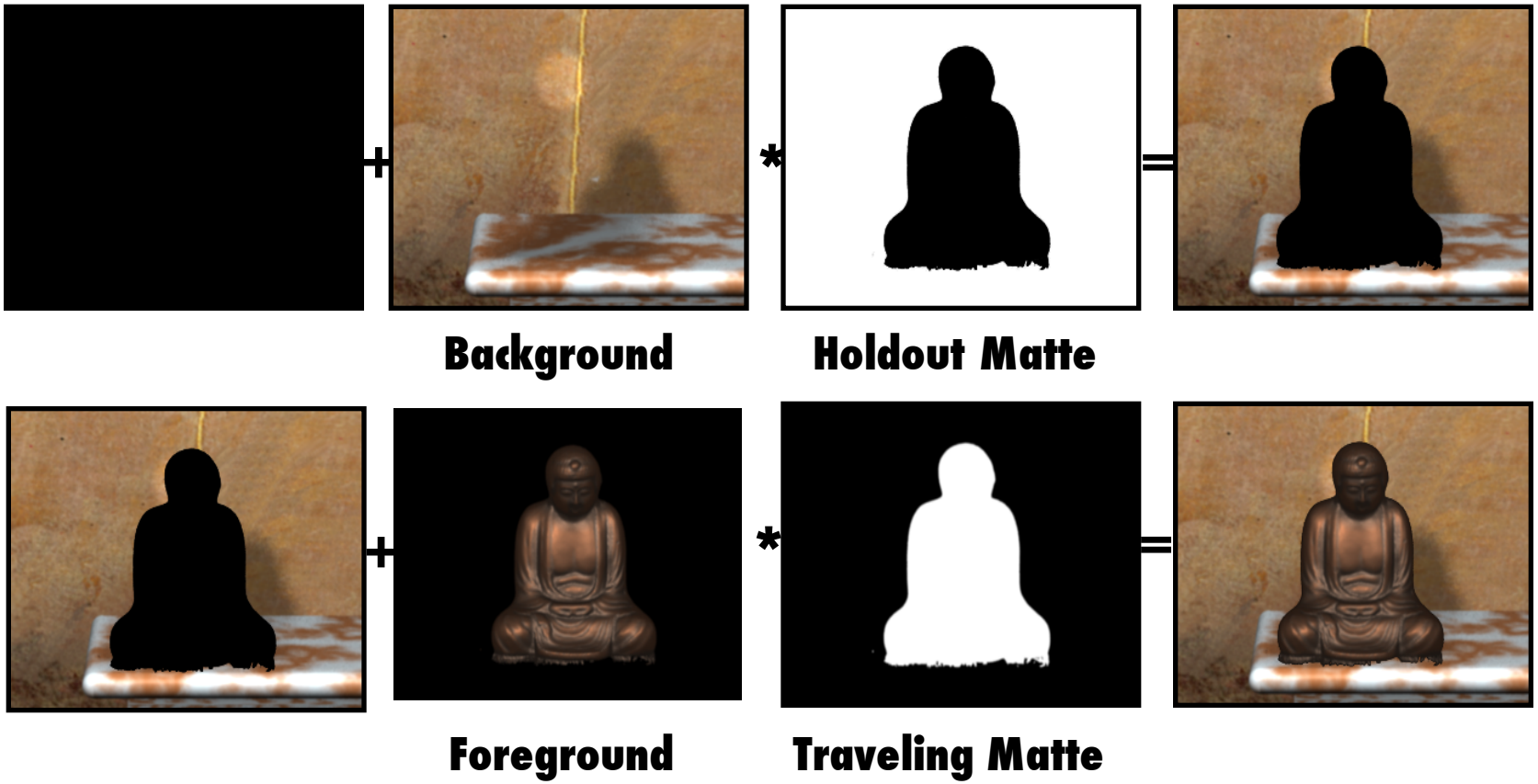
- Given Foreground F_A and Background F_B images (note that background has $\alpha_B=1$)
- For premultiplied alpha:

  Output = $F_A+(1-\alpha_A)\ F_B$
- For non-premultiplied:

  Output = $\alpha_A\ F_A+(1-\alpha_A)\ F_B$

operation	quadruple	diagram	F_A	F_B
A **over** *B*	(0,A,B,A)		1	$1-\alpha_A$

Composing Two Elements

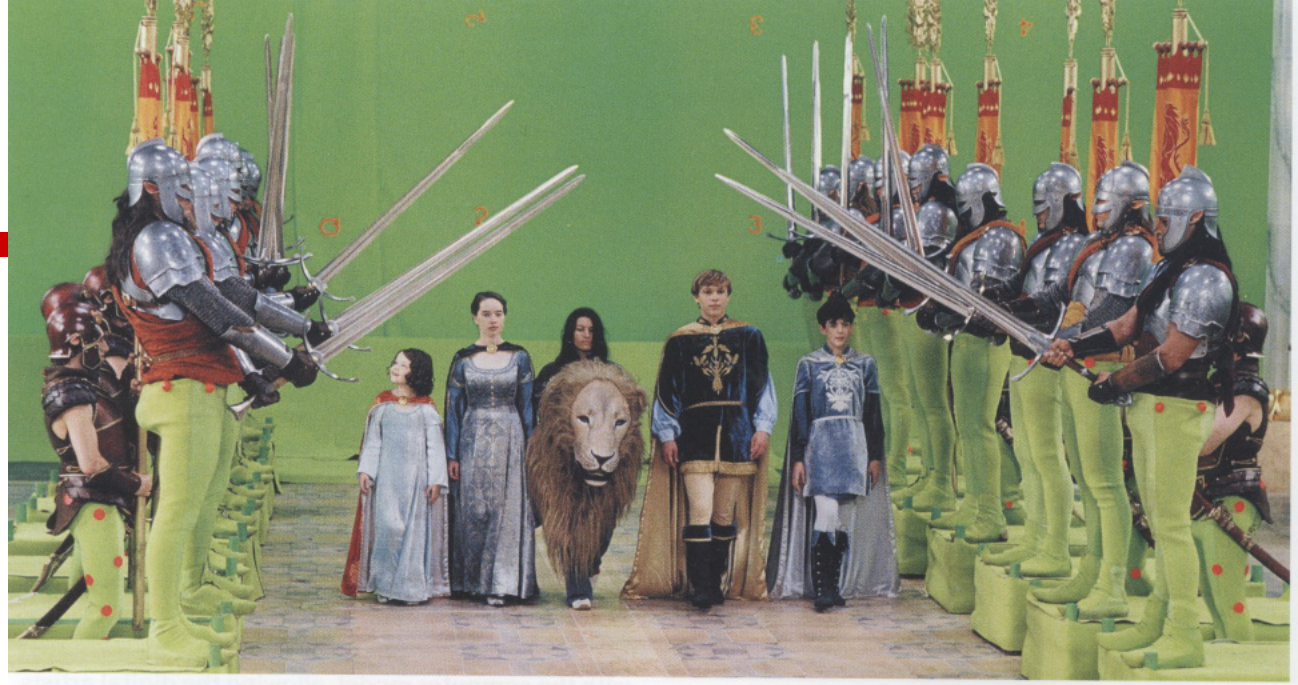


Slide from Pat Hanrahan

Limitations of Alpha

- Hard to represent stainglasses
  - It focuses on subpixel occlusion
- Does not model more complex optical effects
  - e.g. magnifying glass

Questions?

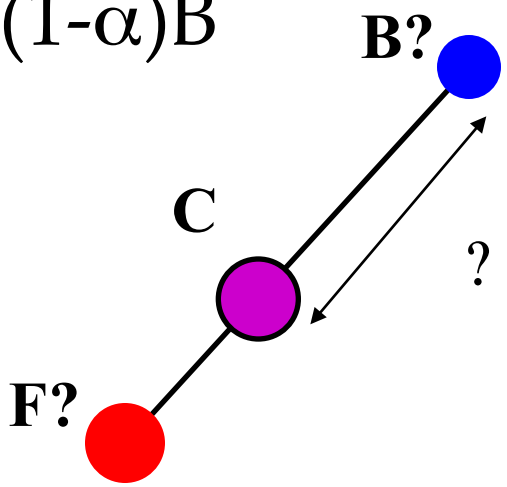


From Cinefex

Matting

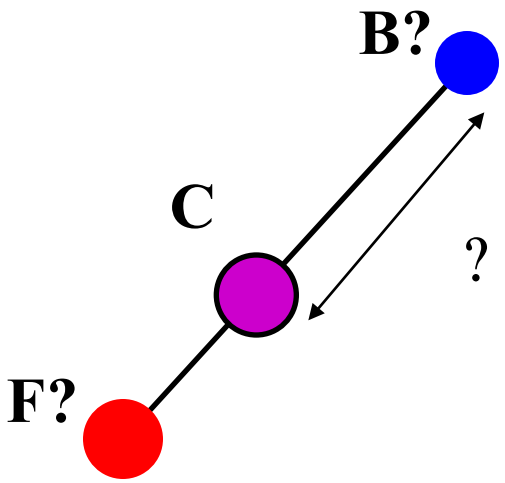
Matting problem

- Inverse problem:
  Assume an image is the *over* composite of a foreground and a background
- Given an image color C, find F, B and α so that
  $C=\alpha F+(1-\alpha)B$



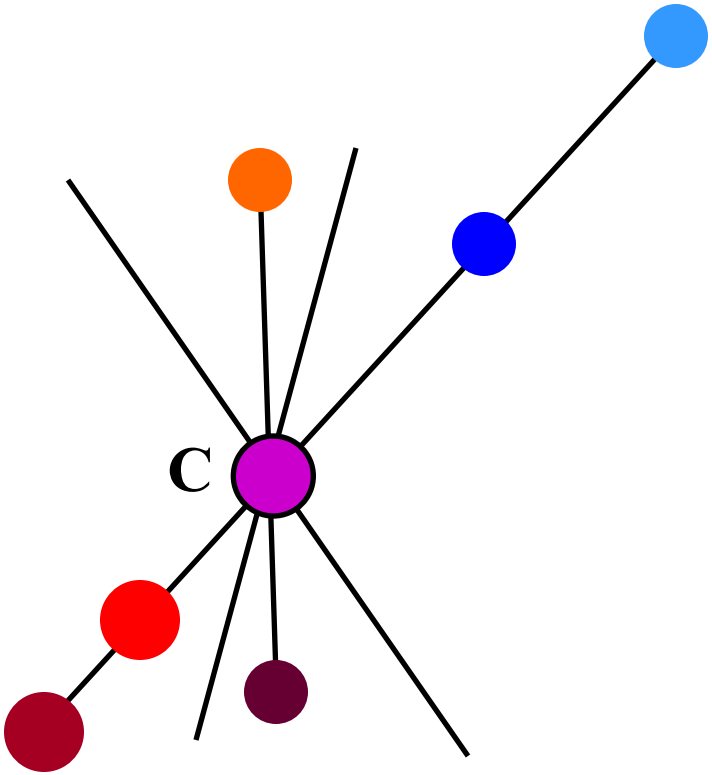
Matting Ambiguity

- $C=\alpha F+(1-\alpha)B$
- How many unknowns, how many equations?



