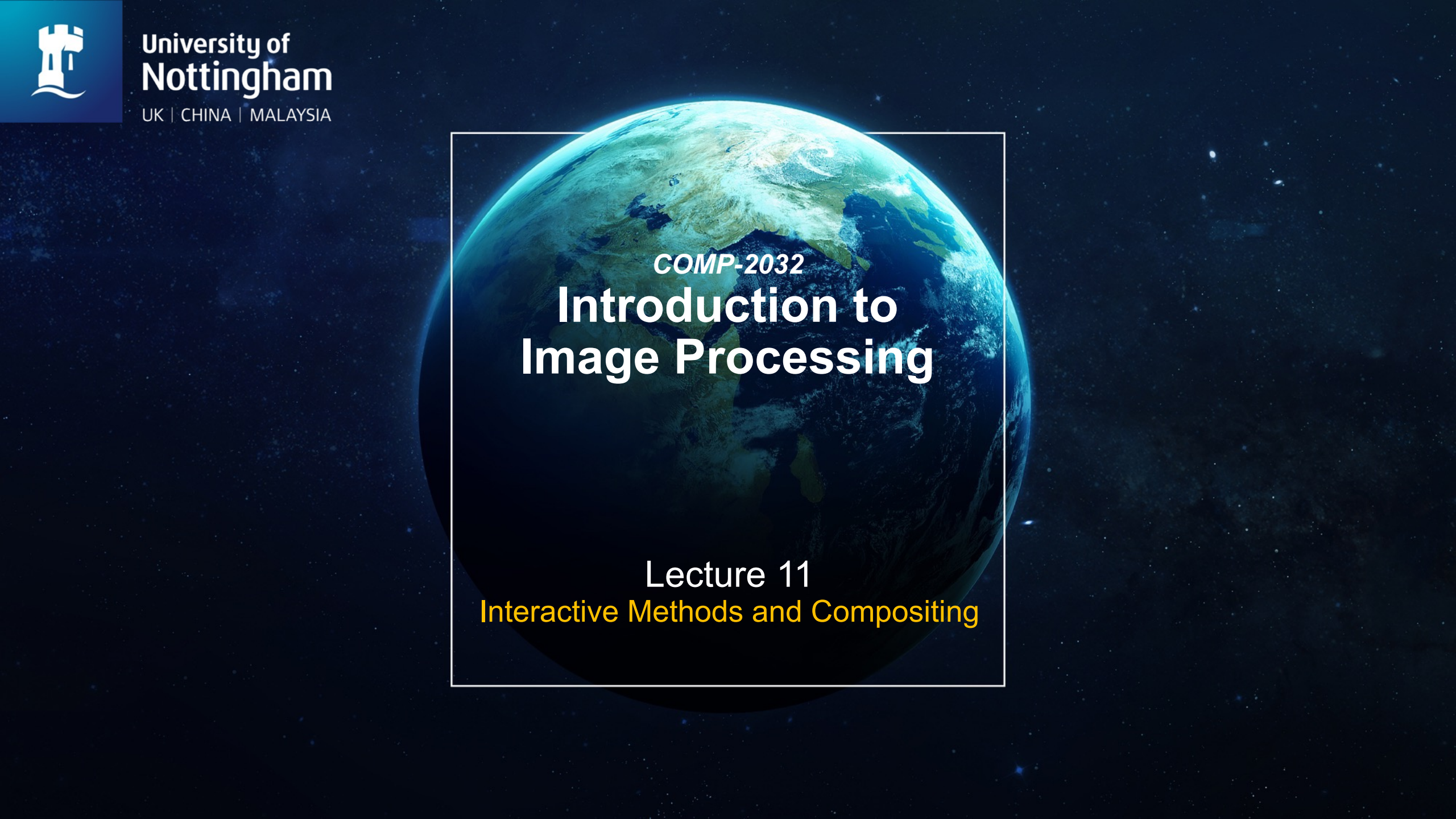

University of
Nottingham
UK | CHINA | MALAYSIA
COMP-2032
Introduction to
Image Processing
Lecture 11
Interactive Methods and Compositing

Learning Outcomes

IDENTIFY

1. Livewire Algorithm
2. Blending Images
3. Geometric Transformation

University of Nottingham
UK | CHINA | MALAYSIA

LiveWire Algorithm

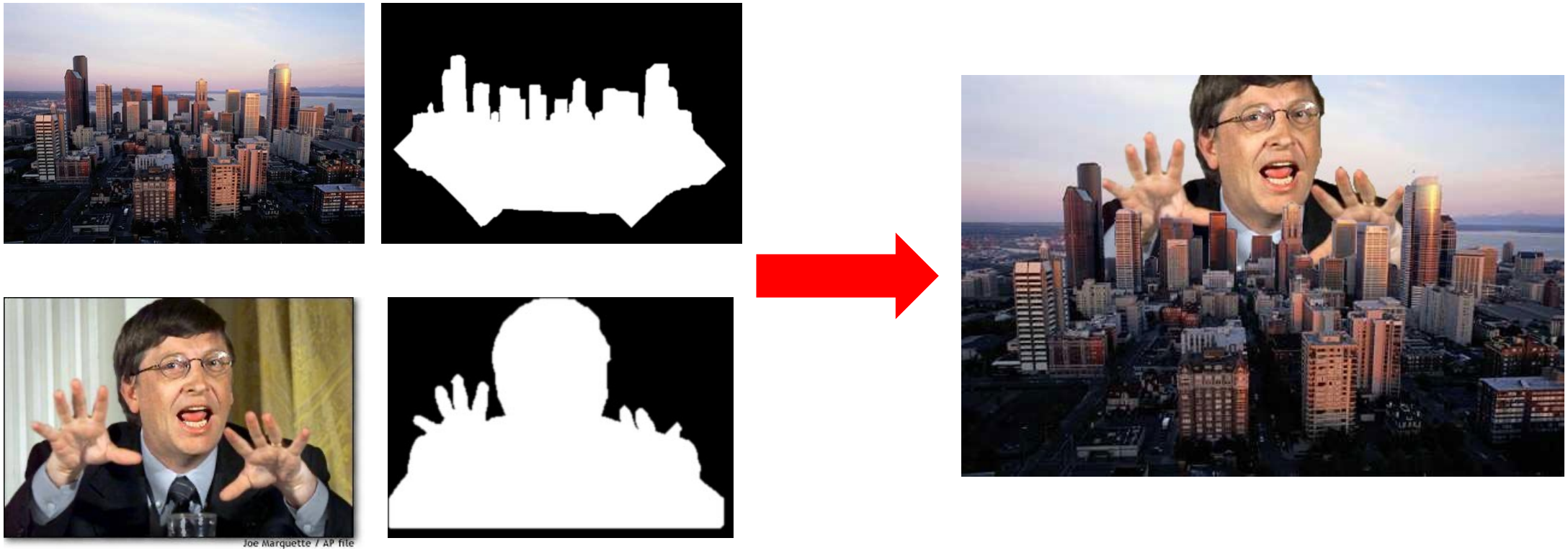
Compositing

© NASA

ACK: Prof. Tony Pridmore, UNUK

Compositing

Extract sprites using e.g. *intelligent scissors* in Photoshop



Blend them into the composite (in the right order)

