



Image Warping, Compositing & Morphing

Tom Funkhouser
Princeton University
COS 426, Spring 2006



Image Processing

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



Image Processing

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



Image Warping

- Move pixels of image
 - Mapping
 - Resampling



Source image

→ Warp



Destination image



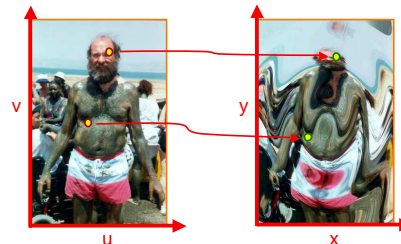
Image Warping

- Mapping
 - Forward
 - Reverse
- Resampling
 - Point sampling
 - Triangle filter
 - Gaussian filter



Mapping

- Define transformation
 - Describe the destination (x,y) for every location (u,v) in the source (or vice-versa, if invertible)

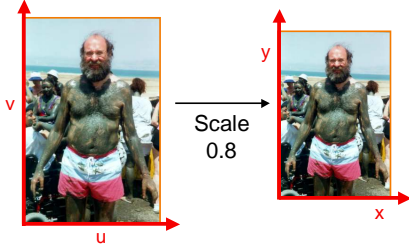


Example Mappings



- Scale by factor:

- $x = factor * u$
- $y = factor * v$

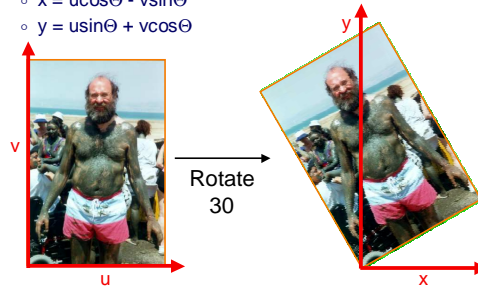


Example Mappings



- Rotate by Θ degrees:

- $x = u \cos \Theta - v \sin \Theta$
- $y = u \sin \Theta + v \cos \Theta$

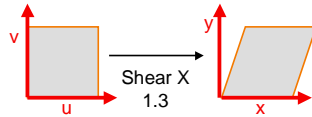


Example Mappings



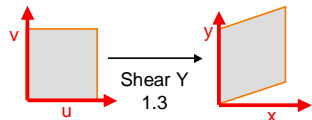
- Shear in X by factor:

- $x = u + factor * v$
- $y = v$



- Shear in Y by factor:

- $x = u$
- $y = v + factor * u$



Other Mappings



- Any function of u and v:

- $x = f_x(u,v)$
- $y = f_y(u,v)$



Fish-eye



"Swirl"



"Rain"

Image Warping Implementation I



- Forward mapping

- Iterate over source image

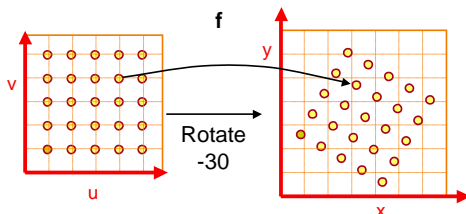
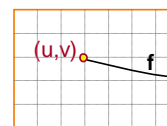


Image Warping Implementation I

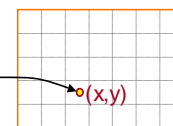


- Forward mapping:

```
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);
        ???
    }
}
```



Source image



Destination image

Image Warping Implementation I

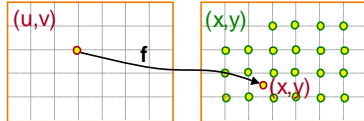


- Forward mapping:

```

for (int iu = 0; iu < umax; iu++) {
  for (int iv = 0; iv < vmax; iv++) {
    float x = fx(iu,iv);
    float y = fy(iu,iv);
    for (int ix = 0; ix <= xmax; ix++) {
      for (int iy = 0; iy <= ymax; iy++) {
        dst(ix,iy) += ???
      }
    }
  }
}