Matting Ambiguity

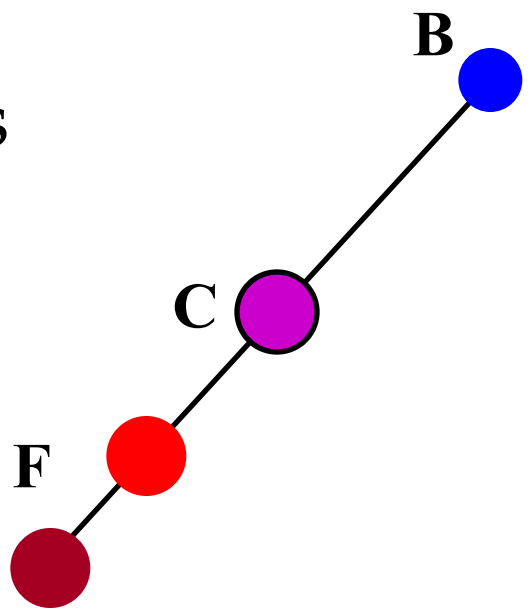
- $C=\alpha F+(1-\alpha)B$
- 7 unknowns: α and triplets for F and B
- 3 equations, one per color channel



Matting Ambiguity

- $C=\alpha F+(1-\alpha)B$
- 7 unknowns: α and triplets for F and B
- 3 equations, one per color channel
- With known background (e.g. blue/green screen):

  4 unknowns, 3 equations



Traditional Blue Screen Matting

Petro Vlahos
GORDON E. SAWYER AWARD
66TH ACADEMY AWARDS
1993

- Invented by Petro Vlahos (Technical Academy Award 1995)
- Recently formalized by Smith & Blinn
- Initially for film, then video, then digital
- Assume that the foreground has no blue
- Note that computation of α has to be analog, needs to be simple enough

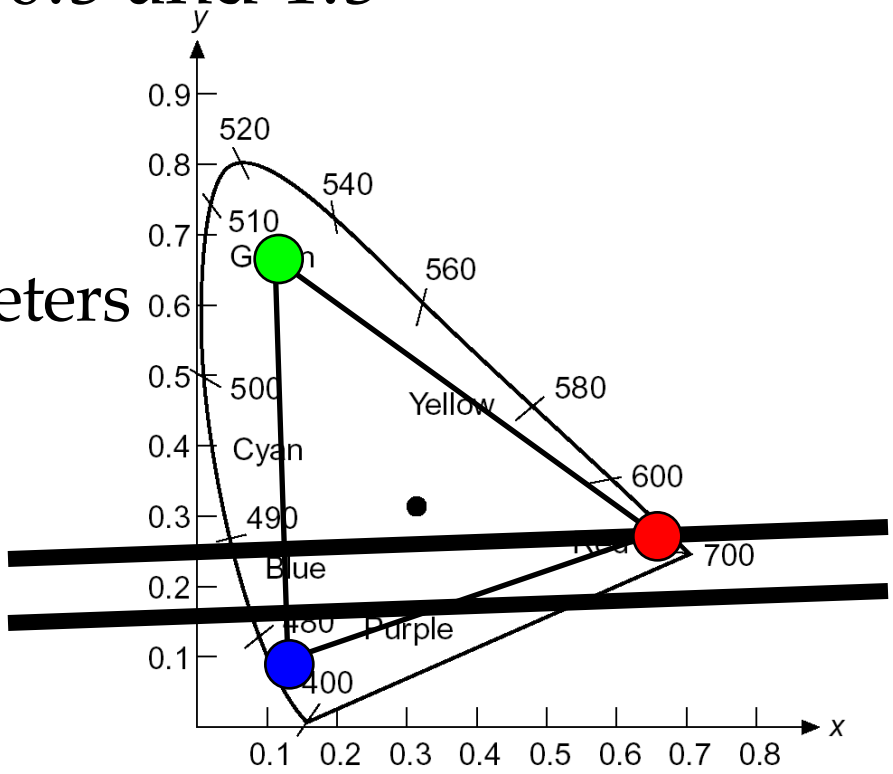
From Cinefex

Traditional Blue Screen Matting

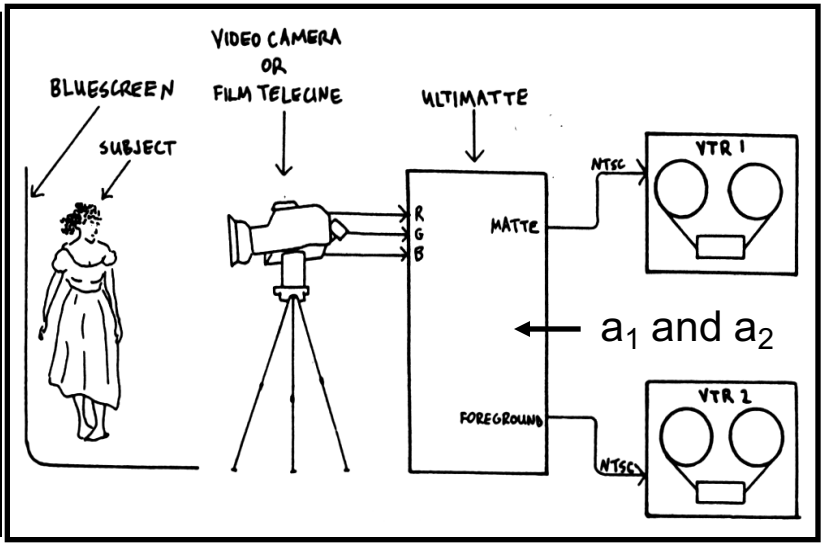
- Assume that blue *b* and green *g* channels of the foreground respect $b < a_2 g$ for a_2 typically between 0.5 and 1.5
- $\alpha = 1 - a_1(b - a_2 g)$
  - clamped to 0 and 1
  - a_1 and a_2 are user parameters
  - Note that $\alpha = 1$ where assumption holds

$b - a_2 g = 0$

$b - a_2 g = 1/a_1$



The Ultimatte



Blue/Green Screen Matting Issues

- Color limitation
  - Annoying for blue-eyed people

  ➔ adapt screen color (in particular green)
- Blue/Green spilling
  - The background illuminates the foreground, blue/green at silhouettes
  - Modify blue/green channel, e.g. set to min (b, a_2g)
- Shadows
  - How to extract shadows cast on background

Blue/Green Screen Matting Issues

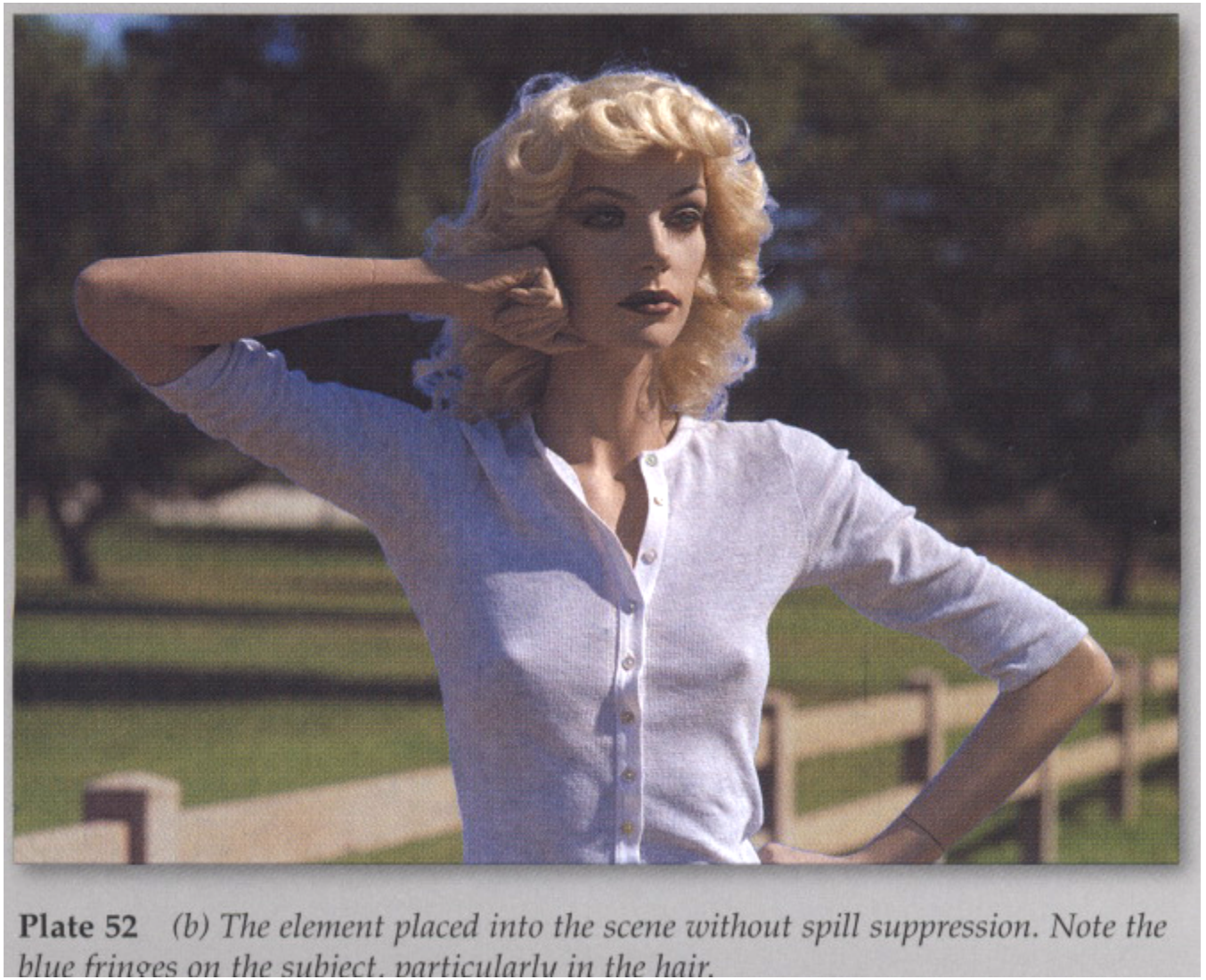


Plate 52 *(b) The element placed into the scene without spill suppression. Note the blue fringes on the subject, particularly in the hair.*

From the Art & Science of Digital Compositing

- http://www.digitalgreenscreen.com/figure3.html

Figure 3. *Firefox Blue Spill Matte Series 1, original shot. Note blue reflected on wing surfaces from bluescreen -- undesirable but unavoidable on such surfaces.*

Extension: Chroma Key

- Blue/Green screen matting exploits color channels
- Chroma key can use an arbitrary background color
- See e.g.
  - http://www.cs.utah.edu/~michael/chroma/
  - Keith Jack, "Video Demystified", Independent Pub Group (Computer), 1996

Extension: Natural Matting

[Ruzon & Tomasi 2000, Chuang et al. 2001]

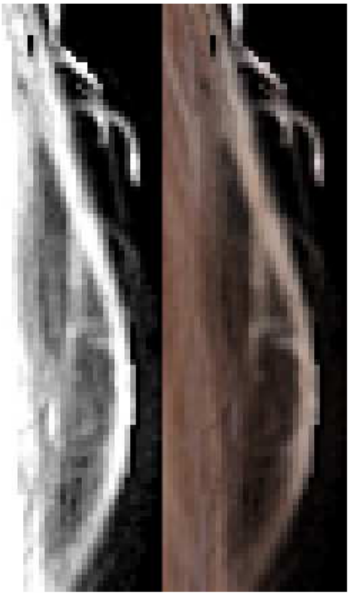
- Given an input image with arbitrary background
- The user specifies a coarse *Trimap* (known Foreground, known background and unknown region)
- Goal: Estimate F, B, alpha in the unknown region
  - We don't care about B, but it's a byproduct/unknown

