Image Warping

Adam Finkelstein & Tim Weyrich
Princeton University
COS 426, Spring 2008

Image Processing

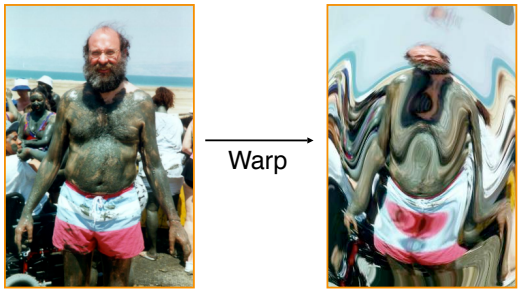
- Pixel operations
  - Add random noise
  - Add luminance
  - Add contrast
  - Add saturation
- Filtering
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median
- Quantization
  - Uniform Quantization
  - Floyd-Steinberg dither
- Warping
  - Scale
  - Rotate
  - Warp
- Combining
  - Composite
  - Morph

Image Processing

- Pixel operations
  - Add random noise
  - Add luminance
  - Add contrast
  - Add saturation
- Filtering
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median
- Quantization
  - Uniform Quantization
  - Floyd-Steinberg dither
- Warping
  - Scale
  - Rotate
  - Warp
- Combining
  - Composite
  - Morph

Image Warping

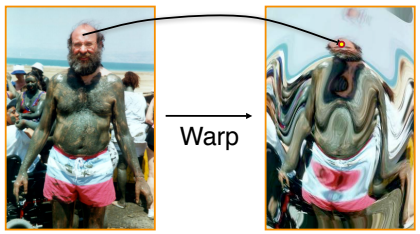
- Move pixels of an image



Source image Destination image

Image Warping

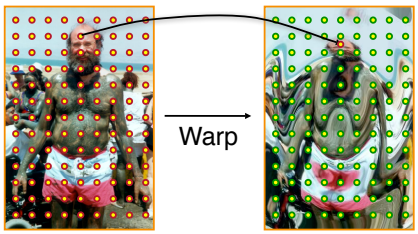
- Issues:
  - How do we specify where every pixel goes? (mapping)



Source image Destination image

Image Warping

- Issues:
  - How do we specify where every pixel goes? (mapping)
  - How do we compute colors at dest pixels? (resampling)



Source image Destination image

Example

- Image scaling:
  - o (x',y') = (sx*x, sy*y)
  - o I(x',y') = ???


Original


1/2X


2X

Image Warping

- Image warping requires resampling of image

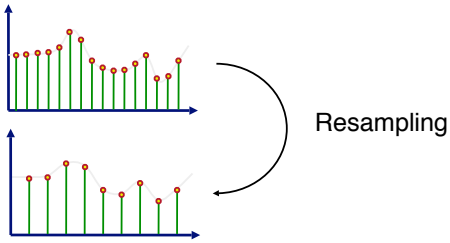


Image Resampling Pipeline

- Resampling requires bandlimiting function in order to avoid aliasing artifacts

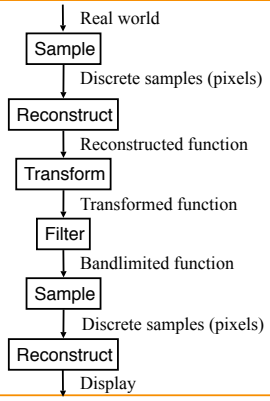


Image Resampling Pipeline

- Resampling requires convolution with low-pass filter in order to reduce aliasing artifacts (in practice)

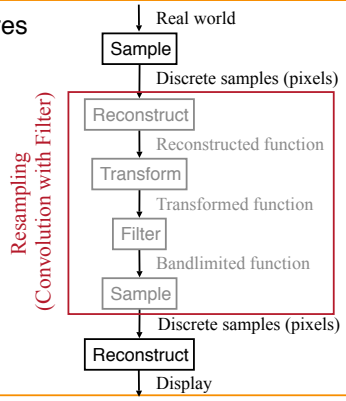


Image Resampling

- Convolution with a triangle filter:

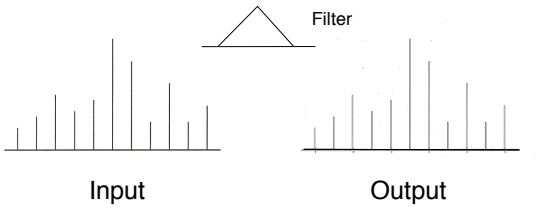


Image Resampling

- Convolution with a triangle filter:

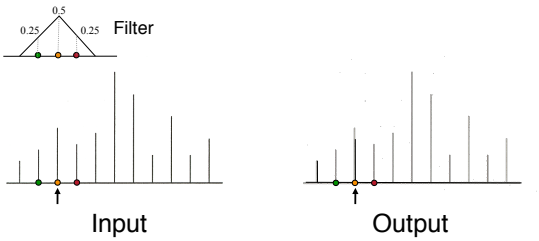
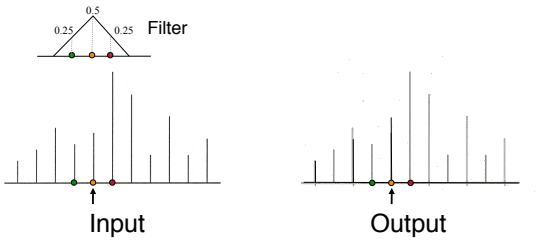


Image Resampling

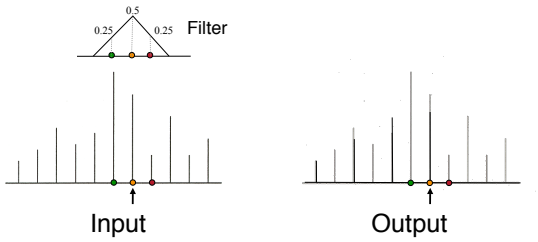
- Convolution with a triangle filter:



13

Image Resampling

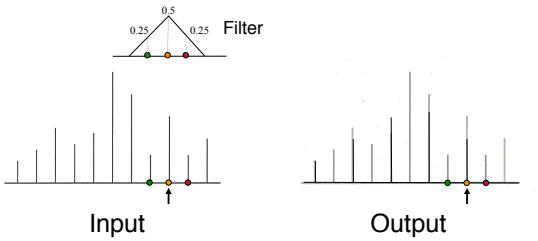
- Convolution with a triangle filter:



14

Image Resampling

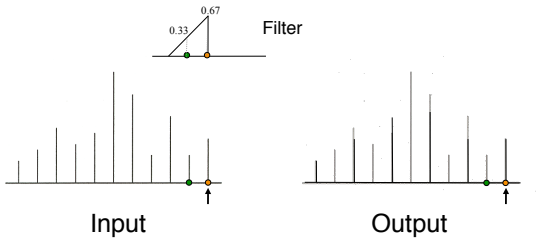
- Convolution with a triangle filter:



15

Image Resampling

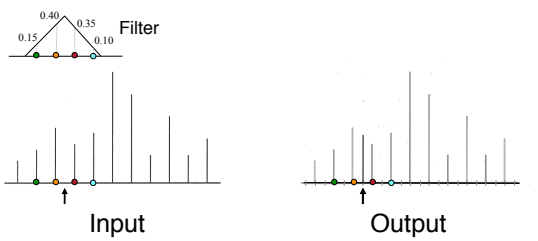
- What if the convolution runs off the end?



16

Image Resampling

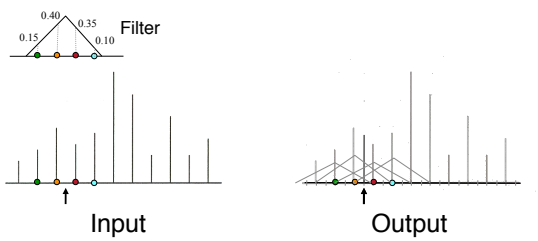
- What if output sample is between input samples?



17

Image Resampling

- What if output sample is between input samples?



18

Image Resampling

- What if scale factor is smaller (e.g., 1/3)?

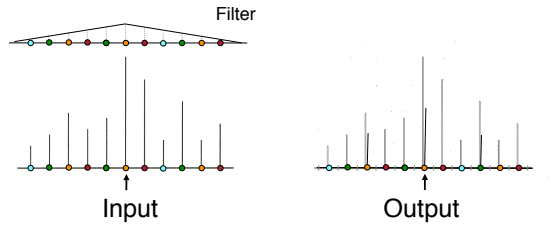


Image Resampling

- What if scale factor is greater than one?