```



Source image

Destination image

Image Warping Implementation I

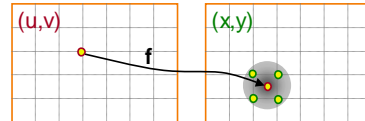


- Forward mapping:

```

for (int iu = 0; iu < umax; iu++) {
  for (int iv = 0; iv < vmax; iv++) {
    float x = fx(iu,iv);
    float y = fy(iu,iv);
    for (int ix = xlo; ix <= xhi; ix++) {
      for (int iy = ylo; iy <= yhi; iy++) {
        dst(ix,iy) += k(x,y,ix,iy) * src(iu,iv);
      }
    }
  }
}

```



Source image

Destination image

Image Warping Implementation I

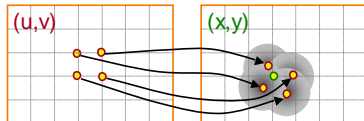


- Forward mapping:

```

for (int iu = 0; iu < umax; iu++) {
  for (int iv = 0; iv < vmax; iv++) {
    float x = fx(iu,iv);
    float y = fy(iu,iv);
    for (int ix = xlo; ix <= xhi; ix++) {
      for (int iy = ylo; iy <= yhi; iy++) {
        dst(ix,iy) += k(x,y,ix,iy) * src(iu,iv);
      }
    }
  }
}

```



Source image

Destination image

Image Warping Implementation I

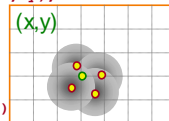


- Forward mapping:

```

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

```



Destination image

Image Warping Implementation II



- Reverse mapping
 - Iterate over destination image

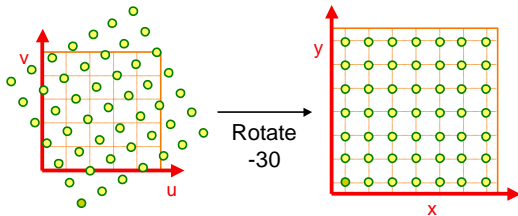


Image Warping Implementation II



- Reverse mapping
 - Iterate over destination image

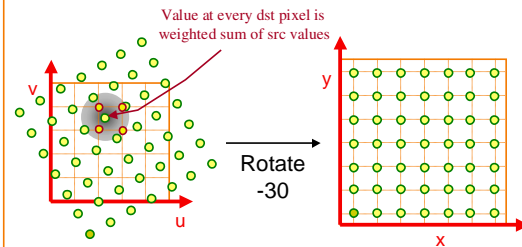


Image Warping Implementation II



- Reverse mapping:

```
for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
        float u =  $f_x^{-1}(ix, iy)$ ;
        float v =  $f_y^{-1}(ix, iy)$ ;
        ???
    }
}
```

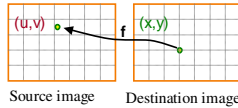


Image Warping Implementation II



- Reverse mapping:

```
for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
        float u =  $f_x^{-1}(ix, iy)$ ;
        float v =  $f_y^{-1}(ix, iy)$ ;
        dst(ix, iy) = 0;
        for (int iu = u_lo; iu < u_hi; iu++) {
            for (int iv = v_hi; iv < v_hi; iv++) {
                dst(ix, iy) += ???
            }
        }
    }
}
```

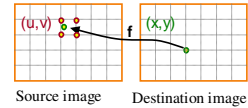


Image Warping Implementation II



- Reverse mapping:

```
for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
        float u =  $f_x^{-1}(ix, iy)$ ;
        float v =  $f_y^{-1}(ix, iy)$ ;
        dst(ix, iy) = 0;
        for (int iu = u_lo; iu < u_hi; iu++) {
            for (int iv = v_hi; iv < v_hi; iv++) {
                dst(ix, iy) += k(u, v, iu, iv) * src(u, v);
            }
        }
    }
}
```