ACK: Prof. Tony Pridmore, UNUK

Intelligent Scissors

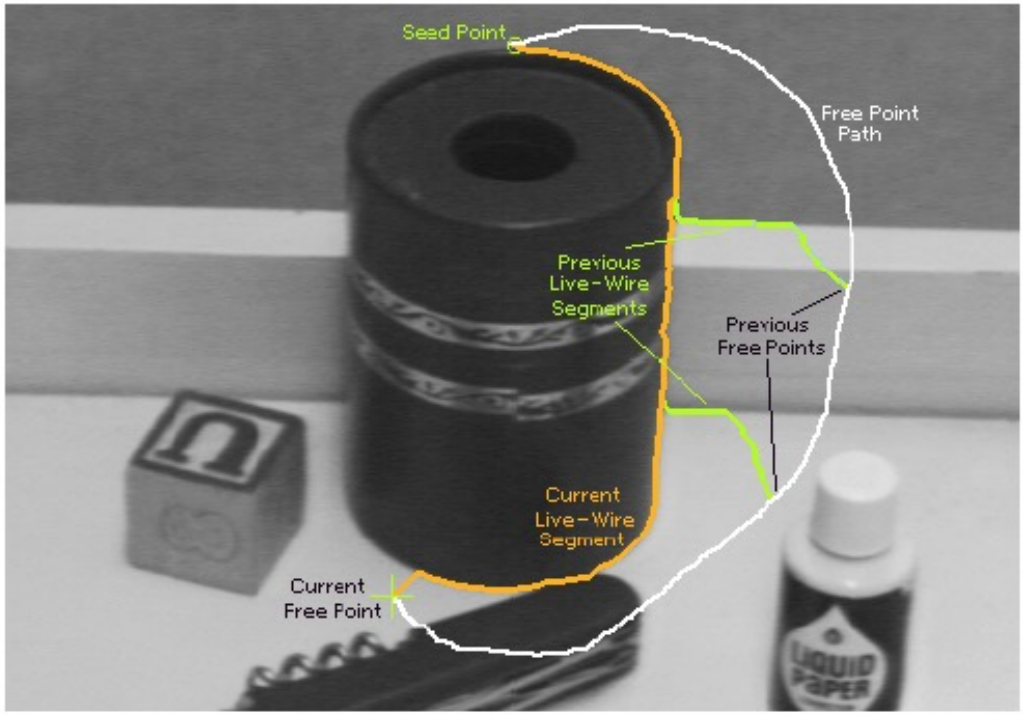


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

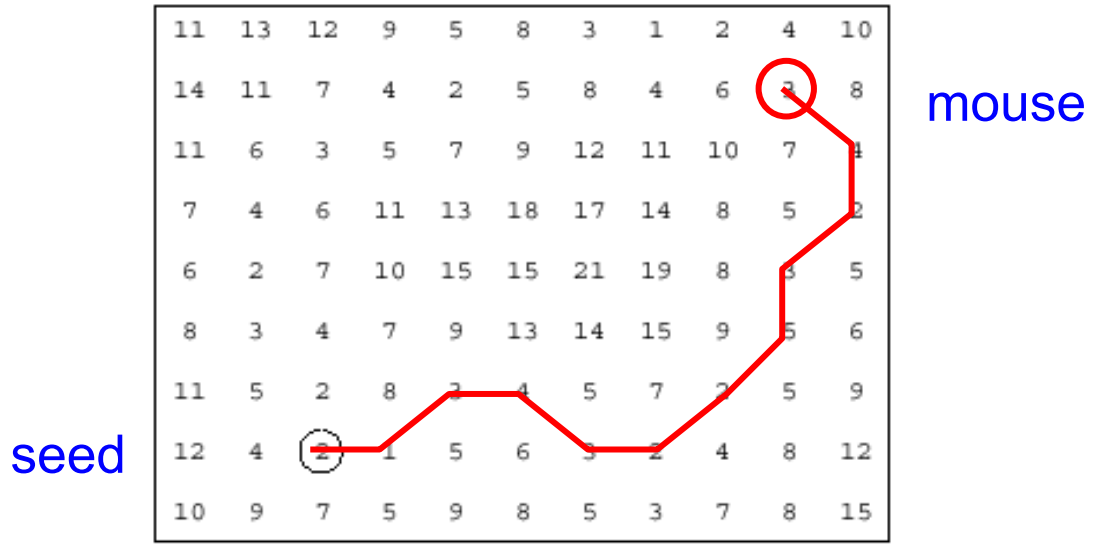
E. N. Mortensen and W. A. Barrett, Intelligent Scissors for Image Composition, in *ACM Computer Graphics (SIGGRAPH `95)*, 1995

ACK: Prof. Tony Pridmore, UNUK

Intelligent Scissors

We need a path from seed to mouse that follows the object boundary as closely as possible

- Define a path that stays as close as possible to edges
- Quantify the ”cost” of including each pixel in the path
- Edge pixels have low cost
- Find lowest cost path from seed to cursor

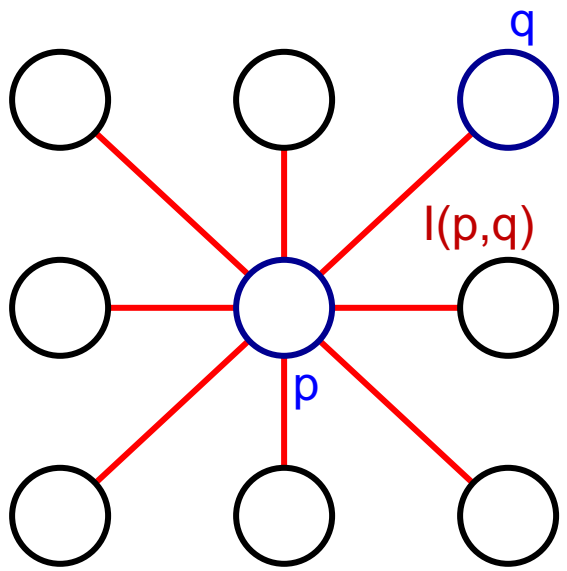


ACK: Prof. Tony Pridmore, UNUK

Intelligent Scissors

Treat the image as a graph

- Node for every pixel *p*
- Link between every adjacent pair of pixels, *p,q*
- Cost *I(p,q)* for each link



Cost combines several edge measures

- Laplacian Zero-Crossing, $f_Z(q)$
- Gradient Magnitude, $f_G(q)$, Gradient Direction, $f_D(p,q)$

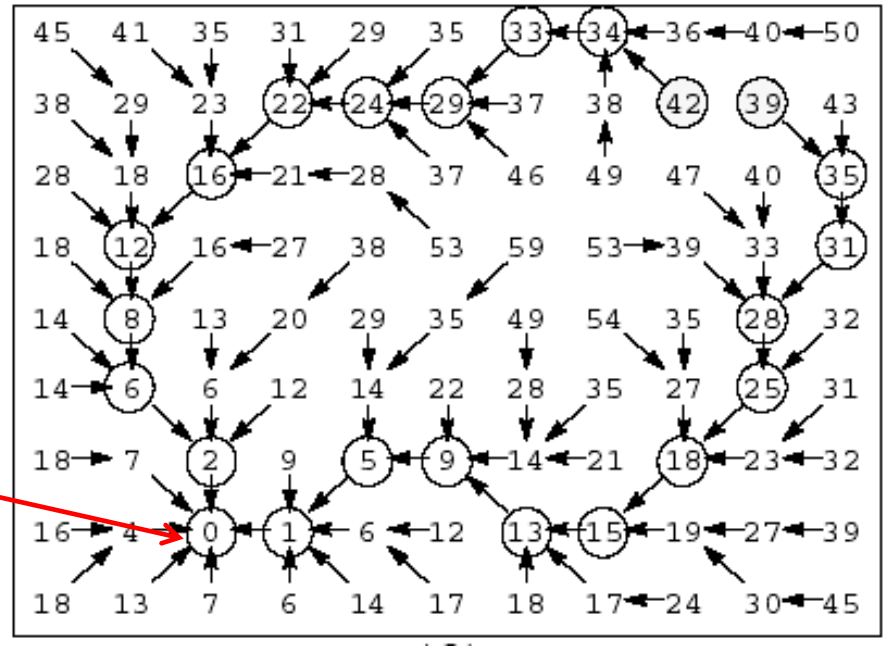
$$I(p,q) = \omega_Z \cdot f_Z(q) + \omega_D \cdot f_D(p,q) + \omega_G \cdot f_G(q)$$

p is an edge: we want q to be an edge too, with a high gradient, and the gradient direction to be similar at p and q

ACK: Prof. Tony Pridmore, UNUK

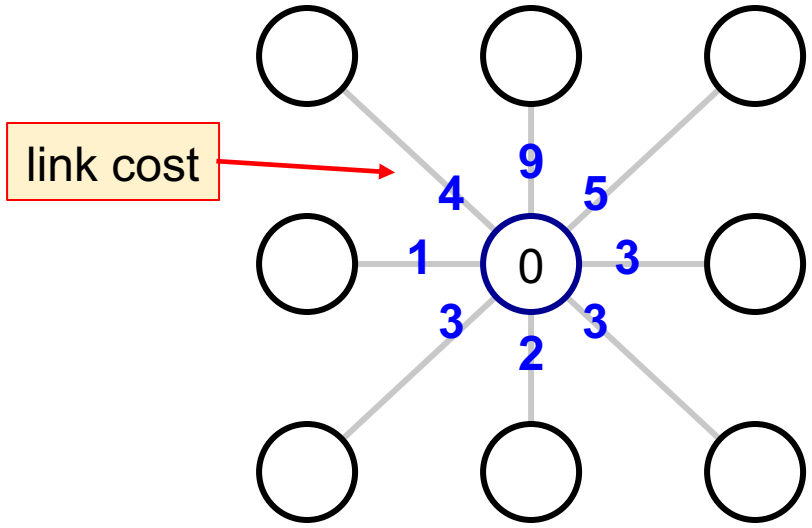
The LiveWire Algorithm

- Finds the minimum cost path between seed point and every point in the graph
- Output is a pointer at each pixel indicating the minimum cost path back to the seed
- As the mouse moves over the image the path from the current pixel is followed back to the seed