images from Chuang et al

Bayesian approach

Alpha Matte

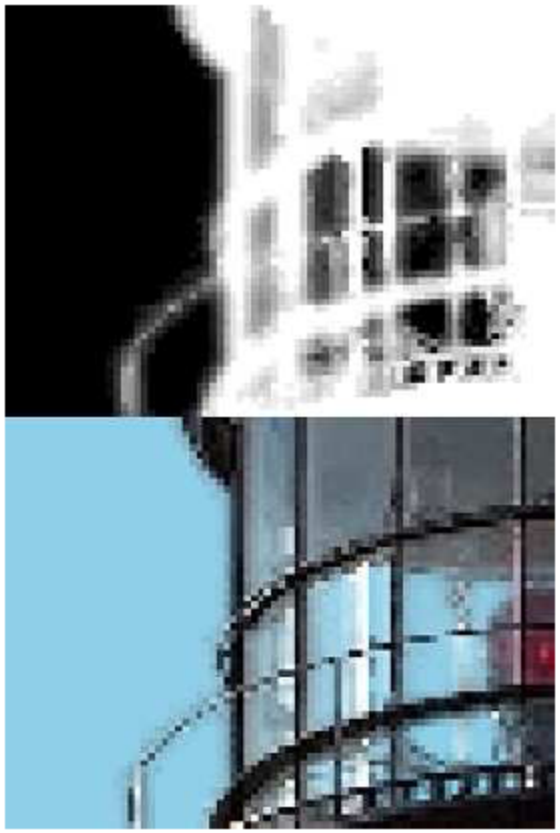
Composite



Inset

Alpha Matte

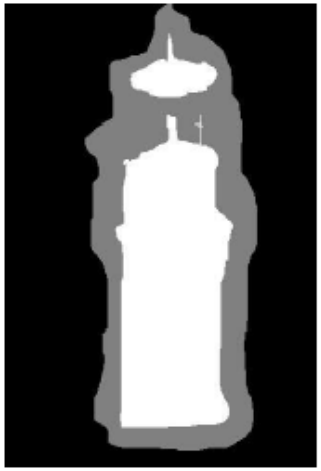
Composite



Inset



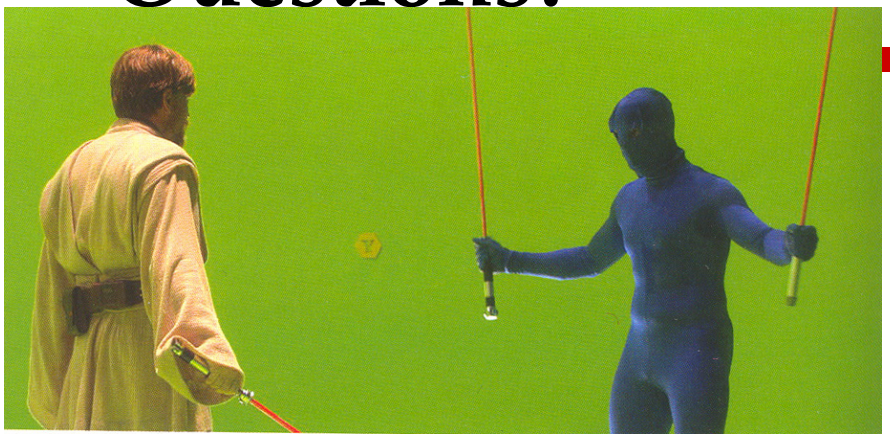


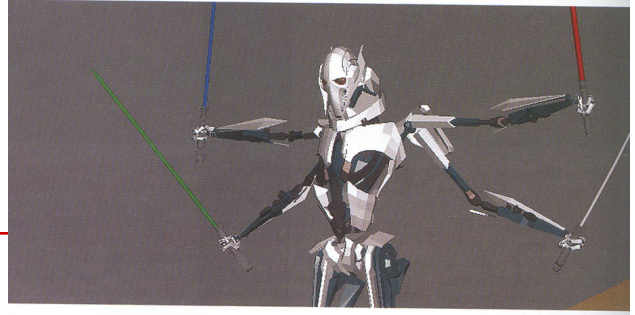


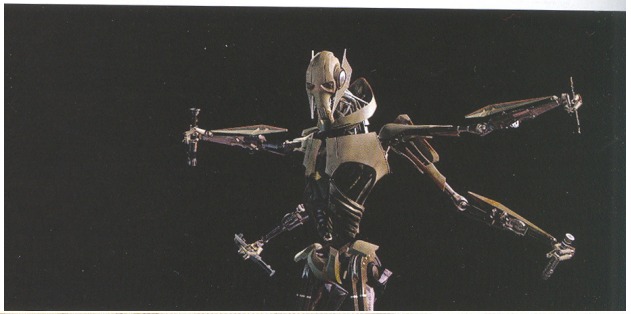


From Chuang et al 2001

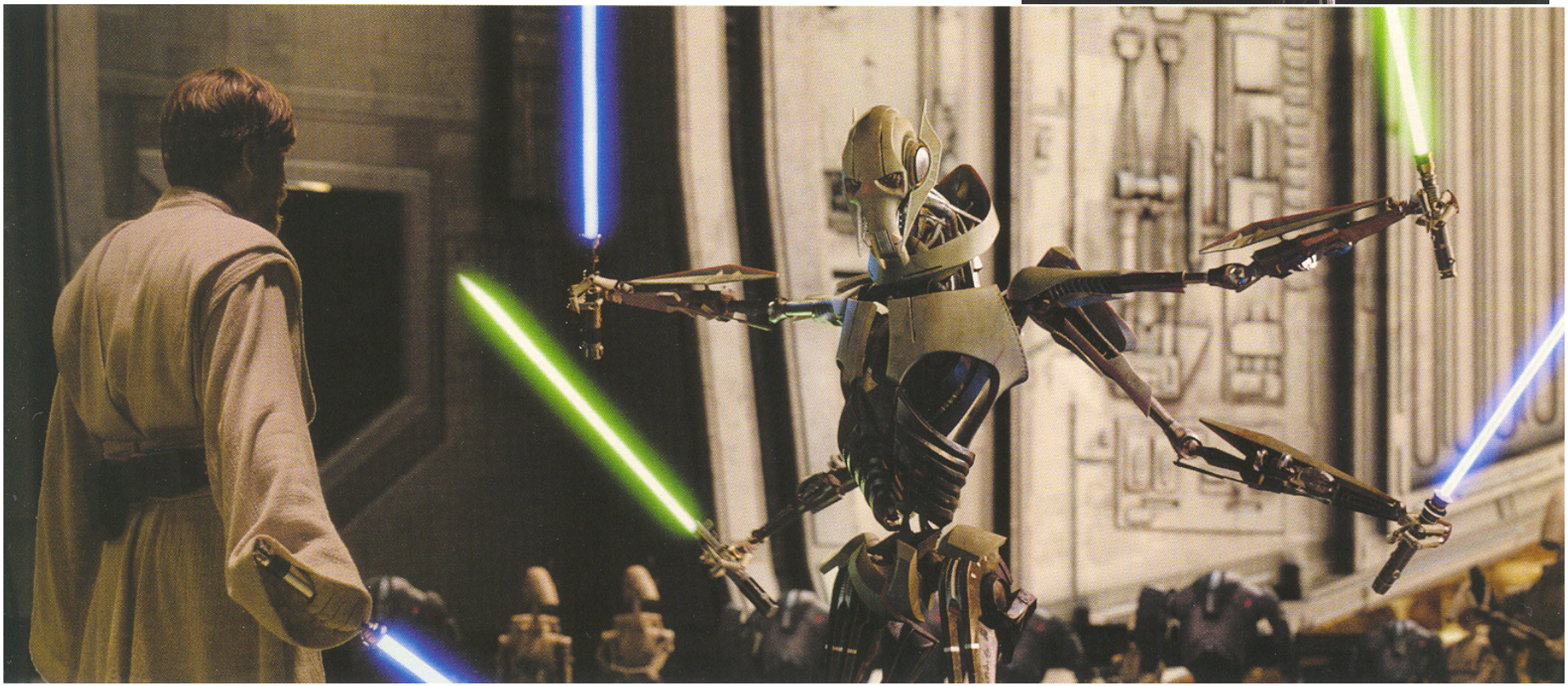
Questions?





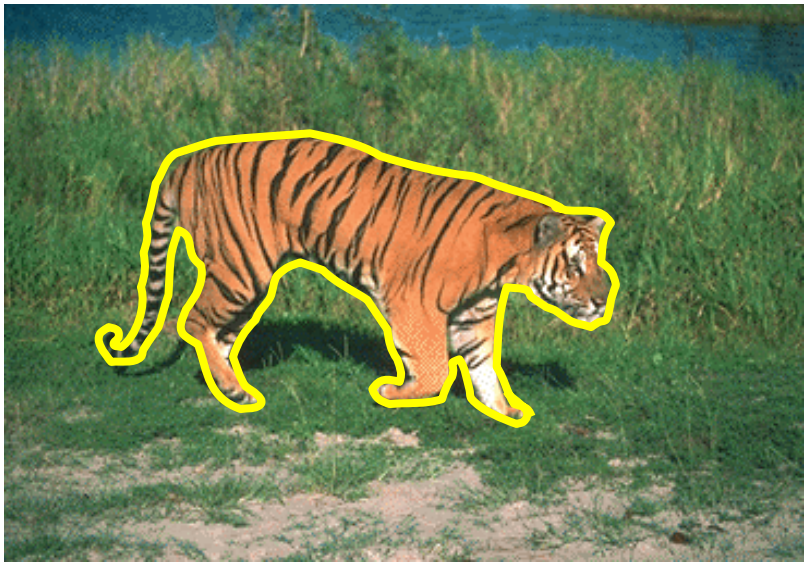


From Cinefex



Intelligent Scissors

Extracting Objects



- How could this be done?
  - hard to do manually
  - hard to do automatically ("image segmentation")
  - easy to do *semi-automatically*

Intelligent Scissors (demo)

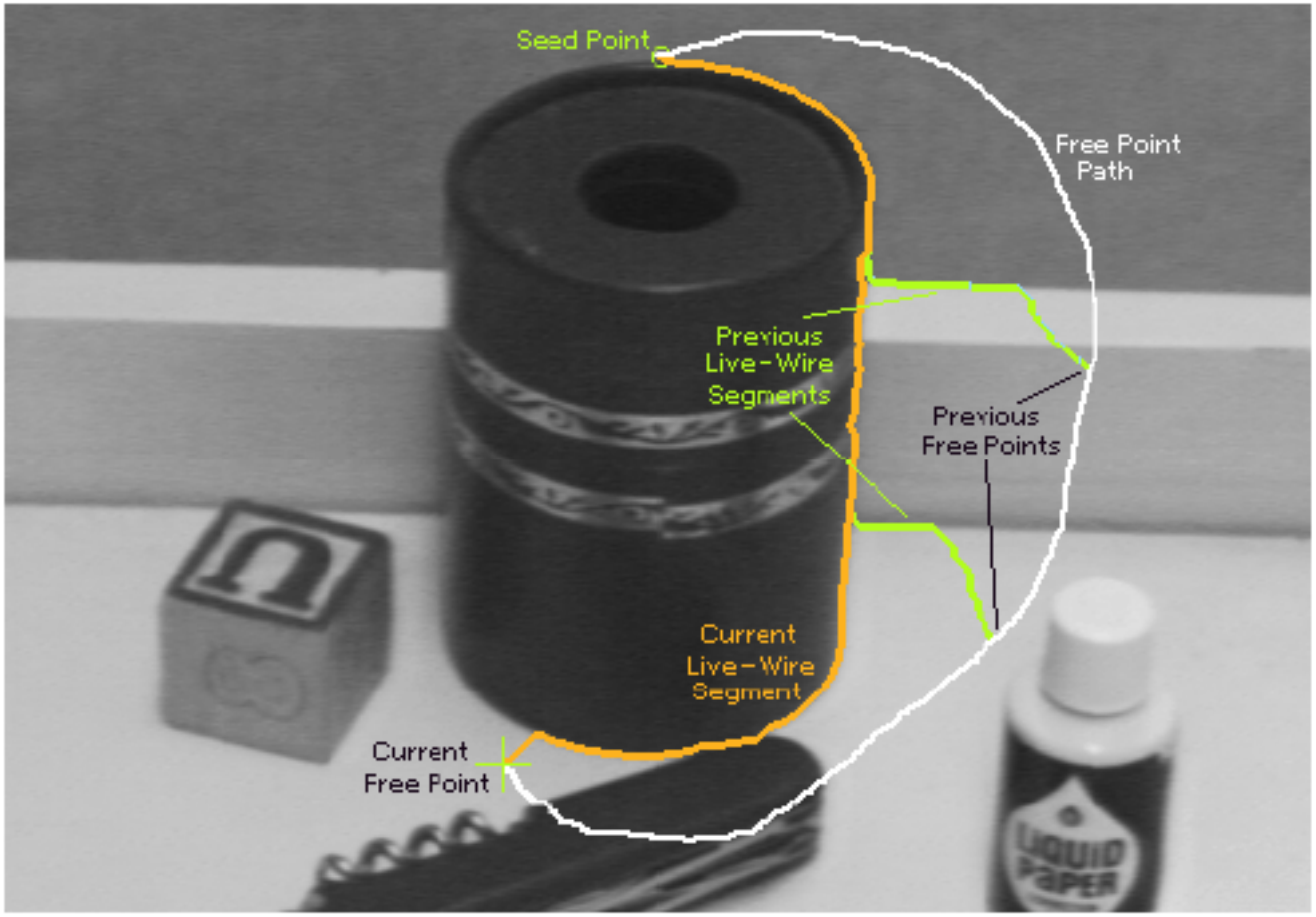


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

- Approach answers a basic question
  - Q: how to find a path from seed to mouse that follows object boundary as closely as possible?
  - A: define a path that stays as close as possible to edges

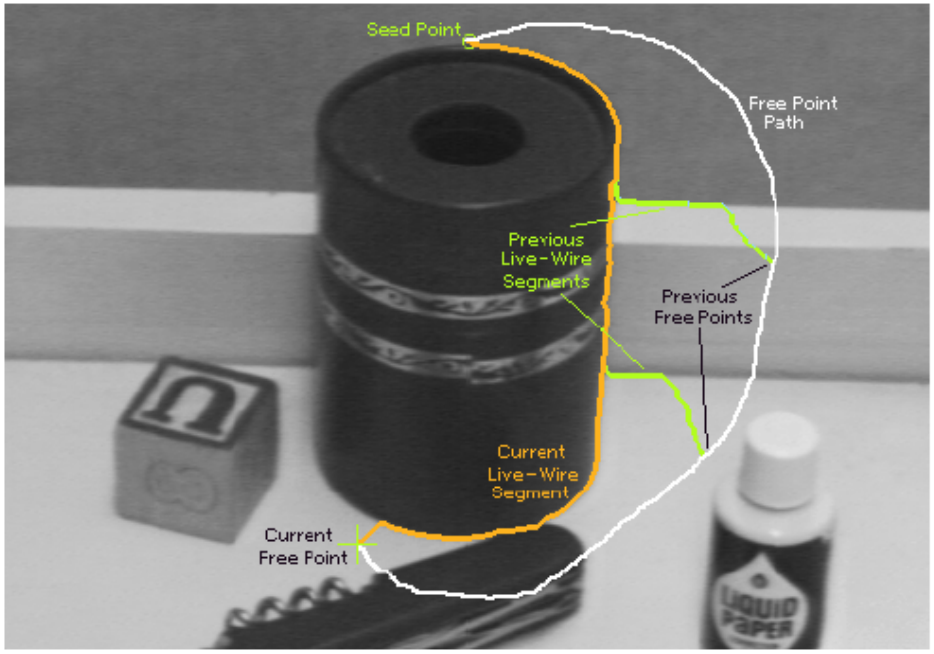
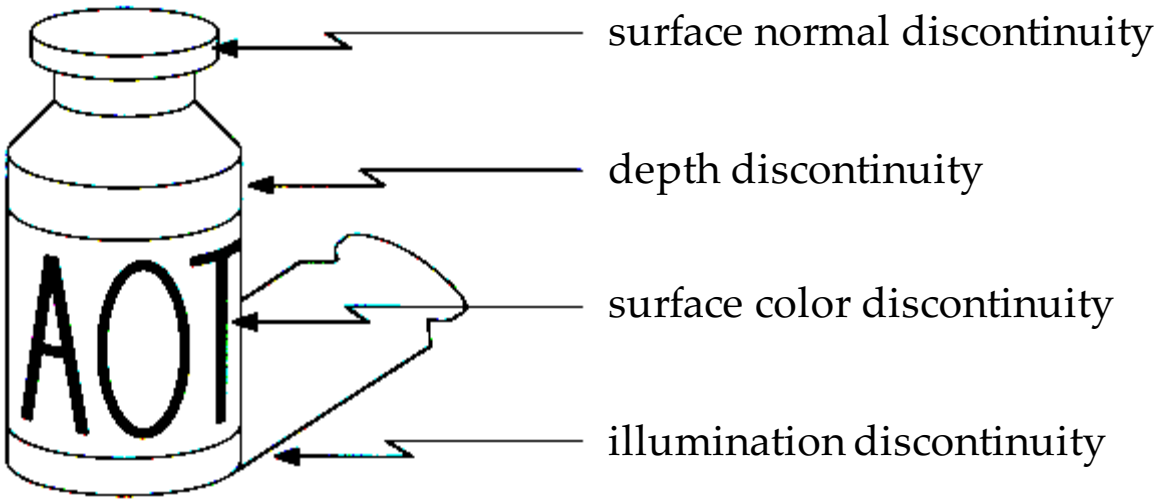