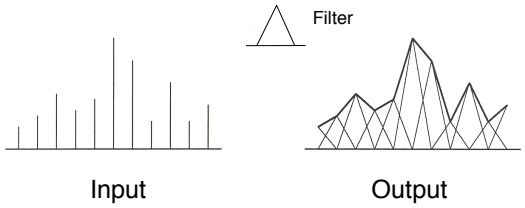


Figure 2.4 Wolberg

Image Resampling

- What is we use a Gaussian filter?

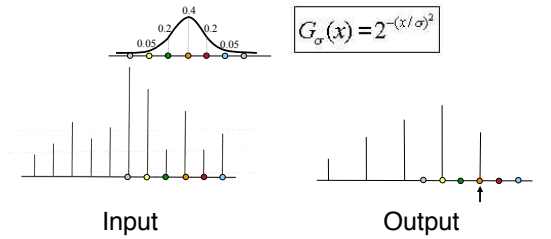


Figure 2.4 Wolberg

Image Resampling

- What is we use a Gaussian filter?

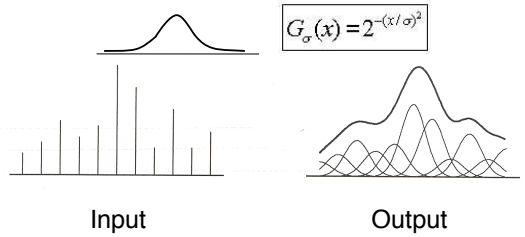
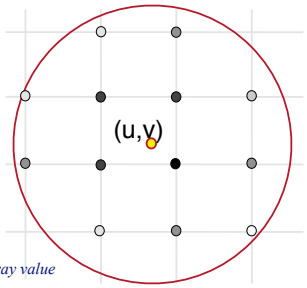


Figure 2.4 Wolberg

Image Resampling

- What if we are resampling a 2D image?
  - Same ideas

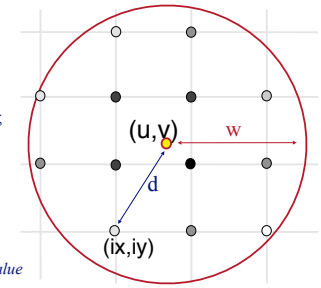


filter values represented by gray value

Image Resampling

- Compute weighted sum of pixel neighborhood
  - Output is weighted average of input, where weights are normalized values of filter kernel (k)

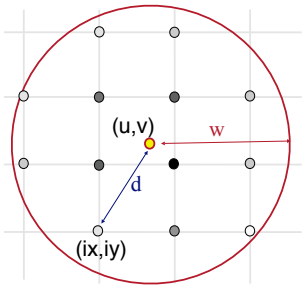
```
dst(ix,iy) = 0;
for (ix = u-w; ix <= u+w; ix++)
  for (iy = v-w; iy <= v+w; iy++)
    d = dist (ix,iy)↔(u,v)
    dst(ix,iy) += k(ix,iy)*src(ix,iy);
```

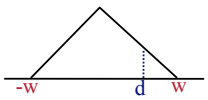


k(ix,iy) represented by gray value

Image Resampling

- For isotropic Triangle and Gaussian filters, k(ix,iy) is function of r and w





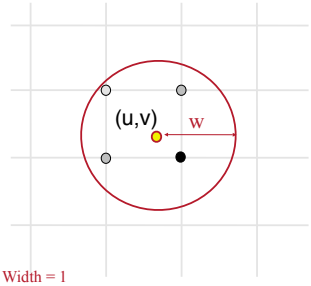
Triangle filter

$k(i,j)=1 - d/w$

Filter Width = 2

Image Resampling

- For isotropic Triangle and Gaussian filters, k(ix,iy) is function of r and w
  - Filter width chosen based on scale factor (or blur)



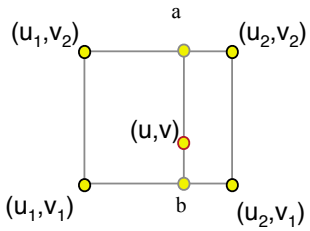
Triangle filter

Width of filter affects blurriness

Filter Width = 1

Triangle Filtering (with width <= 1)