ACK: Prof. Tony Pridmore, UNUK

LiveWire

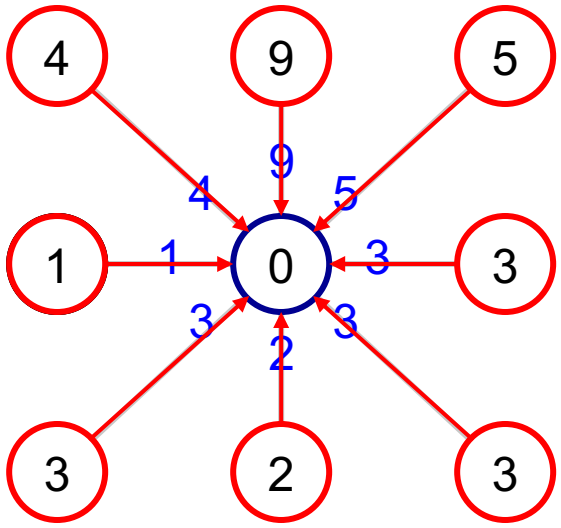


1. Init code cost to ∞, set *p = seed point*, *cost(p) = 0*
2. Expand *p* as follows:
   - For each of *p*'s neighbours *q* that are not expanded
     - Set cost(*q*) = min(cost(*p*) + c_{pq}, cost(*q*))

Algorithm

ACK: Prof. Tony Pridmore, UNUK

LiveWire

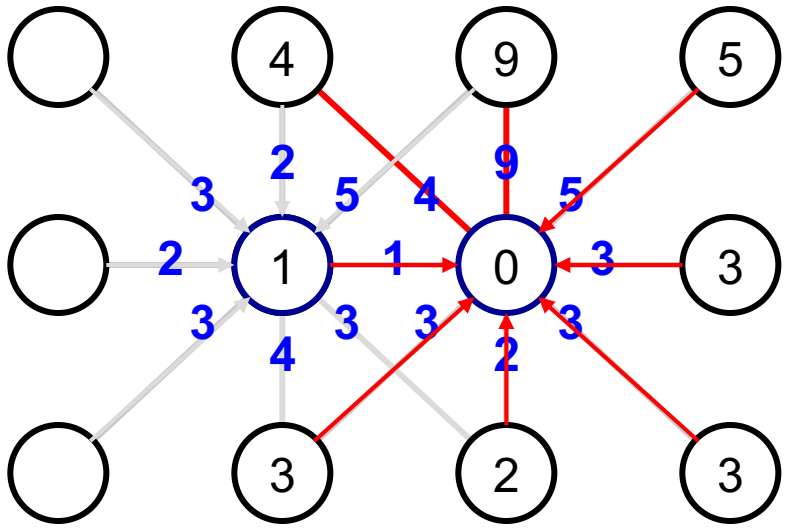


1. Init code cost to ∞, set *p = seed point*, *cost(p) = 0*
2. Expand *p* as follows:
   - For each of *p*'s neighbours *q* that are not expanded
     - Set cost(*q*) = min(cost(*p*) + c_{pq}, cost(*q*))
       - ✓ if *q*'s cost changed, make *q* point back to *p*
     - Put *q* on the ACTIVE list (if not already there)

Algorithm

ACK: Prof. Tony Pridmore, UNUK

LiveWire

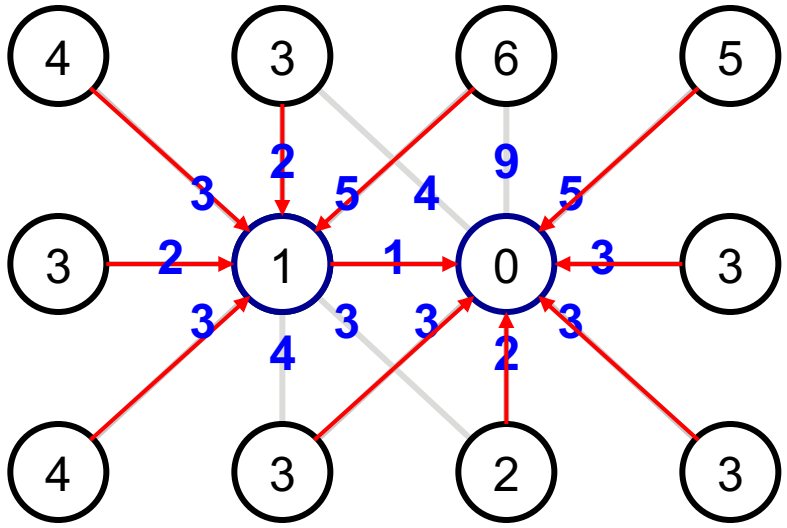


1. Init code cost to ∞, set *p = seed point*, *cost(p) = 0*
2. Expand *p* as follows:
   - For each of *p*'s neighbours *q* that are not expanded
     - ➢ Set cost(*q*) = min(cost(*p*) + c_{pq}, cost(*q*))
       - ✓ if *q*'s cost changed, make *q* point back to *p*
     - ➢ Put *q* on the ACTIVE list (if not already there)
3. Set *r* = node with minimum cost of the ACTIVE list
4. Repeat Step 2 for *p* = *r*

Algorithm

ACK: Prof. Tony Pridmore, UNUK

LiveWire

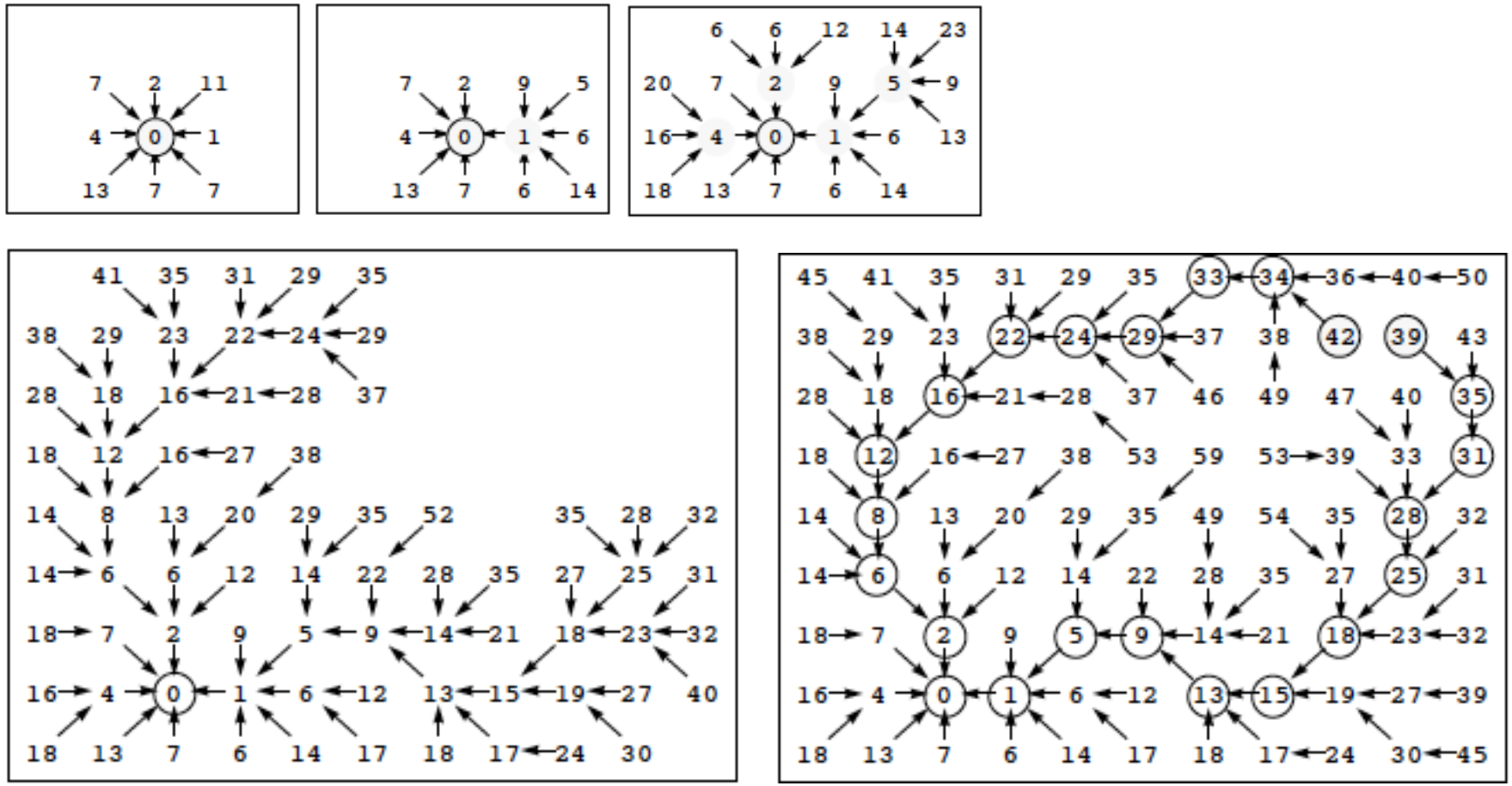


1. Init code cost to ∞, set *p = seed point*, *cost(p) = 0*
2. Expand *p* as follows:
   - For each of *p*'s neighbours *q* that are not expanded
     - Set cost(*q*) = min(cost(*p*) + c_{pq}, cost(*q*))
       - ✓ if *q*'s cost changed, make *q* point back to *p*
     - Put *q* on the ACTIVE list (if not already there)
3. Set *r* = node with minimum cost of the ACTIVE list
4. Repeat Step 2 for *p* = *r*