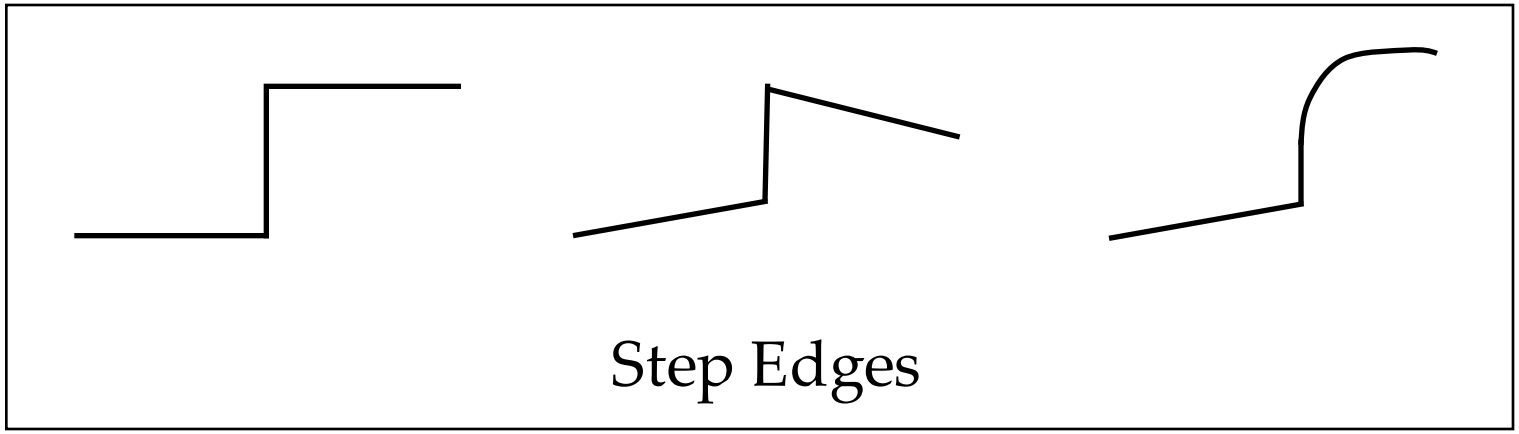
Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

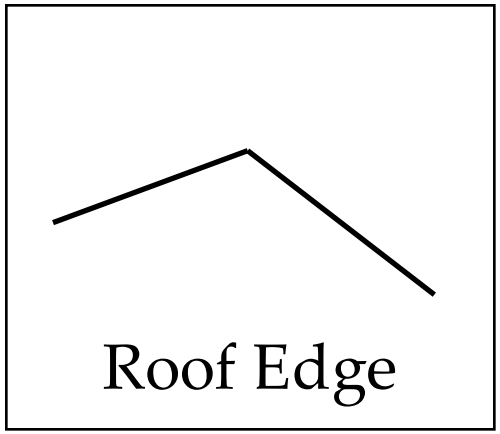
Origin of Edges

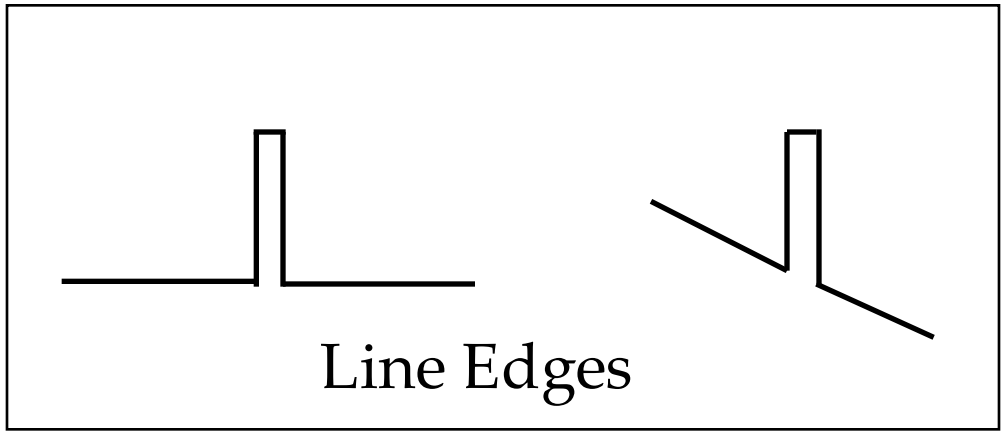


- Edges are caused by a variety of factors

Edge Types

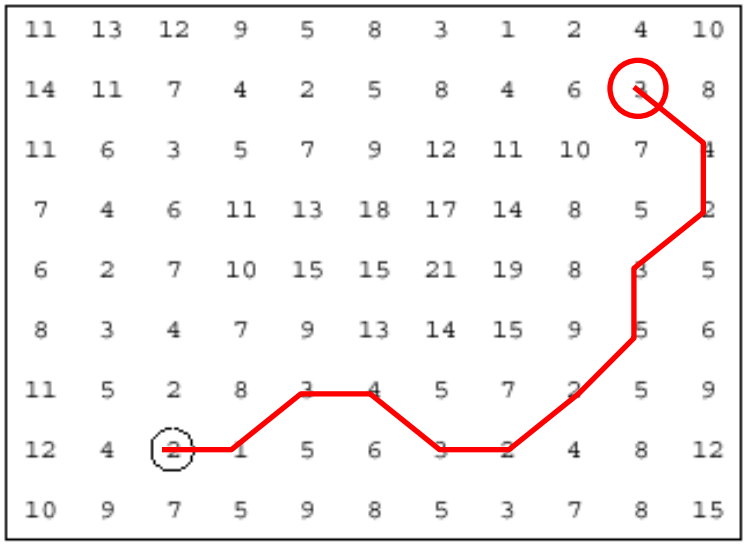






Intelligent Scissors

- Basic Idea
  - Define edge score for each pixel
    - edge pixels have low cost
  - Find lowest cost path from seed to mouse (or the other point you clicked)



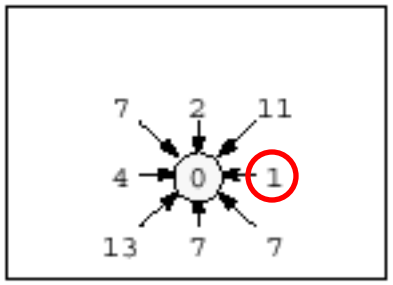
Questions

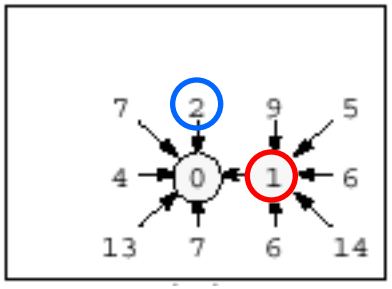
- How to define costs?
- How to find the path?

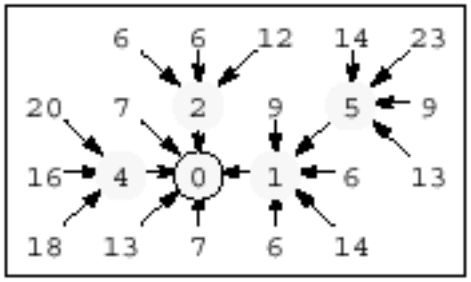
Path Search (basic idea)

- Graph Search Algorithm
  - Computes minimum cost path from seed to *all other pixels*

11	13	12	9	5	8	3	1	2	4	10
14	11	7	4	2	5	8	4	6	3	8
11	6	3	5	7	9	12	11	10	7	4
7	4	6	11	13	18	17	14	8	5	2
6	2	7	10	15	15	21	19	8	3	5
8	3	4	7	9	13	14	15	9	5	6
11	5	2	8	3	4	5	7	2	5	9
12	4	2	1	5	6	3	2	4	8	12
10	9	7	5	9	8	5	3	7	8	15

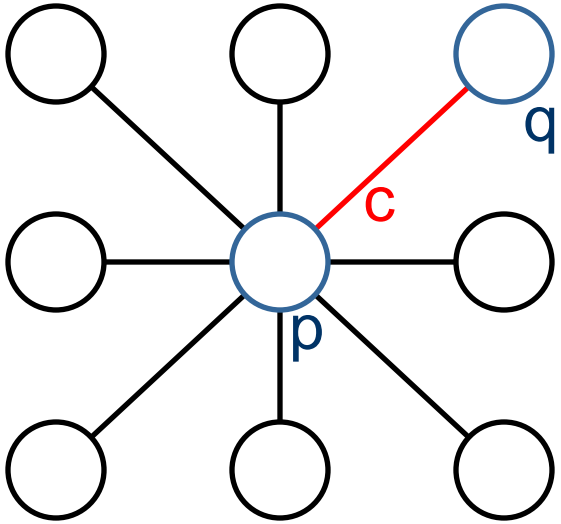






Better: Make Each *Edge* have a cost

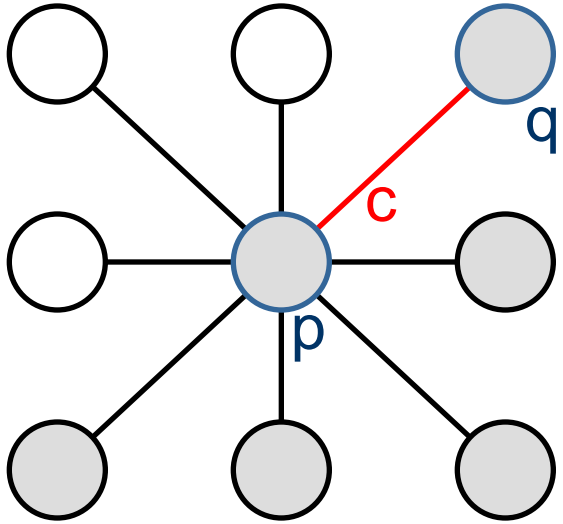
- Treat the image as a graph



- Graph
  - node for every pixel **p**
  - link between every adjacent pair of pixels, **p**,**q**
  - cost **c** for each link
- Note: each *link* has a cost
  - this is a little different than the figure before where each pixel had a cost

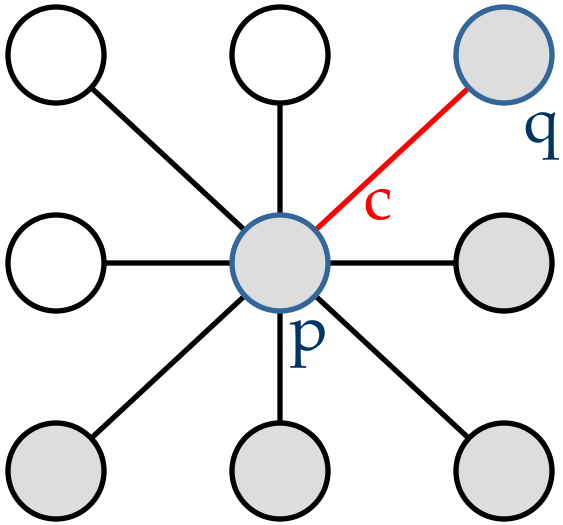
Defining the Costs

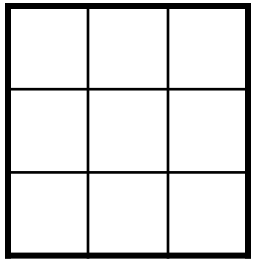
- Treat the image as a graph



- Want to hug image edges: how to define cost of a link?
  - the link should follow the intensity edge
    - want intensity to change rapidly to the link
  - c: difference of intensity perpendicular to link