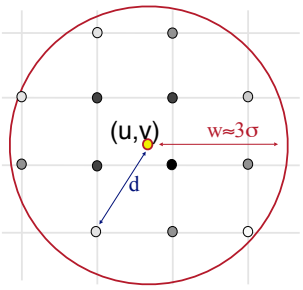
- Bilinearly interpolate four closest pixels
  - a = linear interpolation of src(u_1,v_2) and src(u_2,v_2)
  - b = linear interpolation of src(u_1,v_1) and src(u_2,v_1)
  - dst(x,y) = linear interpolation of "a" and "b"



Filter Width = 1

Gaussian Filtering

- Kernel is Gaussian function



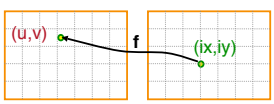
Gaussian Function

$$G_\sigma(d) = 2^{-(d/\sigma)^2}$$

Image Scale

- Possible scale implementation:

```
Scale(src, dst, sx, sy) {
  w ≈ max(1/sx,1/sy);
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix / sx;
      float v = iy / sy;
      dst(ix,iy) = Resample(src, u, v, k, w);
    }
  }
}
```



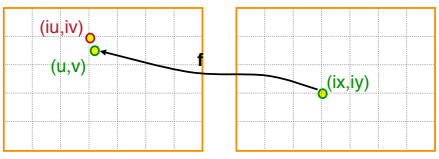
Source image Destination image

Point Sampling

- Possible (poor) resampling implementation:

```
float Resample(src, u, v, w) {
  int iu = round(u);
  int iv = round(v);
  return src(iu,iv);
}
```

This method is simple, but it causes aliasing

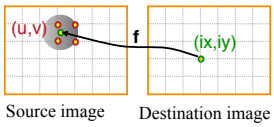


Source image Destination image

Triangle and Gaussian Sampling

- Better resampling implementation:

```
float Resample(src, u, v, k, w)
{
  float dst = 0;
  float ksum = 0;
  int ulo = u - w; etc.
  for (int iu = ulo; iu < uhi; iu++) {
    for (int iv = vlo; iv < vhi; iv++) {
      dst += k(u,v,iu,iv,w) * src(u,v)
      ksum += k(u,v,iu,iv,w);
    }
  }
  return dst / ksum;
}
```



Sampling Method Comparison

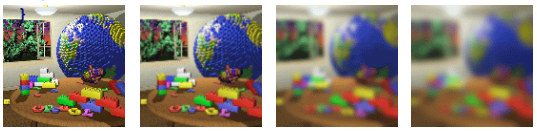
- Trade-offs
  - Aliasing versus blurring
  - Computation speed

Point Triangle Gaussian

Image Blur

- Possible blur implementation:

```
Blur(src, dst, sigma) {
  w ≈ 3*sigma;
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix;
      float v = iy;
      dst(ix,iy) = Resample(src, u, v, k, w);
    }
  }
}
```



Increasing sigma

Image Rotation

- Possible rotation implementation:

```
Rotate(src, dst, Θ) {
  w ≈ 1
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix*cos(-Θ) - iy*sin(-Θ);
      float v = ix*sin(-Θ) + iy*cos(-Θ);
      dst(ix,iy) = Resample(src,u,v,w);
    }
  }
}
```

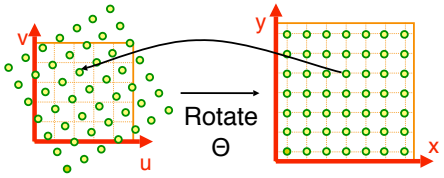


Image Warping (in General)

- Possible warp implementation (reverse mapping):

```
Warp(src, dst) {
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float w ≈ 1 / scale(ix, iy);
      float u = f_x^-1(ix,iy);
      float v = f_y^-1(ix,iy);
      dst(ix,iy) = Resample(src,u,v,w);
    }
  }
}
```

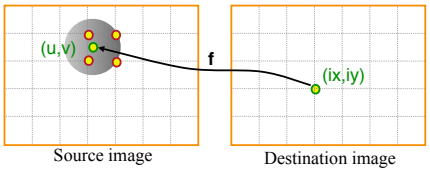


Image Warping (in General)