Algorithm

ACK: Prof. Tony Pridmore, UNUK

LiveWire



ACK: Prof. Tony Pridmore, UNUK

Cooling

The tool allows multiple seeds, but all seeds must be manually selected

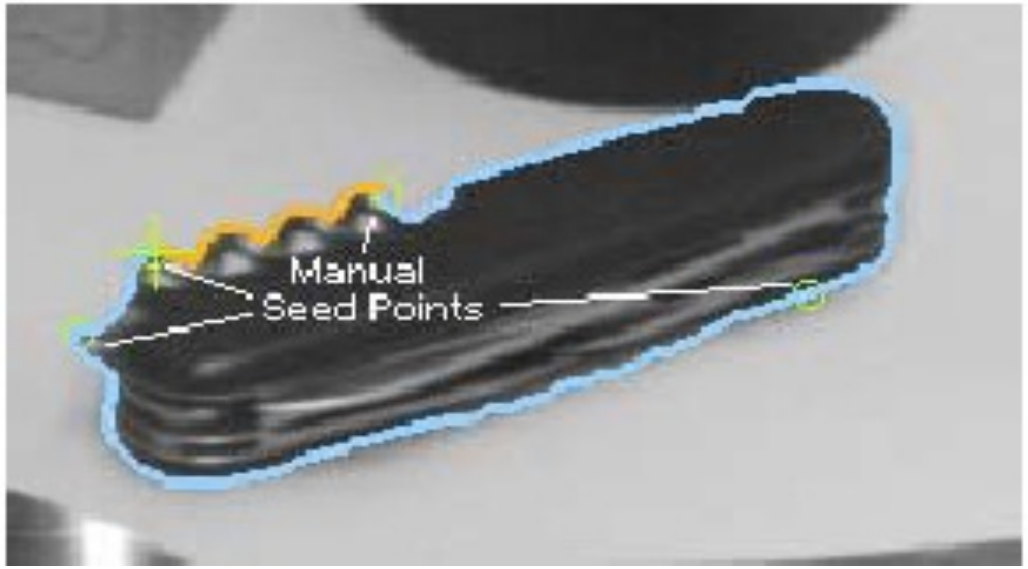
- Complex objects may require many seed points

Can detect seed points automatically

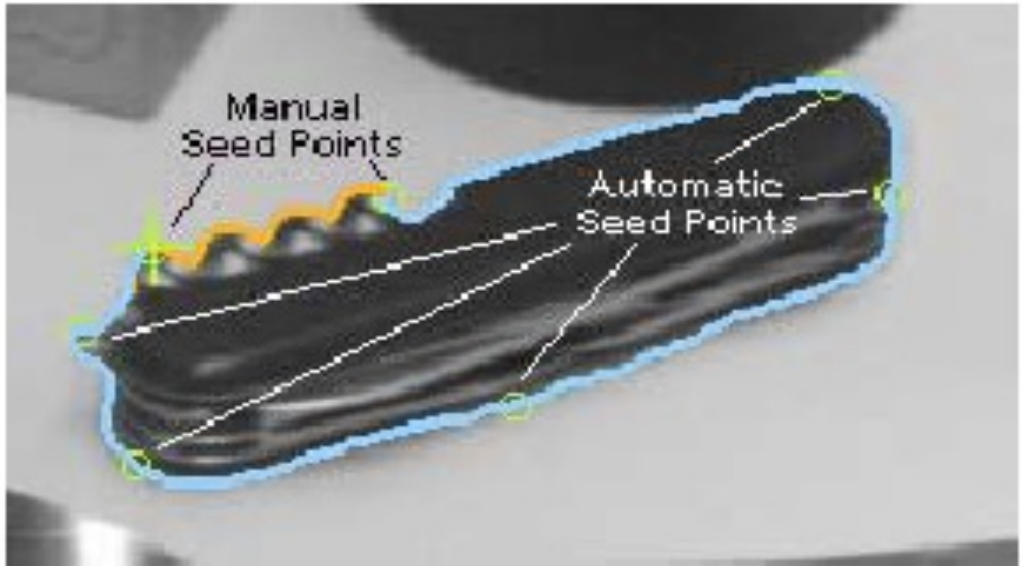
- As the user wraps the object, the early sections tend to (and need to) be fixed
- Path segments that remain fixed over N cursor movements are considered stable
- New seed points are created at the ends of stable segments

ACK: Prof. Tony Pridmore, UNUK

Cooling



Basic version: several manual seed points are needed to capture the shape



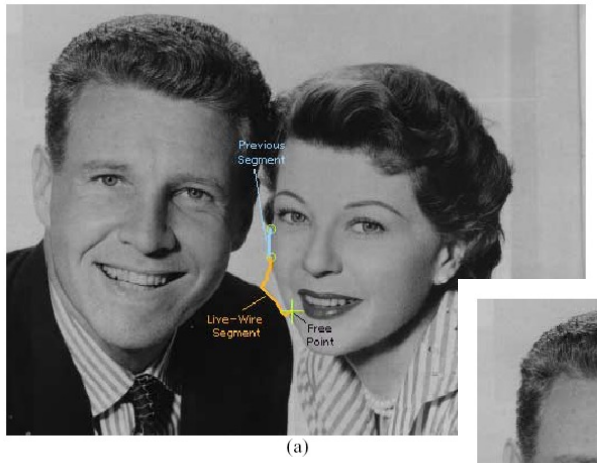
With cooling: two seeds are enough, the rest are created automatically

ACK: Prof. Tony Pridmore, UNUK

Interactive Dynamic Training

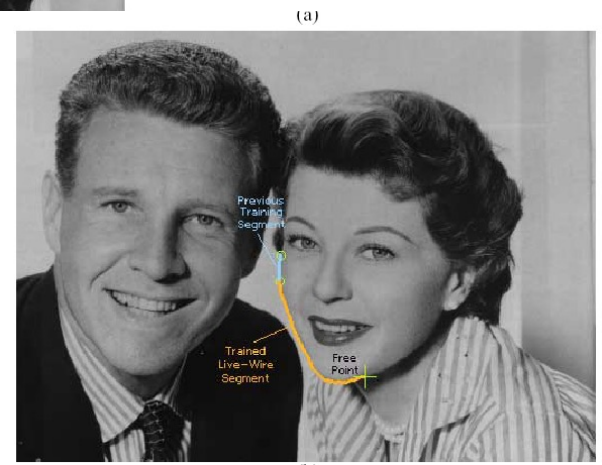
Some objects have stronger edges than other

If the desired edge is weaker than a nearby edge, then the path ‘jumps’ over to the stronger edge



Train the gradient magnitude to desire the weaker edge

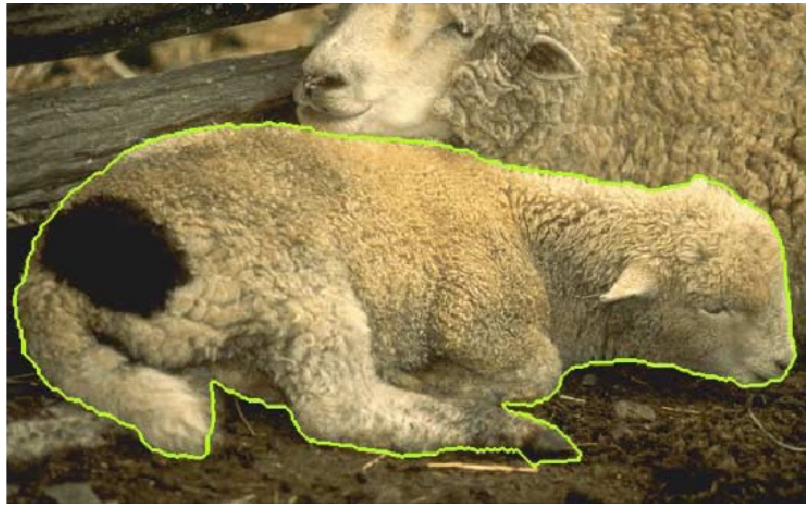
- Use a sample of good path to train gradient magnitude
- i.e., change the weighting of the gradient magnitude term
- Update sample as path moves along the desired edge