Defining the Costs



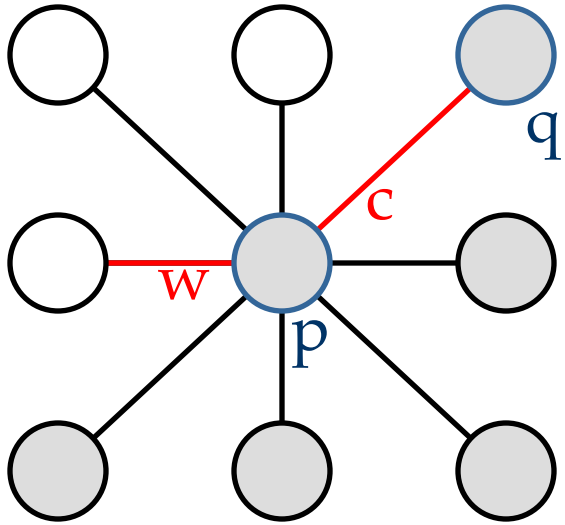


- c can be computed using a cross-correlation filter
  - assume it is centered at p

Defining the Costs

$\frac{1}{4}$

1	1	
-1	-1	



$\frac{1}{2}$

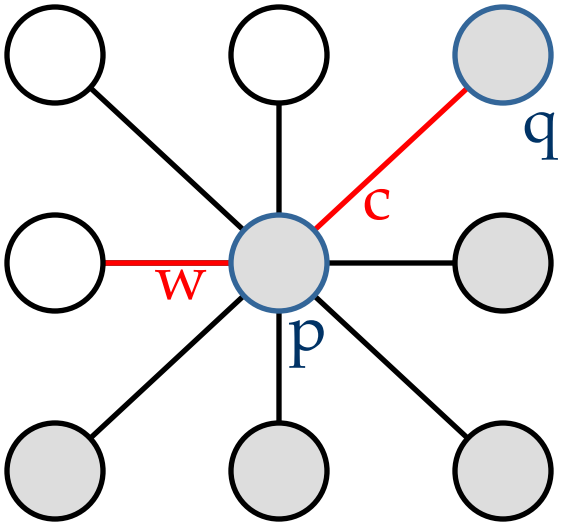
	1	
		-1

- c can be computed using a cross-correlation filter
  - assume it is centered at p

Defining the Costs

$\frac{1}{4}$

1	1	
-1	-1	



$\frac{1}{2}$

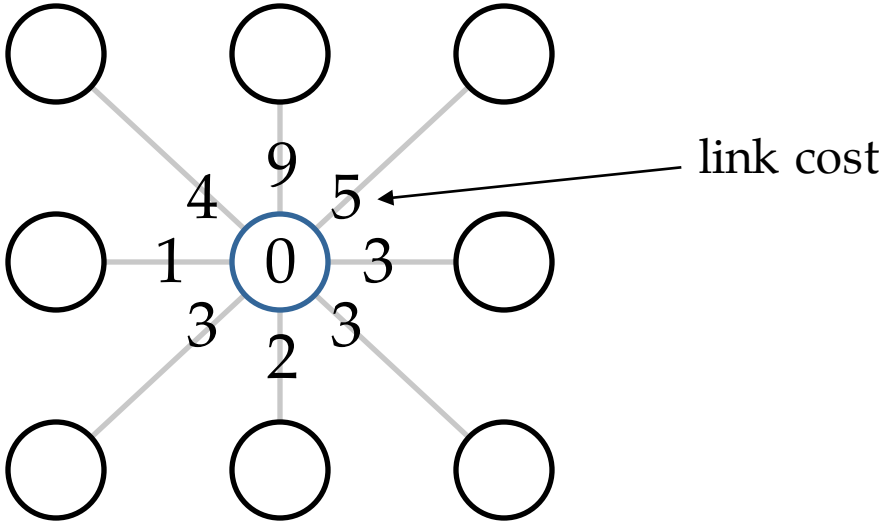
	1	
		-1

- c can be computed using a cross-correlation filter
  - assume it is centered at p
- Also typically scale c by its length
  - set c = (max- | filter response |)×length
    - where max = maximum | filter response | over all pixels in the image

Finding Cut

- How do we find the path?

Dijkstra's Shortest Path Algorithm

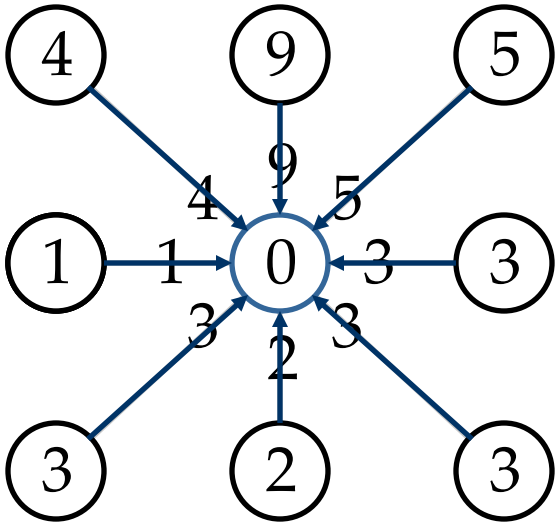


- Algorithm
  1. Initialize all node costs to inf, set p = seed point, and set cost(p) = 0
  2. expand p as follows:

     for each of p's neighbors q that are not expanded

     - set cost(q) = min(cost(p) + c_{pq}, cost(q))

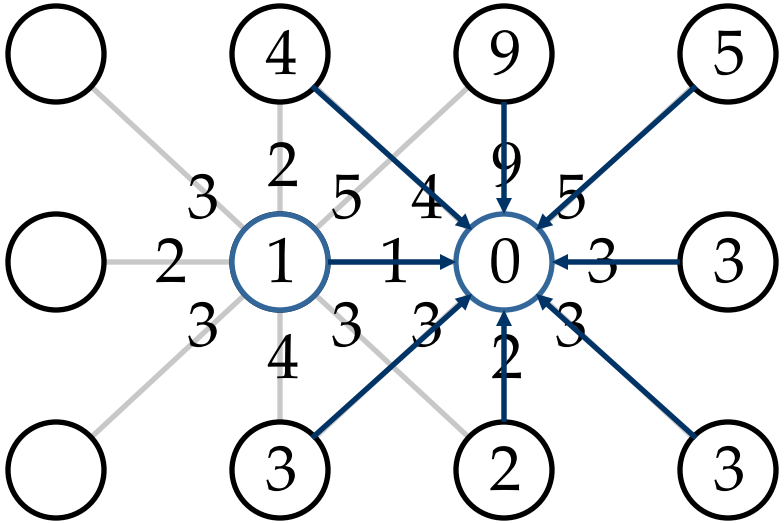
Dijkstra's Shortest Path Algorithm



- Algorithm
  1. init node costs to inf, set p = seed point, cost(p) = 0
  2. expand p as follows:

     for each of p's neighbors q that are not expanded
     - set cost(q) = min(cost(p) + c_{pq}, cost(q))
       - if q's cost changed, make q point back to p
     - put q on the ACTIVE list (if not already there)

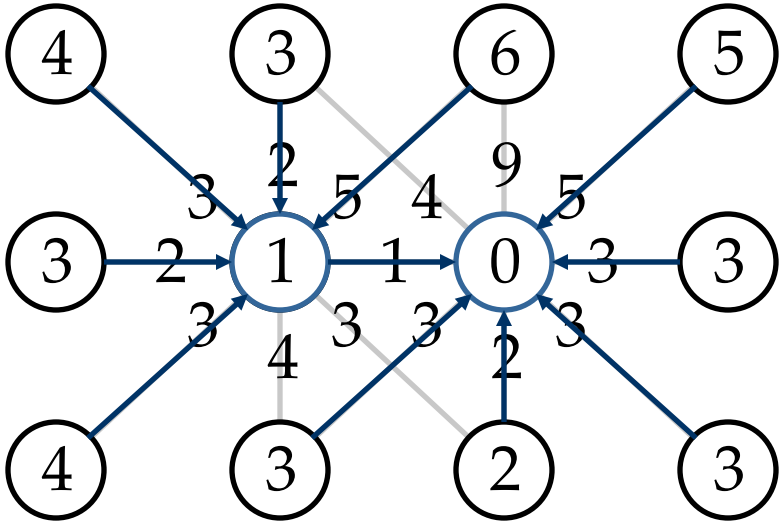
Dijkstra's Shortest Path Algorithm



- Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:
   for each of p's neighbors q that are not expanded
   - set cost(q) = min(cost(p) + c_{pq}, cost(q))
     - if q's cost changed, make q point back to p
   - put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r (put r on EXPAND list)

Dijkstra's Shortest Path Algorithm



- Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:
   for each of p's neighbors q that are not expanded
   - set cost(q) = min(cost(p) + c_{pq}, cost(q))
     - if q's cost changed, make q point back to p
   - put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r (put r on EXPAND list)

Algorithm

Begin

initialize the priority queue *pq* to be empty;

initialize each node to the INITIAL state;

set the total cost of *seed* to be zero;

insert *seed* into *pq*;

extract the node *q* with the minimum total cost in *pq*;

% Now find the shortest path.

while *q* is **not** (0,0) & q is not goal

% If *q* is (0,0), it means the queue is empty.

mark *q* as EXPANDED;

for each neighbor node *r* of *q*

if *r* has **not** been EXPANDED

mark *r* as ACTIVE;

insert *r* in *pq* with the sum of the total cost of q and link cost from *q* to *r* as its total cost;

if inserting r changed it

make an entry for *r* in the Pointers array indicating that currently the best way to reach *r* is from *q*.

extract the node *q* with the minimum total cost in *pq*;

End

Video Example

- https://www.youtube.com/watch?v=_lHSawdgXpI

Dijkstra's Shortest Path Algorithm

- Properties
  - It computes the minimum cost path from the seed to every node in the graph. This set of minimum paths is represented as a *tree*
  - Running time, with N pixels:
    - $O(N^2)$ time if you use an active list
    - O(N log N) if you use an active priority queue (heap)
  - What happens when the user specifies a new seed?