- Alternative implementation (forward mapping):

```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = f_x(iu,iv);
      float y = f_y(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv), x, y, w);   ← weighting ???
    }
  }
}
```

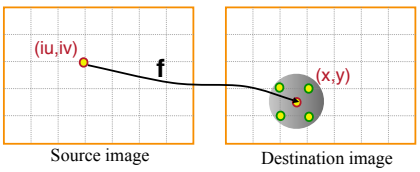


Image Warping (in General)

- Alternative implementation (forward mapping):

```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = f_x(iu,iv);
      float y = f_y(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv), x, y, w);   ← weighting ???
    }
  }
}
```

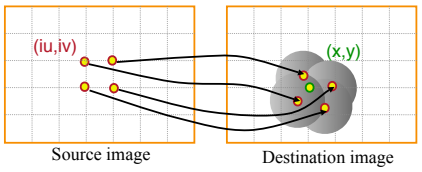


Image Warping (in General)

- Alternative implementation (forward mapping):

```
for (int iu = 0; iu < umax; iu++) {
  for (int iv = 0; iv < vmax; iv++) {
    float x = f_x(iu,iv);
    float y = f_y(iu,iv);
    for (int ix = xlo; ix <= xhi; ix++) {
      for (int iy = ylo; iy <= yhi; iy++) {
        dst(ix,iy) += k(x,y,ix,iy,w) * src(iu,iv);
        ksum(ix,iy) += k(x,y,ix,iy,w);
      }
    }
  }
}
for (ix = 0; ix < xmax; ix++)
  for (iy = 0; iy < ymax; iy++)
    dst(ix,iy) /= ksum(ix,iy)
```

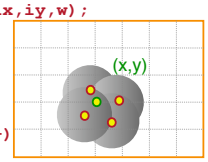
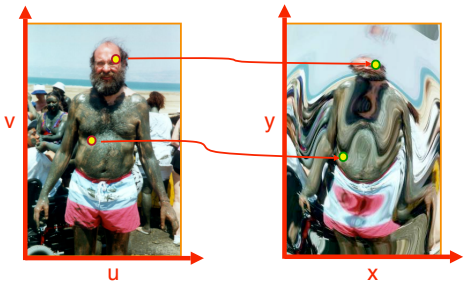


Image Processing

- Pixel operations
  - Add random noise
  - Add luminance
  - Add contrast
  - Add saturation
- Filtering
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median
- Quantization
  - Uniform Quantization
  - Floyd-Steinberg dither
- Warping
  - Scale
  - Rotate
  - Warp
- Combining
  - Composite
  - Morph

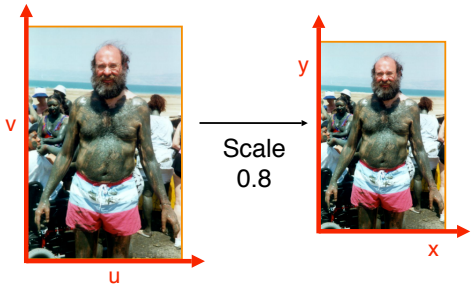
Mapping

- Define transformation
  - Describe the destination (x,y) for every source (u,v) (actually vice-versa, if reverse mapping)