Allow user to enable and disable training as needed

ACK: Prof. Tony Pridmore, UNUK

Results

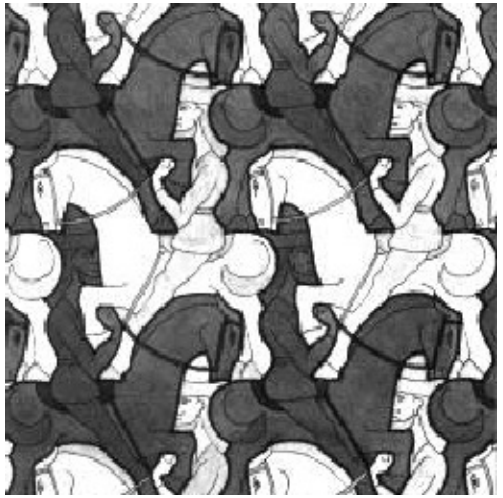


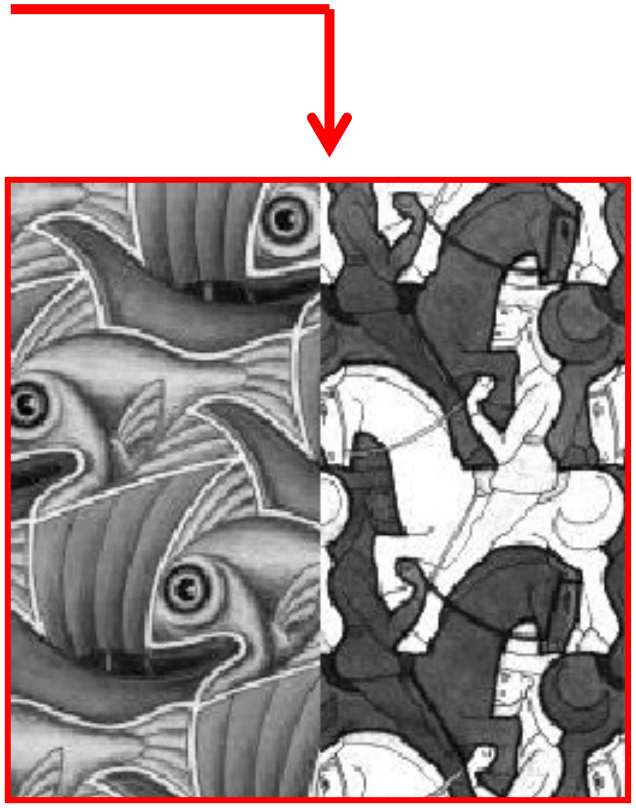
ACK: Prof. Tony Pridmore, UNUK

University of Nottingham
UK | CHINA | MALAYSIA

Blending Images

Blending

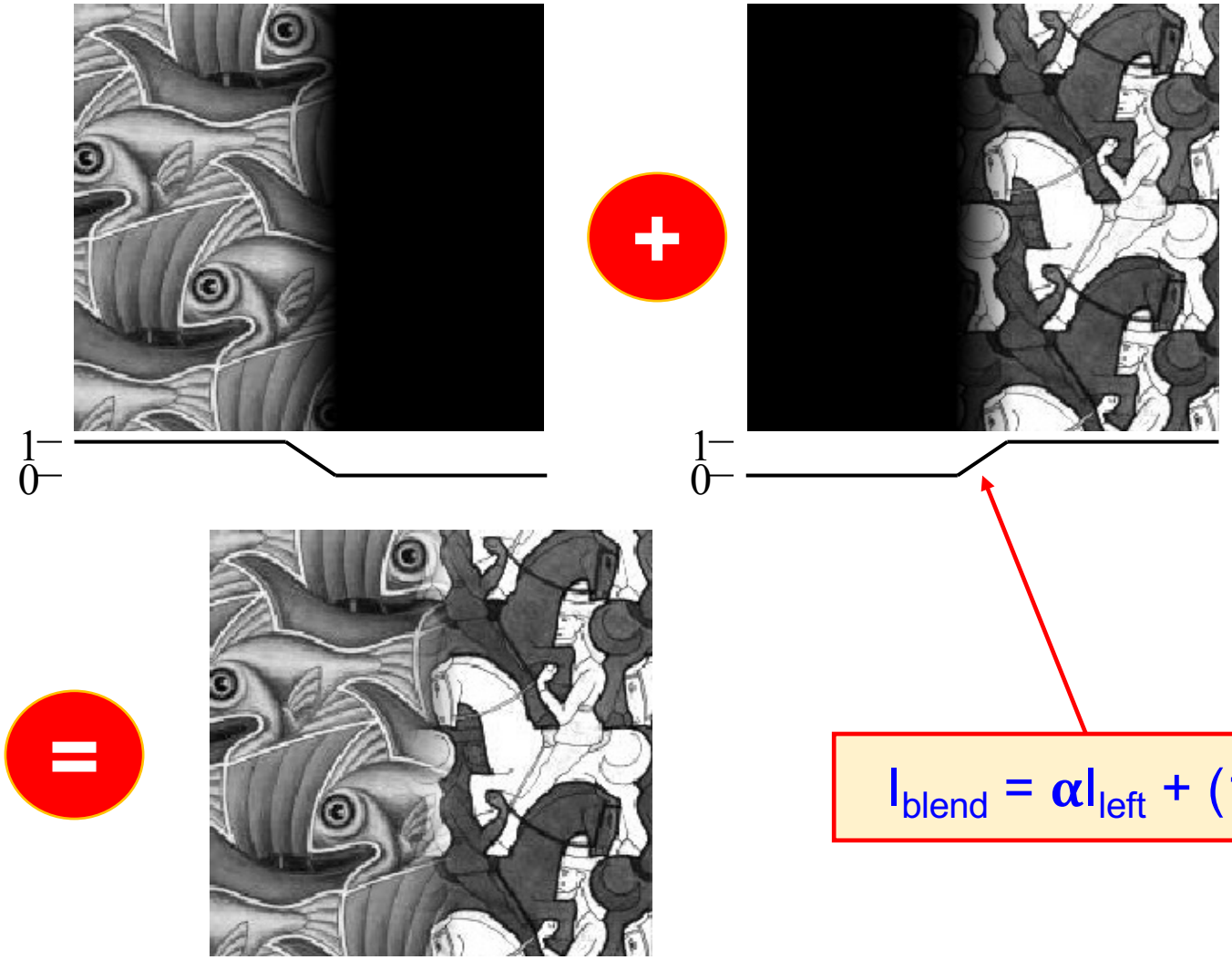




- Just putting sprites next to each other isn't enough
- Boundaries are clear, and attract the eye