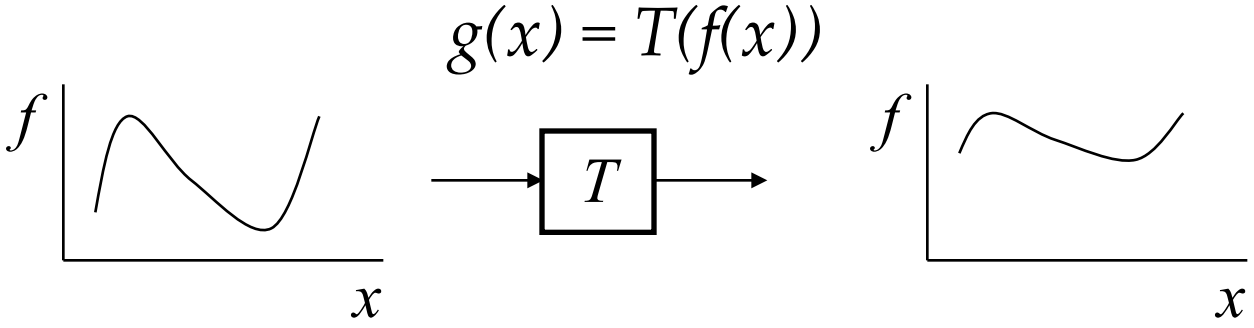
Compositing

Results

Image Warping

Image Warping

- image filtering: change ***range*** of image



- image warping: change ***domain*** of image

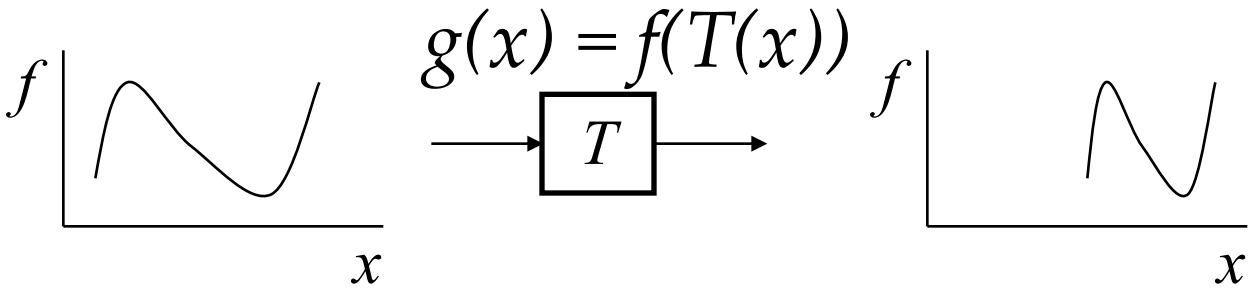
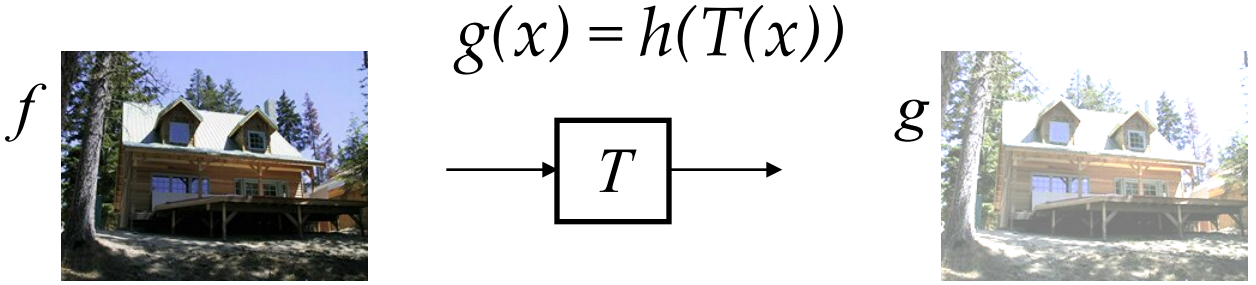
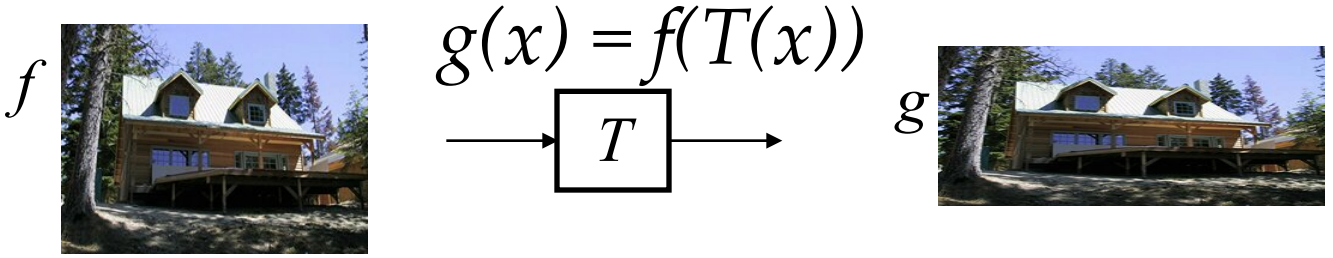


Image Warping

- image filtering: change ***range*** of image



- image warping: change ***domain*** of image



Parametric (Global) Warping

- Examples of parametric warps:

translation

rotation

aspect

affine

perspective

cylindrical

Parametric (Global) Warping

T

p = (x,y)

p' = (x',y')

- Transformation T is a coordinate-changing machine:

$$p' = T(p)$$

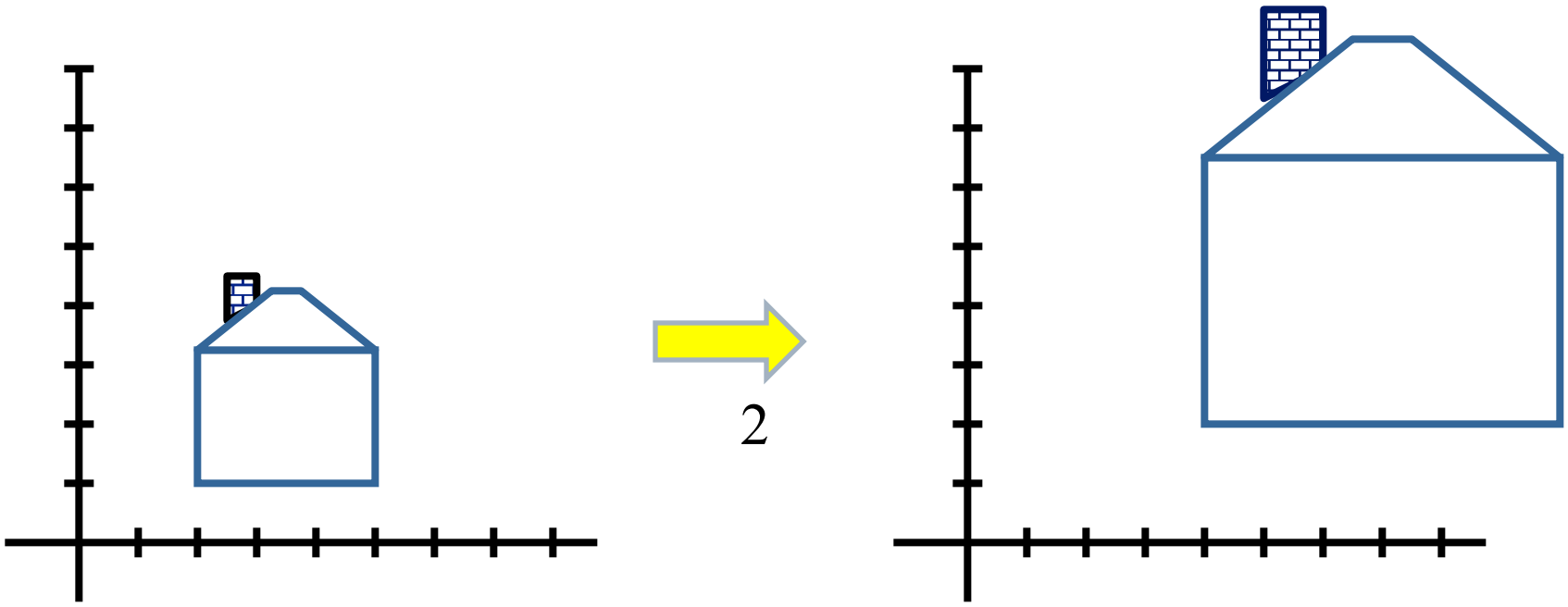
- What does it mean that T is global and parametric?
  - Is the same for any point p
  - can be described by just a few numbers (parameters)
- Let's represent T as a matrix:

$$p' = \mathbf{M}p$$

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

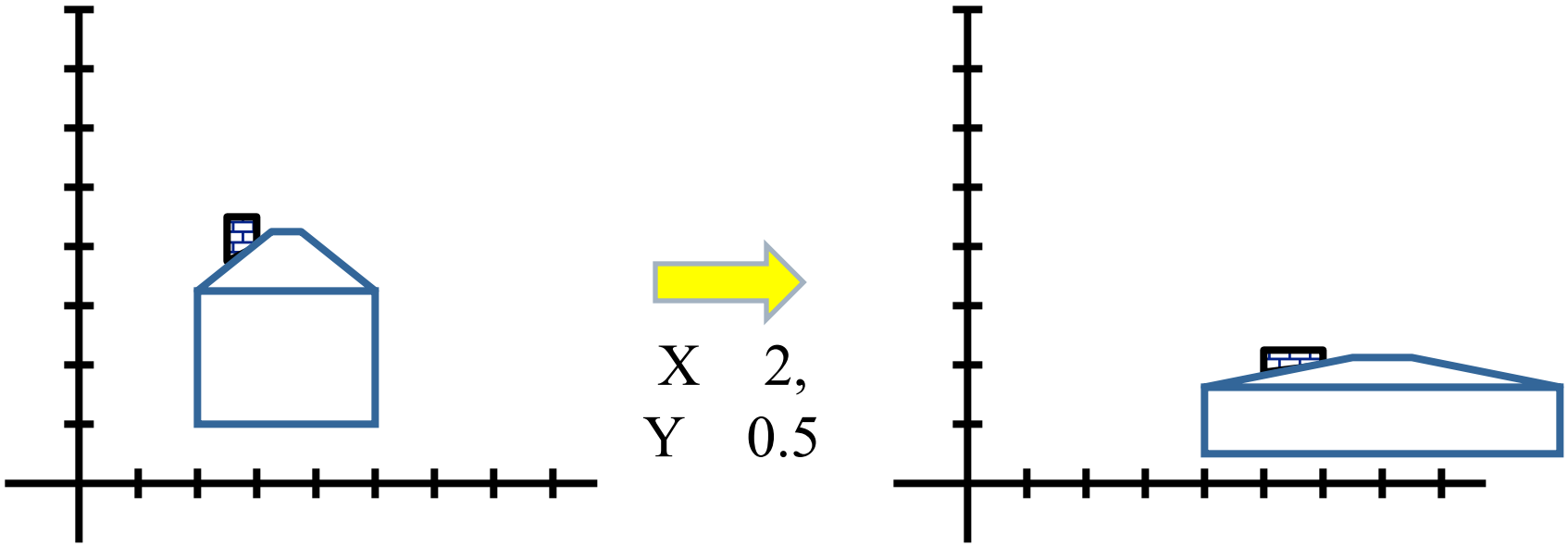
Scaling

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



Scaling

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



Scaling

- Scaling operation:

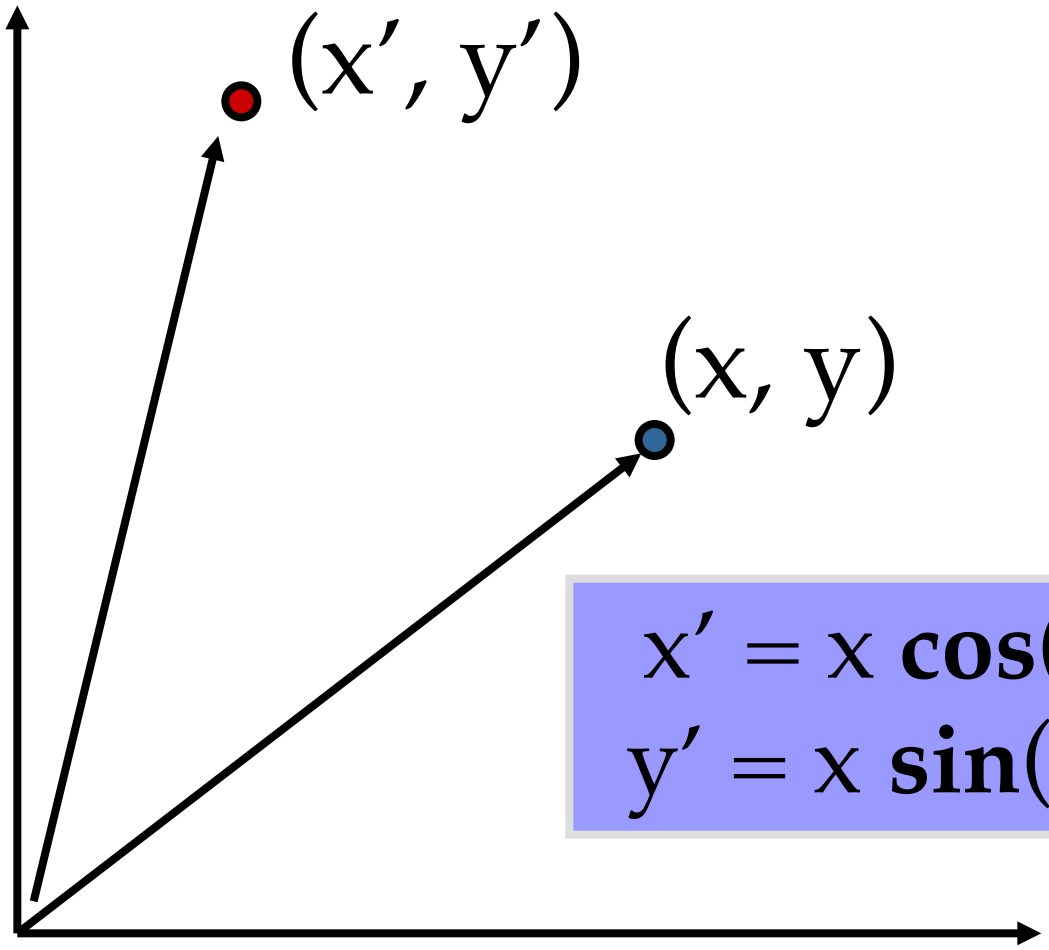
$$x' = ax$$
$$y' = by$$

- Or, in matrix form:

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

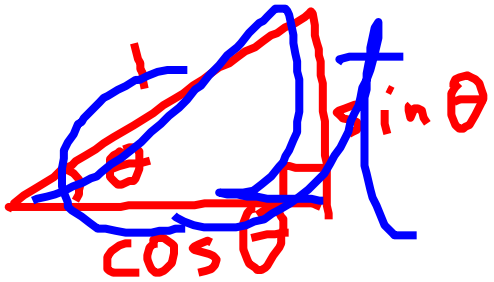
What's inverse of S?