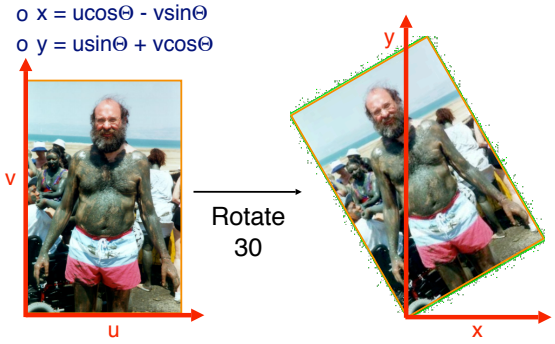
Example Mappings

- Scale by *factor*:
  - x = *factor* * u
  - y = *factor* * v



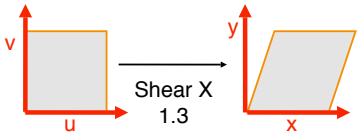
Example Mappings

- Rotate by Θ degrees:
  - x = ucosΘ - vsinΘ
  - y = usinΘ + vcosΘ

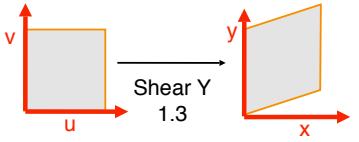


Example Mappings

- Shear in X by *factor:*
  - x = u + *factor* * v
  - y = v



- Shear in Y by *factor:*
  - x = u
  - y = v + *factor* * u



Other Parametric Mappings

- Any function of u and v:
  - x = $f_x(u,v)$
  - y = $f_y(u,v)$

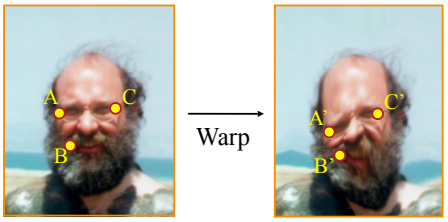
Fish-eye

"Swirl"

"Rain"

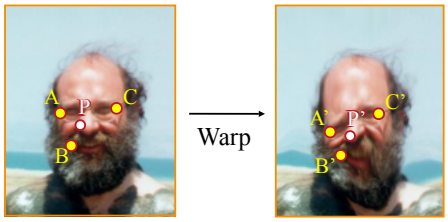
Point Correspondence Mappings

- Mappings implied by correspondences:
  - A ↔ A'
  - B ↔ B'
  - C ↔ C'



Point Correspondence Mappings

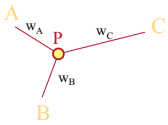
- How compute P' from:
  - A ↔ A'
  - B ↔ B'
  - C ↔ C'
  - P

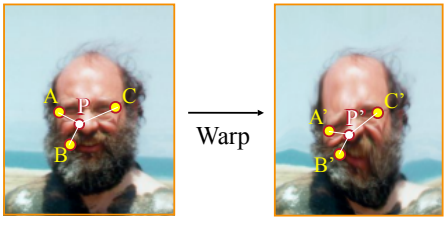


Point Correspondence Mappings

- How compute P from:
  - A ↔ A'
  - B ↔ B'
  - C ↔ C'
  - P'

$P' = w_A A + w_B B + w_C C$



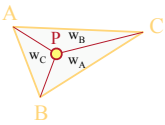


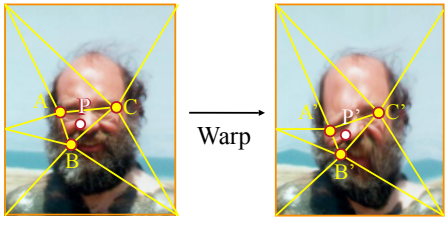
Point Correspondence Mappings

- How compute P' from:
  - A ↔ A'
  - B ↔ B'
  - C ↔ C'
  - P

$P = w_A' A + w_B' B + w_C' C$

Barycentric coordinates



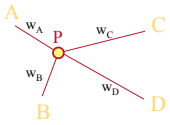


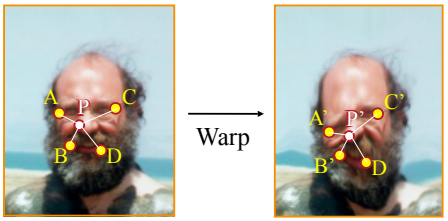
Point Correspondence Mappings

- How compute P' from:
  - o $X_i \leftrightarrow X_i'$
  - o P

$$P = \sum_i w_i' X_i$$

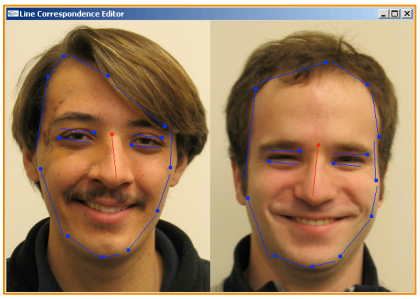
Radial Basis Functions
Thin-Plate Splines





Line Correspondence Mappings

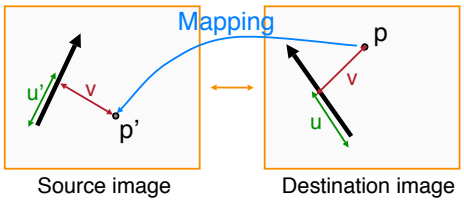
- Beier & Neeley use pairs of lines to specify warp