ACK: Prof. Tony Pridmore, UNUK

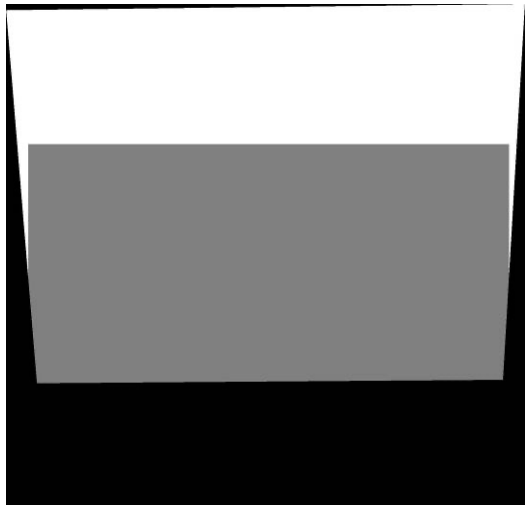
Alpha-Blending/Feathering



$$I_{blend} = \alpha I_{left} + (1-\alpha) I_{right}$$

ACK: Prof. Tony Pridmore, UNUK

Setting α: Simple averaging

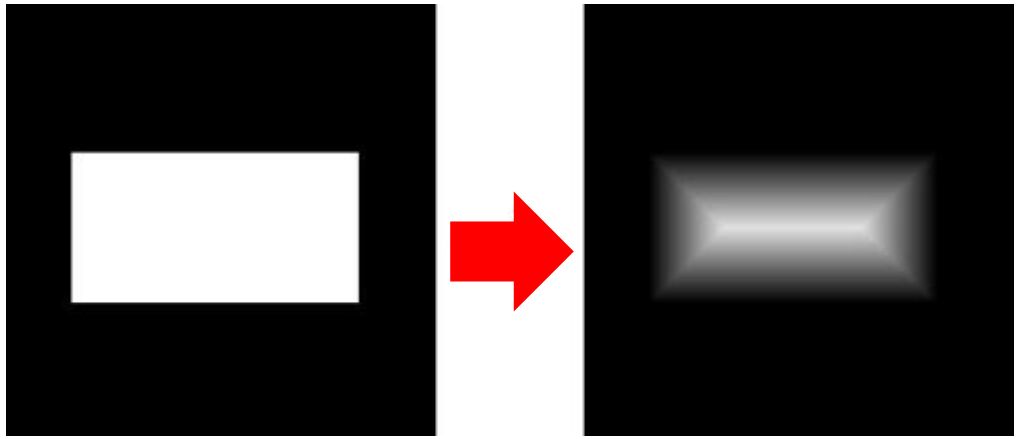


α = 0.5 in the overlap region

ACK: Prof. Tony Pridmore, UNUK

Distance Transforms

Assign a non-negative integer each pixel giving the distance from that pixel to e.g., nearest black pixel

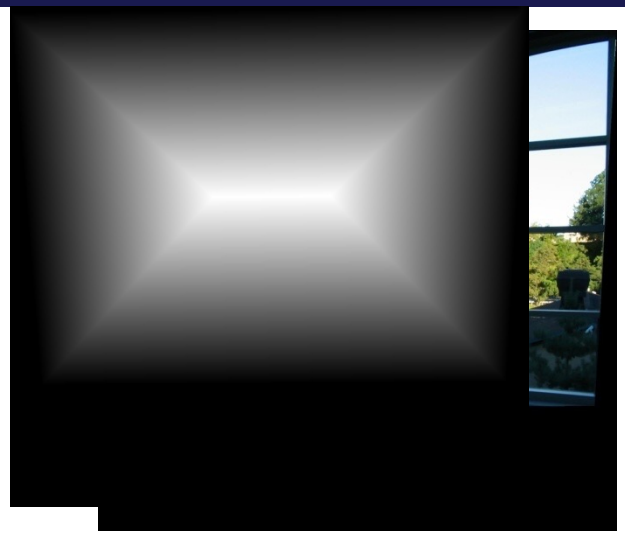


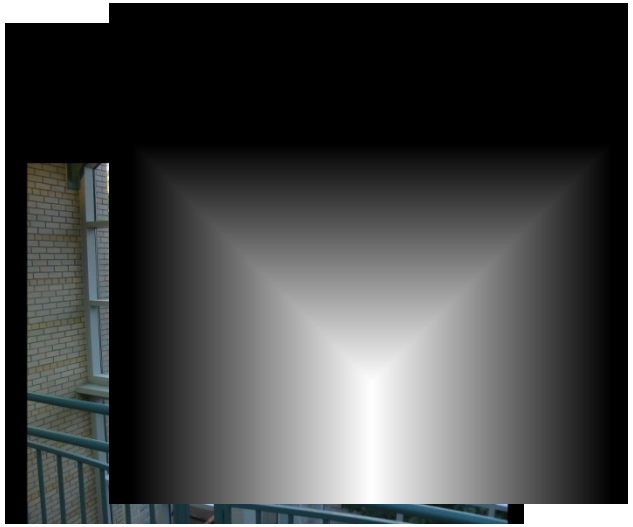
- A range of distance measure exist
- Characterise the shape of a binary image
- Can be used to guide blending

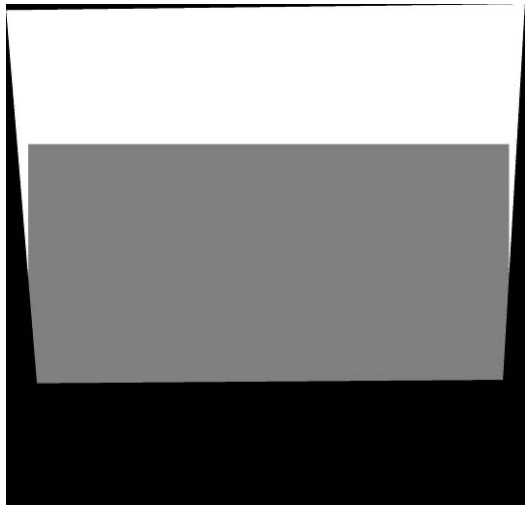
ACK: Prof. Tony Pridmore, UNUK

Setting α: Centre Seam

Distance Transform
bwdist





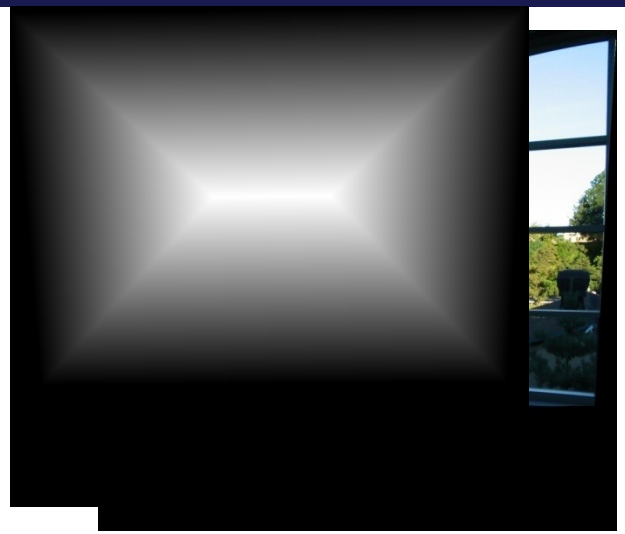


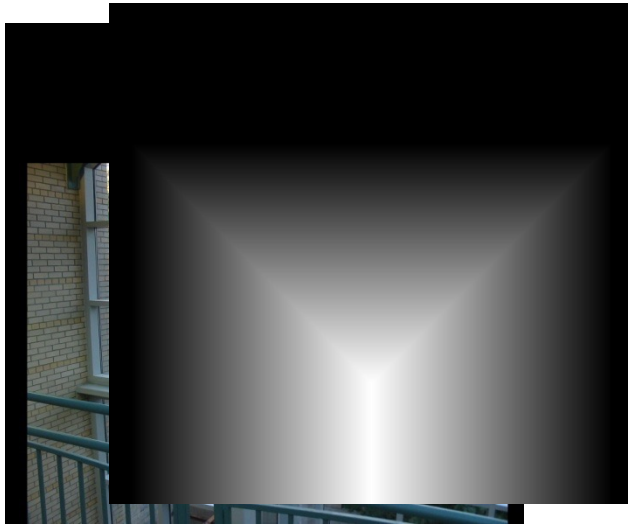
α = logical(dt1>dt2)

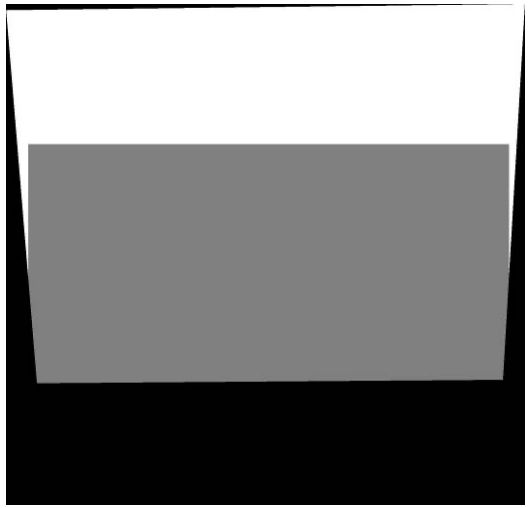
Each pixel uses the image whose centre seam it is closest to