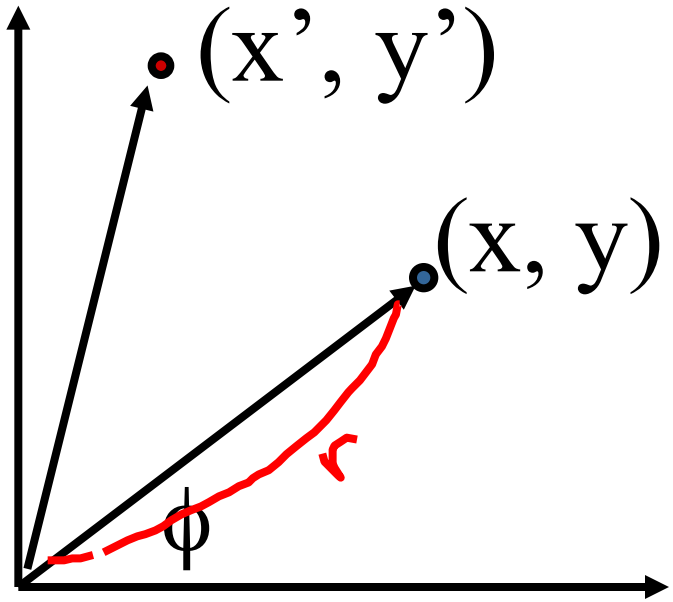
2-D Rotation



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

2-D Rotation





$x = r \cos(\phi)$

$y = r \sin(\phi)$

$x' = r \cos(\phi + \ )$

$y' = r \sin(\phi + \ )$

Trig Identity…

$x' = r \cos(\phi) \cos(\ ) - r \sin(\phi) \sin(\ )$

$y' = r \sin(\phi) \sin(\ ) + r \cos(\phi) \cos(\ )$

Substitute…

$x' = x \ \mathbf{cos}(\ ) - y \ \mathbf{sin}(\ )$

$y' = x \ \mathbf{sin}(\ ) + y \ \mathbf{cos}(\ )$

2-D Rotation

$R^{-1}(\theta) = \frac{1}{\cos^2\theta + \sin^2\theta}\begin{bmatrix}\cos\theta & \sin\theta \\ -\sin\theta & \cos\theta\end{bmatrix}$

- This is easy to capture in matrix form:

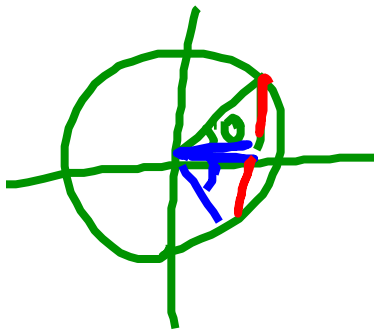
$$\begin{bmatrix}x' \\ y'\end{bmatrix} = \underbrace{\begin{bmatrix}\cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta)\end{bmatrix}}_{\mathbf{R}}\begin{bmatrix}x \\ y\end{bmatrix}$$

$= \begin{bmatrix}\cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta)\end{bmatrix} = R(-\theta)$

$\cos(-\theta) = \cos\theta$

$\sin(-\theta) = -\sin\theta$

- Even though sin(θ) and cos(θ) are nonlinear functions of 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 so $\mathbf{R}^{-1} = \mathbf{R}^{T}$

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)?

$$x' = s_x x$$
$$y' = s_y y$$

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 \\ 0 & \boldsymbol{s}_y \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

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?

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

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

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)?

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

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

2x2 Matrices

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

2D Translation?

$$x' = x + t_x$$
$$y' = y + t_y$$

NO!

Only linear 2D transformations
can be represented with a 2x2 matrix

All 2D Linear Transformations

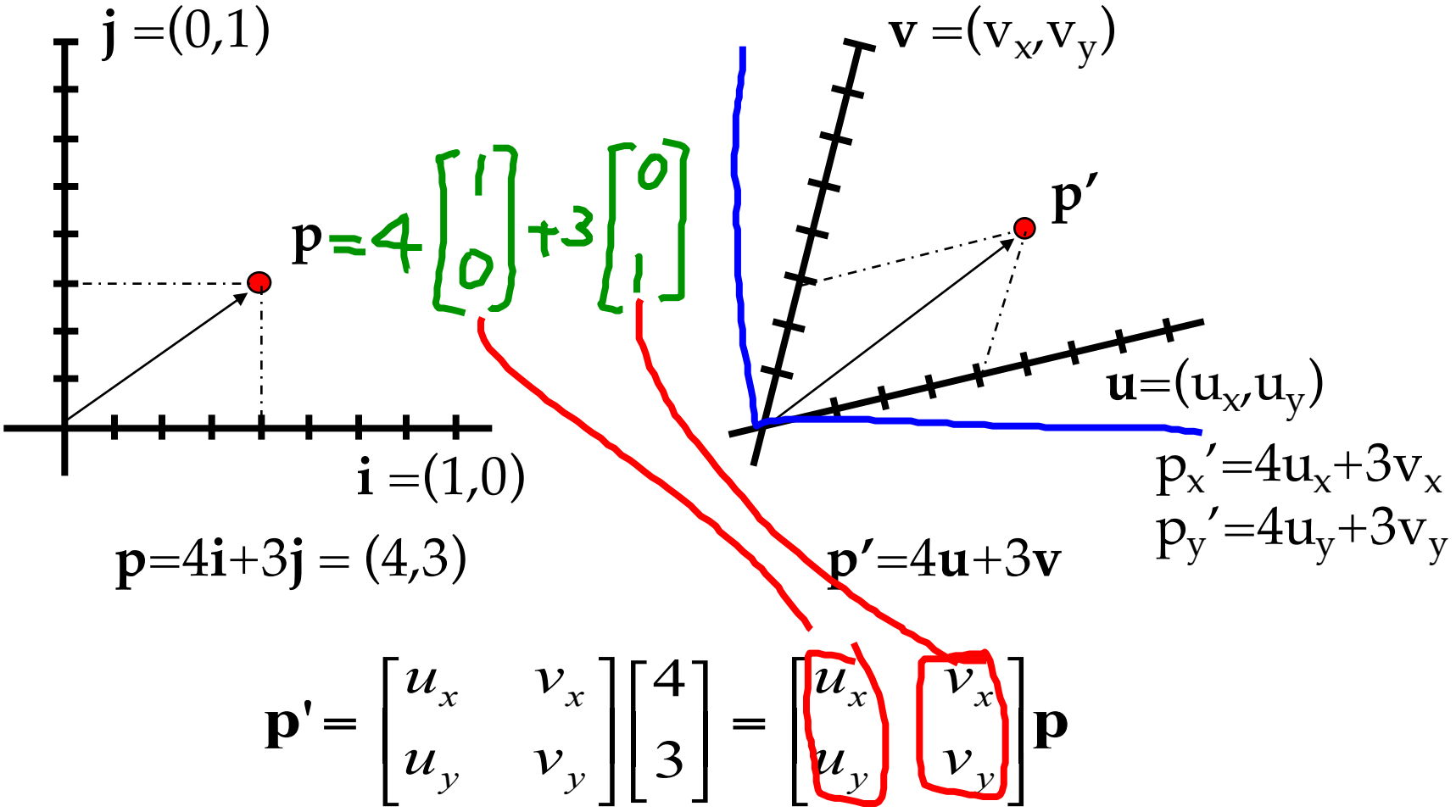
- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

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

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Linear Transformations as Change of Basis



$$\mathbf{p}' = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \mathbf{p}$$

- Any linear transformation is a basis!!!

Homogeneous Coordinates

- **Q: How can we represent translation as a 3x3 matrix?**

$$x' = x + t_x$$

$$y' = y + t_y$$

Homogeneous Coordinates

- ***Homogeneous coordinates***
  - represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{\text{homogeneous coords}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous Coordinates

- **Q: How can we represent translation as a 3x3 matrix?**

$$x' = x + t_x$$

$$y' = y + t_y$$

- A: Using the rightmost column:

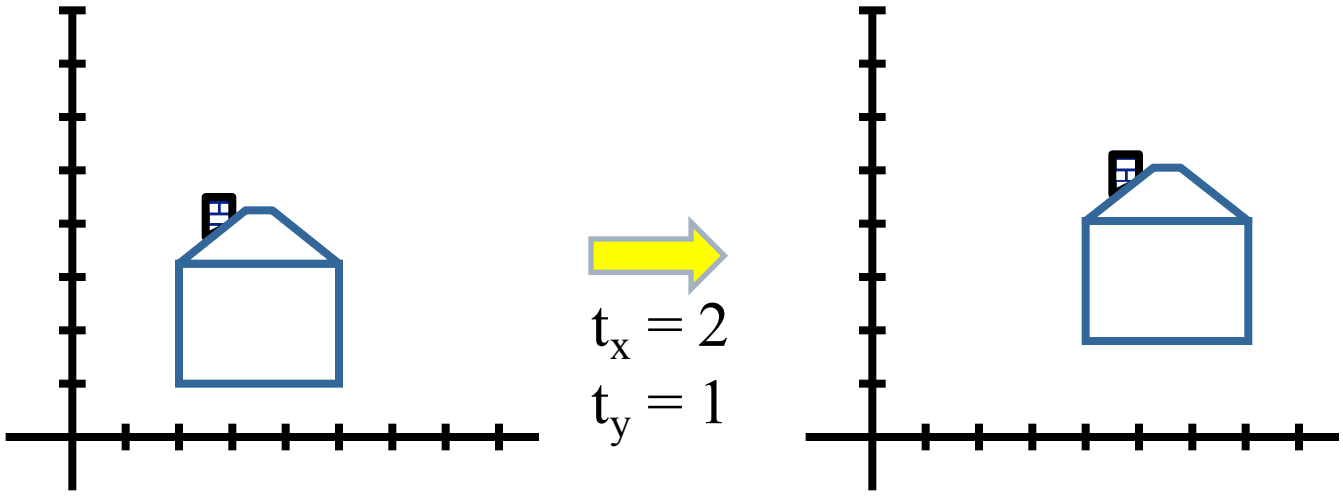
$$\mathbf{T}\boldsymbol{ranslation} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

- Example of translation

Homogeneous Coordinates

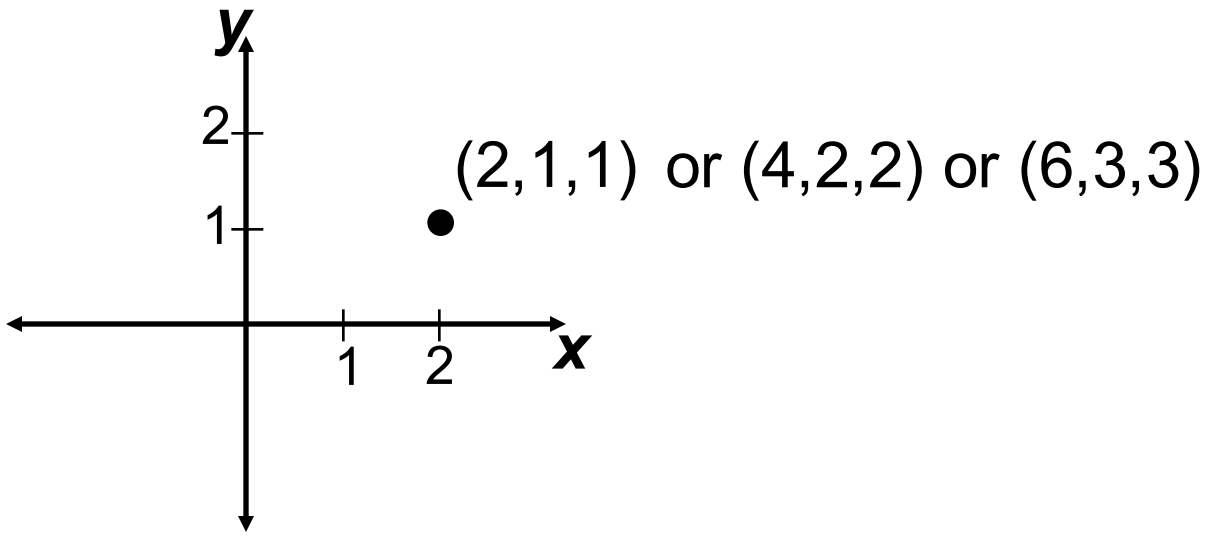
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$



Homogeneous Coordinates

- Add a 3rd coordinate to every 2D point
  - (x, y, w) represents a point at location (x/w, y/w)
  - (x, y, 0) represents a point at infinity
  - (0, 0, 0) is not allowed

Convenient coordinate system to represent many useful transformations



Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

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

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

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

Rotate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Affine Transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations
- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition
  - Models change of basis

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Projective Transformations

- Projective transformations …
  - Affine transformations, and
  - Projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition
  - Models change of basis