Beier & Neeley
SIGGRAPH 92

Line Correspondence Mappings

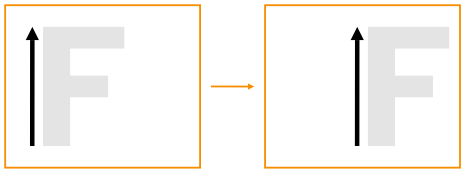
- Beier & Neeley use pairs of lines to specify warp
  - o Given p in dst image, where is p' in source image?



u is a fraction
v is a length (in pixels)

Warping with One Line Pair

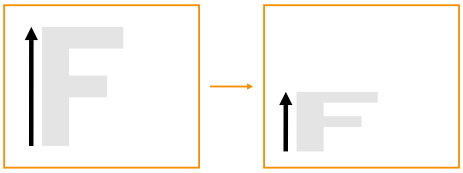
- What happens to the "F"?



Translation!

Warping with One Line Pair

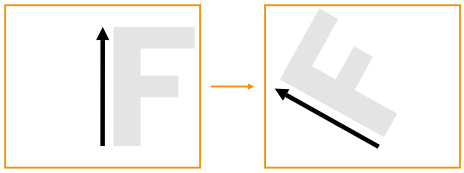
- What happens to the "F"?



Scale!

Warping with One Line Pair

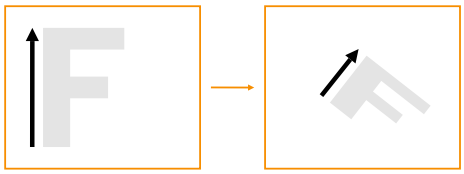
- What happens to the "F"?



Rotation!

Warping with One Line Pair

- What happens to the “F”?

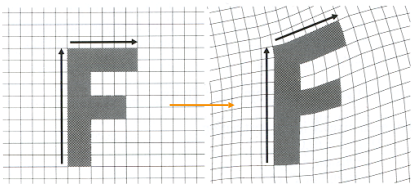


In general, similarity transformations

What types of transformations can’t be specified?

Warping with Multiple Line Pairs

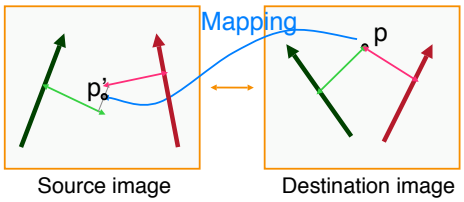
- Use weighted combination of points defined by each pair of corresponding lines



Beier & Neeley, Figure 4

Warping with Multiple Line Pairs

- Use weighted combination of points defined by each pair of corresponding lines



p’ is a weighted average

Weighting Effect of Each Line Pair

- To weight the contribution of each line pair, Beier & Neeley use:

$$weight[i] = \left(\frac{length[i]^p}{a + dist[i]} \right)^b$$

Where:

- *length[i]* is the length of L[i]
- *dist[i]* is the distance from X to L[i]
- *a, b, p* are constants that control the warp

Putting It All Together

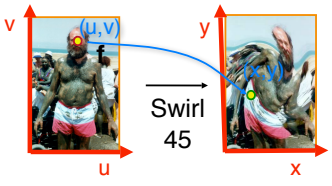
- Warping with correspondences

```
Warp(src, dst, correspondences) {
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float w ≈ 1 / scale(ix, iy);
      float u = f_x^-1(ix,iy,correspondences);
      float v = f_y^-1(ix,iy,correspondences);
      dst(ix,iy) = Resample(src,u,v,w);
    }
  }
}
```


Putting It All Together

- Other fun warps:

```
Swirl(src, dst, Θ) {
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = rot(dist(ix,xcenter)*Θ);
      float v = rot(dist(iy,ycenter)*Θ);
      dst(ix,iy) = Resample(src,u,v,1);
    }
  }
}
```



More COS426 Examples

Randy Carnevale

Sid Kapur

Wei Xiang

Paul Nelson

Summary

- Resampling
  - Triangle filter
  - Gaussian filter
- Mapping
  - Parametric
  - Correspondences
- Image processing
  - Reverse mapping
  - Forward mapping