ACK: Prof. Tony Pridmore, UNUK

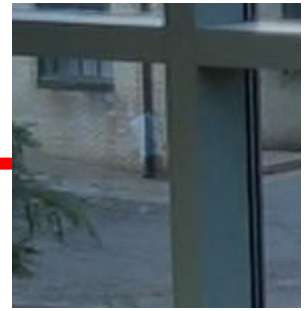
Setting α: Centre Weighting







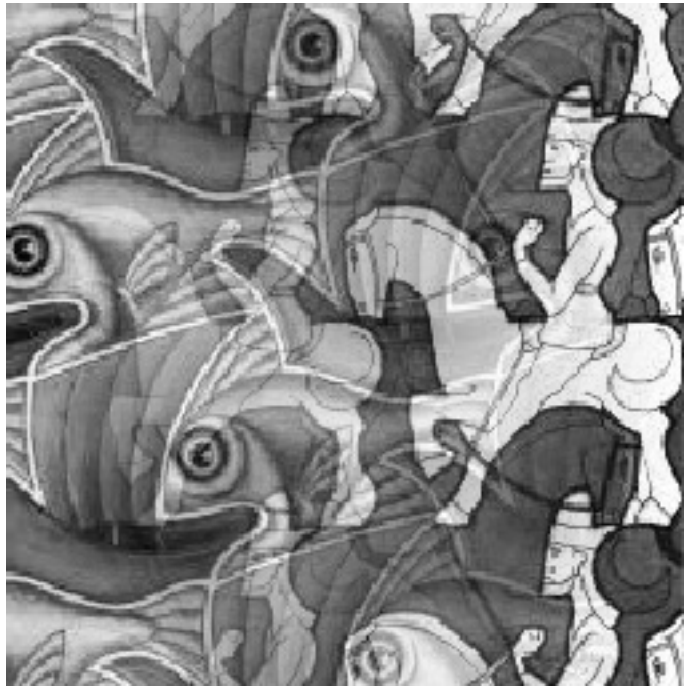
$\alpha = dt1 / (dt1 + dt2)$

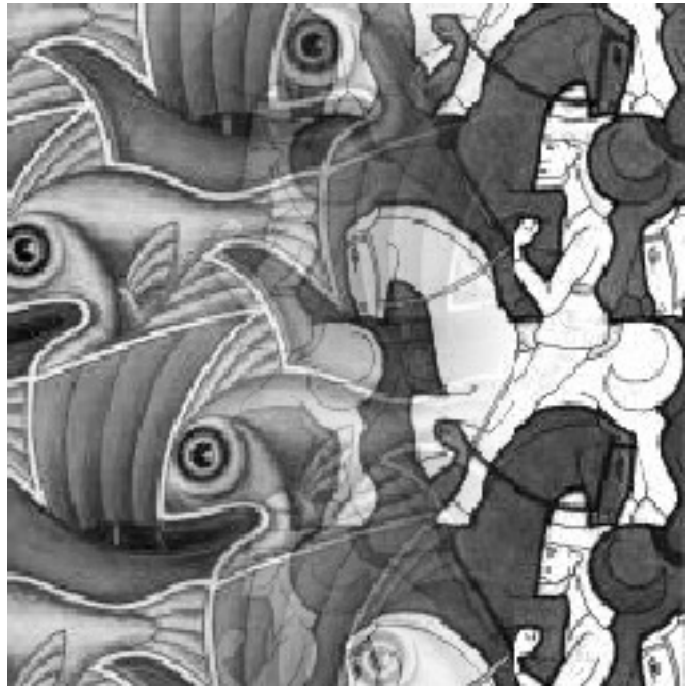


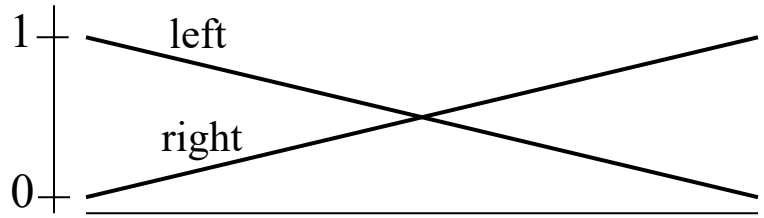
Ghost!

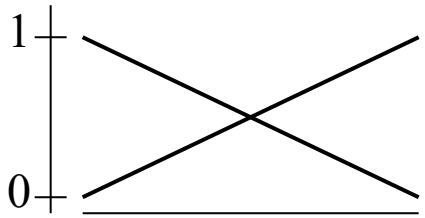
ACK: Prof. Tony Pridmore, UNUK

The Effect of Window Size



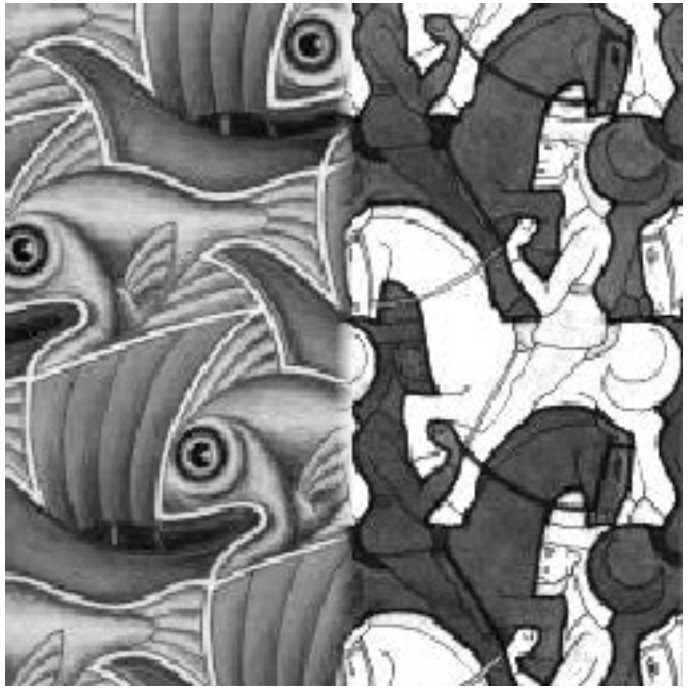


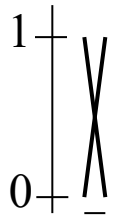


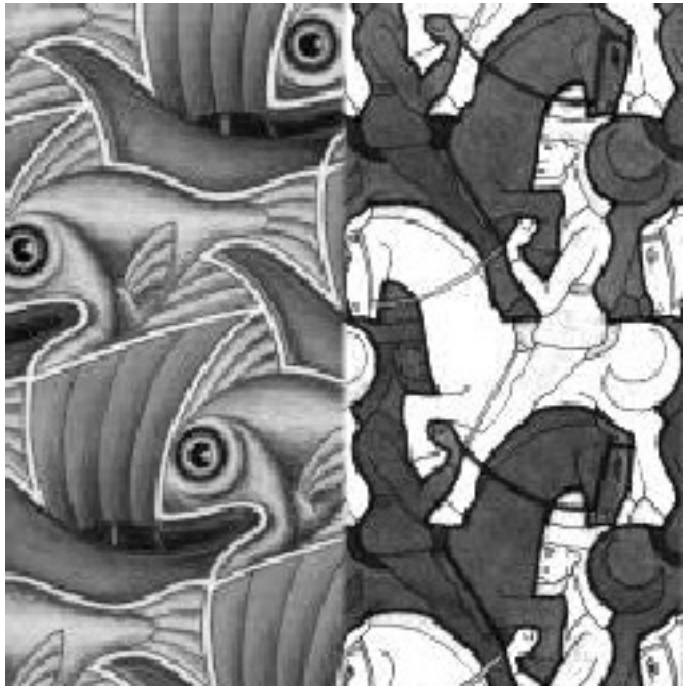


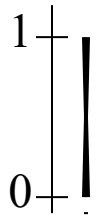
ACK: Prof. Tony Pridmore, UNUK

The Effect of Window Size



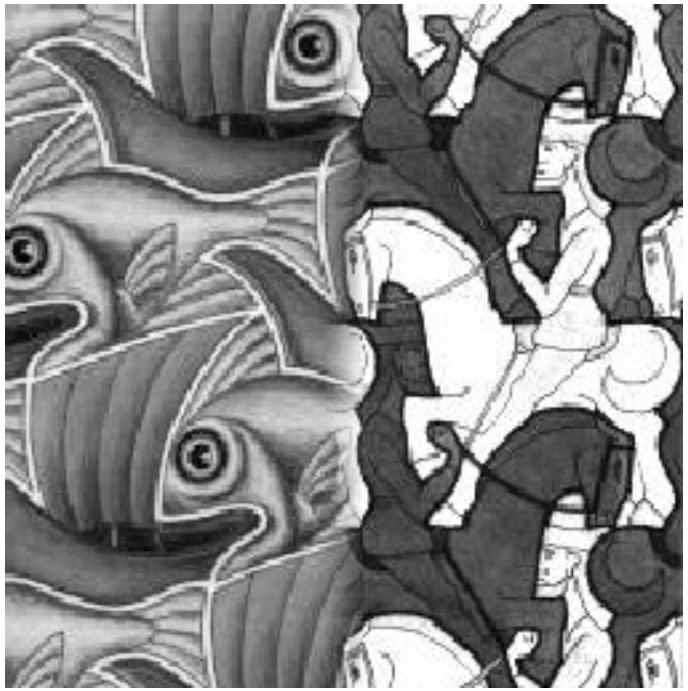


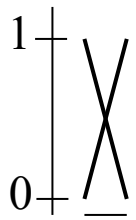




ACK: Prof. Tony Pridmore, UNUK

Good Window Size





The optimal window size leaves the blend smooth, but not ghosted

To avoid seams:

- Window = size of largest prominent feature

To avoid ghosting:

- Window <= 2* size of largest prominent feature

ACK: Prof. Tony Pridmore, UNUK

Break

University of Nottingham
UK | CHINA | MALAYSIA

Geometric Transformations

Geometric Transformations

Transforms we have looked so far effect the content of the image array, but leave the spatial arrangement of the pixels intact, geometric or rubber-sheet transformations do not

Two stages:

1. Spatial transformation of pixel coordinates
2. Assign intensity values to new pixels

ACK: Prof. Tony Pridmore, UNUK

Spatial Transformations

Coordinate transforms are expressed as *(x,y) = T(v,w)*

- e.g., *(x,y) = (v/2, w/2)* shrinks the image to half its size in both dimensions

One of the most common is the affine transform

$$[x \;\; y \;\; 1] = [v \;\; w \;\; 1]\,\mathbf{T} = [v \;\; w \;\; 1] \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

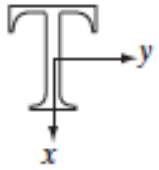
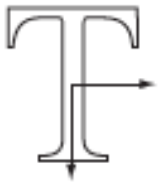
Which can scale, rotate, tanslate or shear an image

Those can be applied separately, or a sequence of transformations can be applied by a single 3 x 3 matrix

The order in which transforms are applied matters

Note

ACK: Prof. Tony Pridmore, UNUK

Transformation Name	Affine Matrix, T	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
Scaling	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\cos\theta + w\sin\theta$	
Translation	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Shear (vertical)	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v + s_v w$ $y = w$	
Shear (horizontal)	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

ACK: Prof. Tony Pridmore, UNUK

Applying Transformations

Forward Mapping

- Scan the input image, compute the (new) position of each pixel in the output image
- What do you do if two input pixels map to one output pixel?
- Some output locations may not be allocated a pixel at all

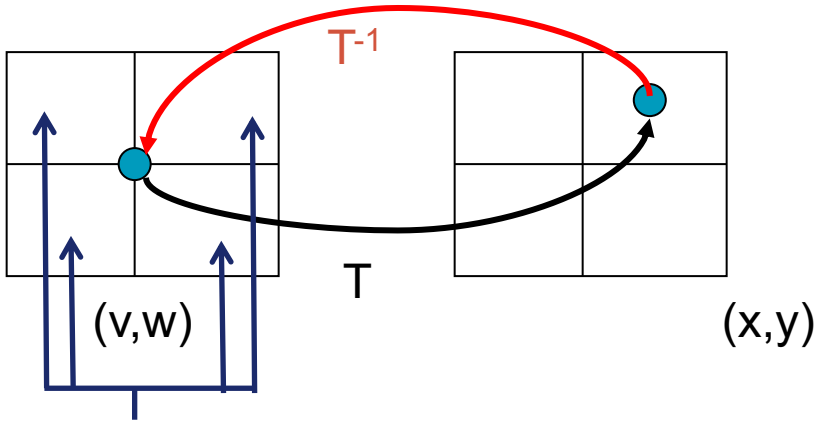
Inverse Mapping

- Scan the output image, use the inverse of the transformation $(v,w) = T^{-1}(x,y)$ to work out which point in the input image maps to each output pixel location
- Use image interpolation to compute a new (output) pixel value from nearby (input) pixel values

ACK: Prof. Tony Pridmore, UNUK

Image Interpolation

In general, transformed pixel coordinates will not map neatly onto the coordinates of a regular rectangular array



We must use nearby (input) intensity values to compute a suitable output value at (x,y)

ACK: Prof. Tony Pridmore, UNUK

Image Interpolation

Nearest neighbour interpolation — Just pick the nearest pixel value and use that

Bilinear interpolation — take the four nearest neighbours and assume intensity varies linearly, then

$$v(x, y) = ax + by + cxy + d$$

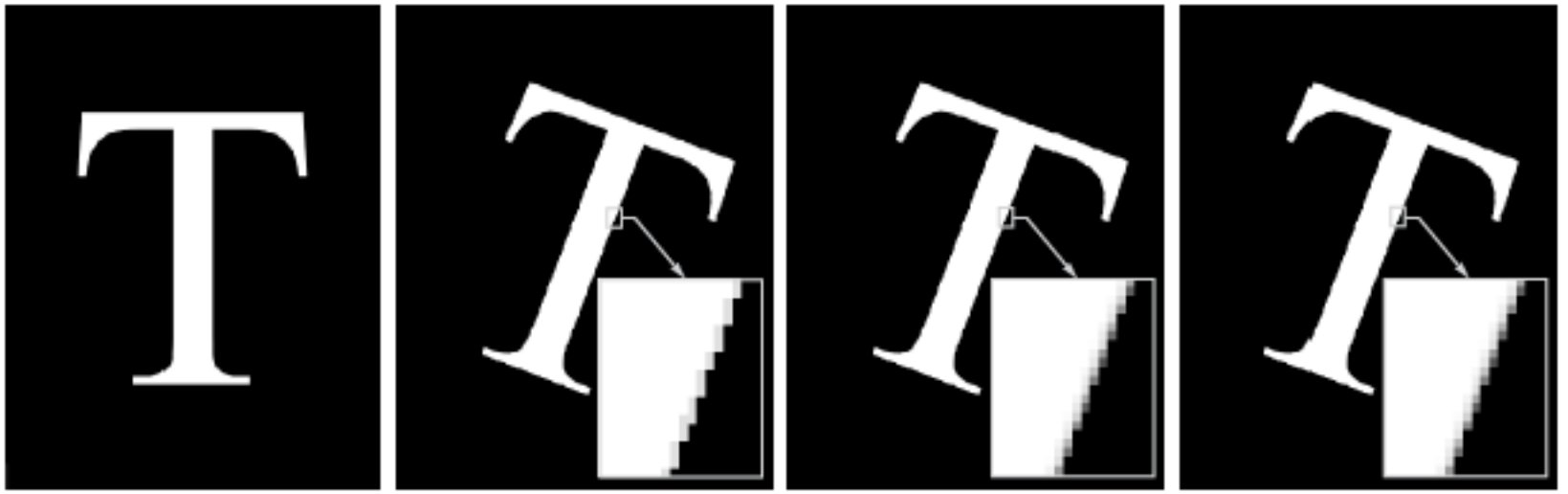
Bicubic interpolation — take the 16 nearest neighbours, fit a cubic, so

$$v(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

Bicubic interpolation is the standard, used in e.g., Photoshop

ACK: Prof. Tony Pridmore, UNUK

Image Interpolation



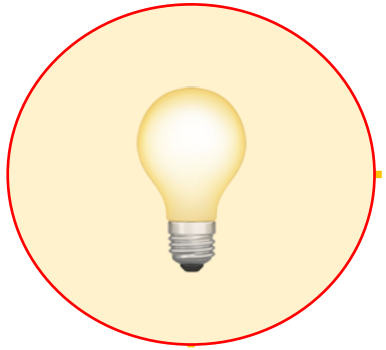
a b c d

FIGURE 2.36 (a) A 300 dpi image of the letter T. (b) Image rotated 21° using nearest neighbor interpolation to assign intensity values to the spatially transformed pixels. (c) Image rotated 21° using bilinear interpolation. (d) Image rotated 21° using bicubic interpolation. The enlarged sections show edge detail for the three interpolation approaches.

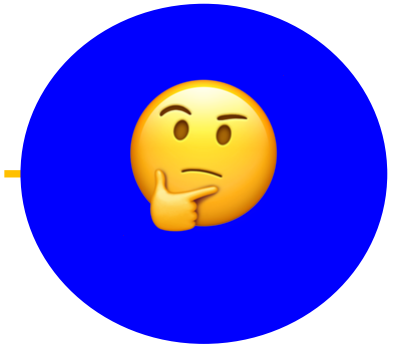
From Gonzalez & Woods

ACK: Prof. Tony Pridmore, UNUK

Summary



1. Livewire Algorithm
2. Blending Images
3. Geometric Transformation




University of
Nottingham
UK | CHINA | MALAYSIA
Questions

University of Nottingham
UK | CHINA | MALAYSIA

NEXT:

Finale & Revision