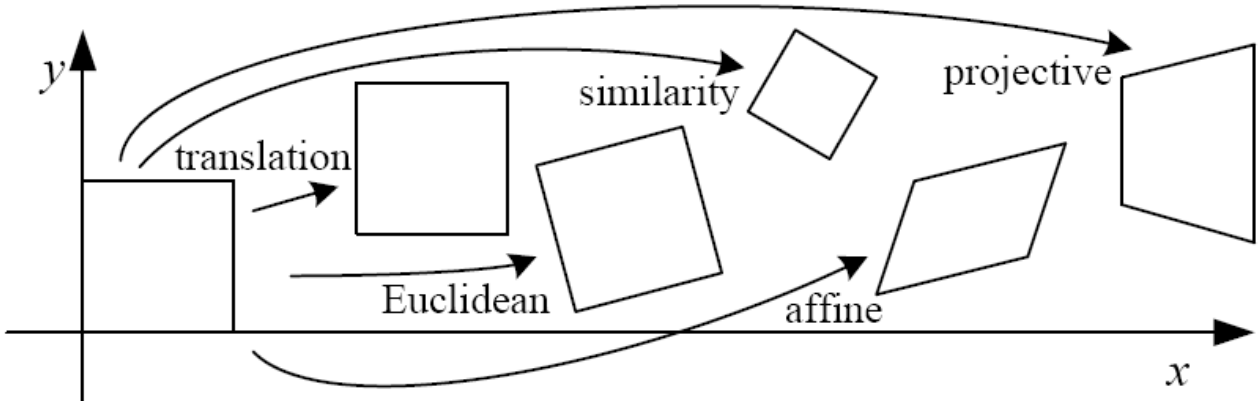
Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\mathbf{p'} = T(t_x,t_y) \quad R(\Theta) \quad S(s_x,s_y) \quad \mathbf{p}$$

2D Image Transformations

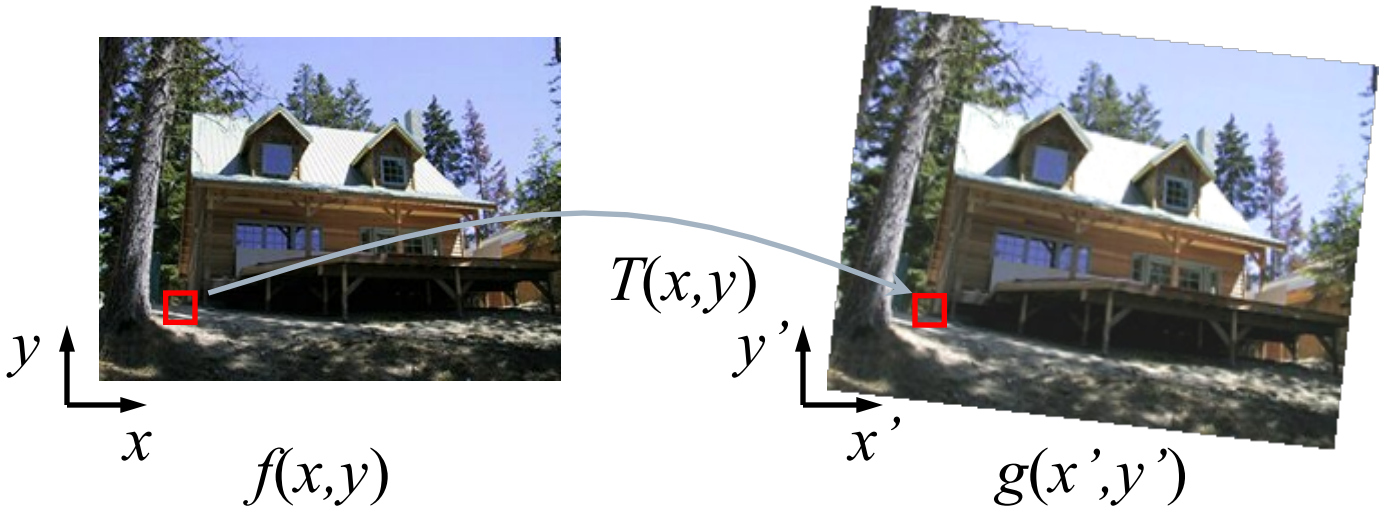


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

These transformations are a nested set of groups

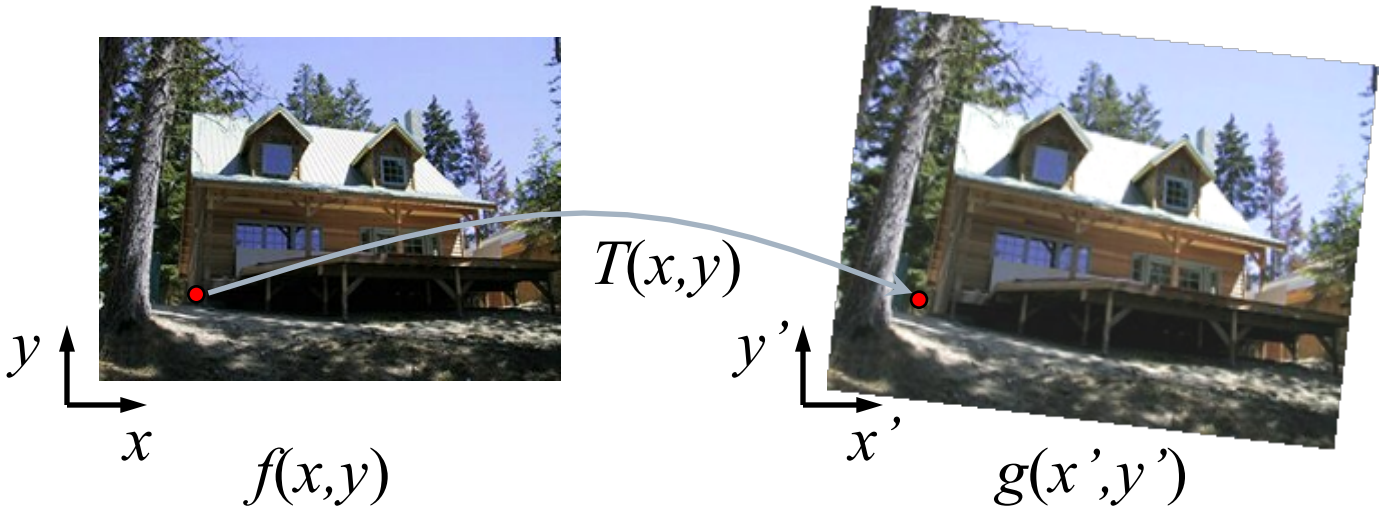
- Closed under composition and inverse is a member

Image Warping



- Given a coordinate transform $(x',y') = h(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(T(x,y))$?

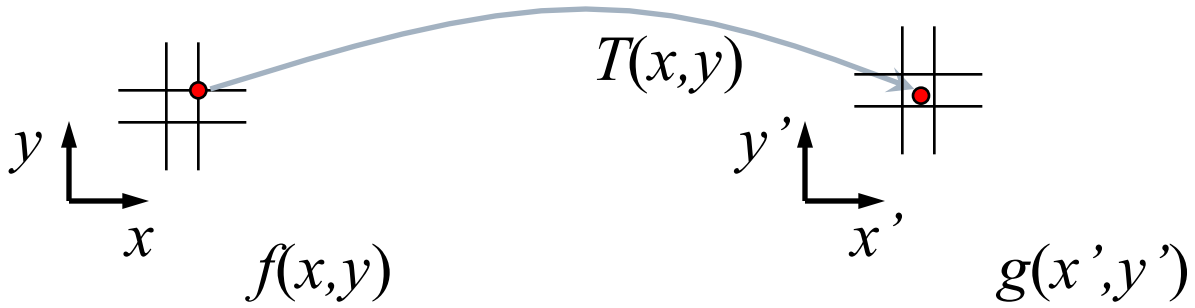
Forward Warping



- Send each pixel $f(x,y)$ to its corresponding location
- $(x',y') = T(x,y)$ in the second image

Q: what if pixel lands "between" two pixels?

Forward Warping



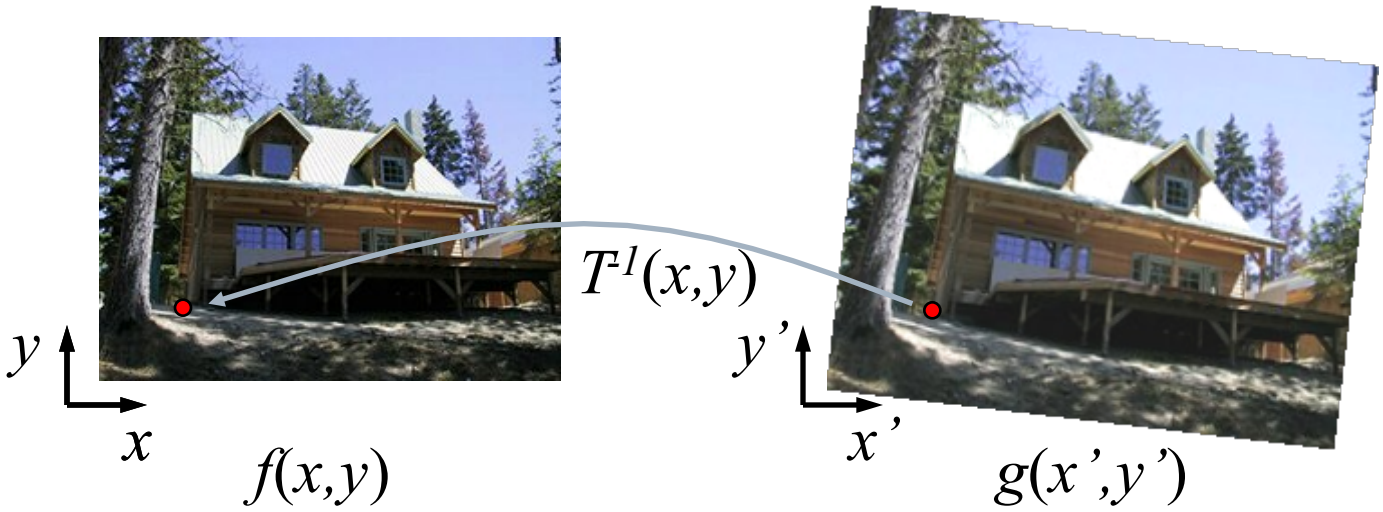
- Send each pixel $f(x,y)$ to its corresponding location
- $(x',y') = T(x,y)$ in the second image

Q: what if pixel lands "between" two pixels?

A: distribute color among neighboring pixels (x',y')

– Known as "splatting"

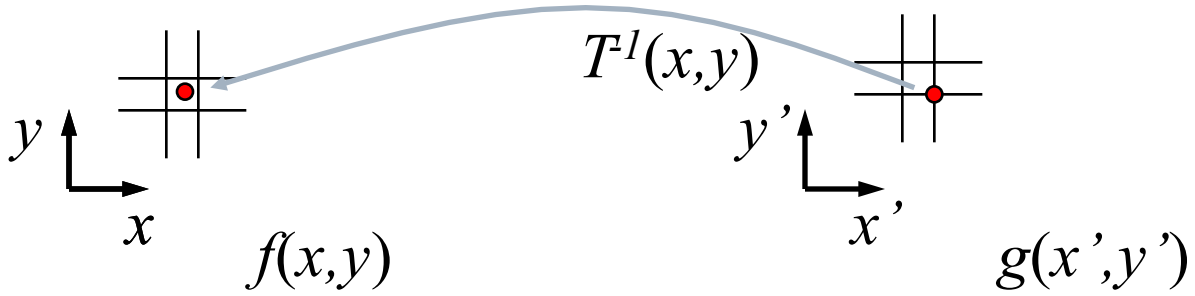
Inverse Warping



- Get each pixel $g(x',y')$ from its corresponding location
- $(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

Inverse Warping



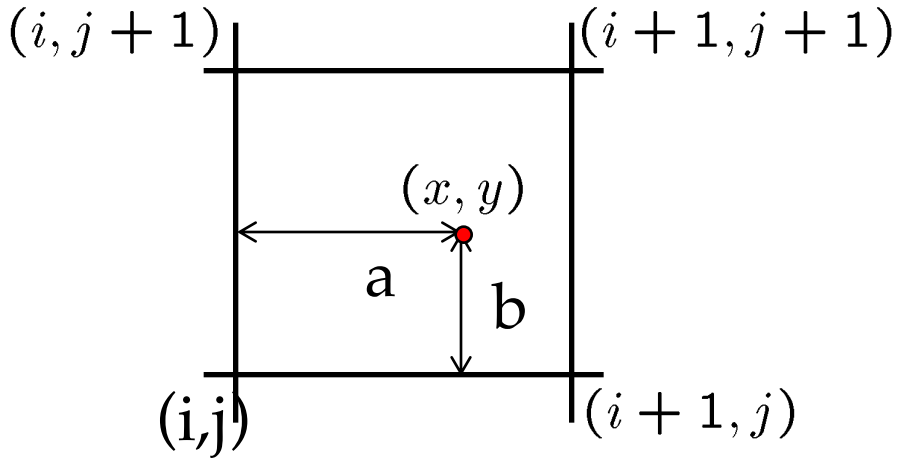
- Get each pixel $g(x',y')$ from its corresponding location
- $(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?
A: *Interpolate* color value from neighbors

– nearest neighbor, bilinear, Gaussian, bicubic

Bilinear Interpolation

- Sampling at *f(x,y):*



$$
\begin{aligned}
f(x,y) = \quad & (1-a)(1-b) && f[i,j] \\
& +a(1-b) && f[i+1,j] \\
& +ab && f[i+1,j+1] \\
& +(1-a)b && f[i,j+1]
\end{aligned}
$$

Forward vs. Inverse Warping

- Q: which is better?
- A: usually inverse—eliminates holes
  - however, it requires an invertible warp function—not always possible...