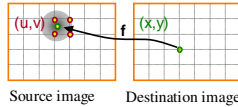


Image Warping Implementation II



- Reverse mapping:

```
for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
        float u =  $f_x^{-1}(ix, iy)$ ;
        float v =  $f_y^{-1}(ix, iy)$ ;
        dst(ix, iy) = 0;
        float ksum = 0;
        for (int iu = u_lo; iu < u_hi; iu++) {
            for (int iv = v_hi; iv < v_hi; iv++) {
                dst(ix, iy) += k(u, v, iu, iv) * src(u, v);
                ksum += k(u, v, iu, iv);
            }
        }
        dst(ix, iy) /= ksum;
    }
}
```

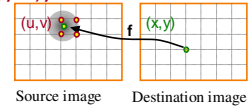


Image Warping Implementation II



- Reverse mapping:

```
for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
        float u =  $f_x^{-1}(ix, iy)$ ;
        float v =  $f_y^{-1}(ix, iy)$ ;
        dst(ix, iy) = 0;
        float ksum = 0;
        for (int iu = u_lo; iu < u_hi; iu++) {
            for (int iv = v_hi; iv < v_hi; iv++) {
                dst(ix, iy) += k(u, v, iu, iv) * src(u, v);
                ksum += k(u, v, iu, iv);
            }
        }
        dst(ix, iy) /= ksum;
    }
}
```

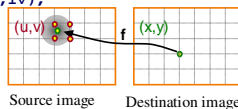


Image Warping Implementation II



- Reverse mapping:

```
float Resample(src, u, v, w)
{
    float dst = 0;
    float ksum = 0;
    int u_lo = u - w;
    for (int iu = u_lo; iu < u_hi; iu++) {
        for (int iv = v_hi; iv < v_hi; iv++) {
            dst += k(u, v, iu, iv) * src(u, v);
            ksum += k(u, v, iu, iv);
        }
    }
    return dst / ksum;
}
```

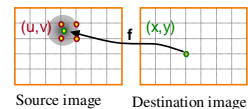


Image Warping Implementation II



- Reverse mapping:

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

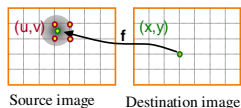


Image Warping



- Mapping
 - Forward
 - Reverse
- » Resampling
 - Point sampling
 - Triangle filter
 - Gaussian filter

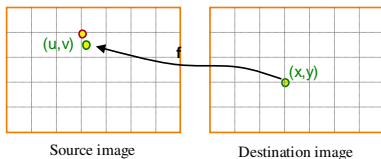
Point Sampling



- Take value at closest pixel:

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

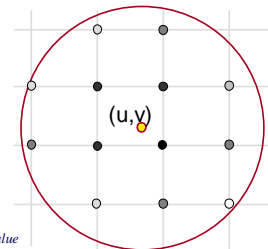
This method is simple, but it causes aliasing



Filtering



- Compute weighted sum of pixel neighborhood
 - Weights are normalized values of kernel function
 - Equivalent to convolution at samples

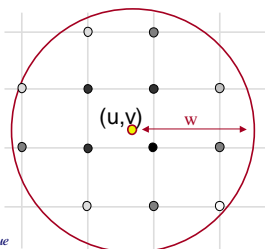


Filtering



- Compute weighted sum of pixel neighborhood
 - Weights are normalized values of kernel function
 - Equivalent to convolution at samples

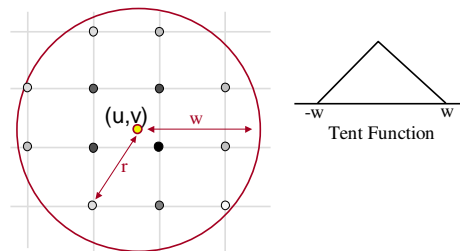
```
s = 0;
for (i = -w; i <= w; i++)
  for (j = -w; j <= w; j++)
    s += k(i,j)*I(u+i, v+j);
```



Triangle Filtering



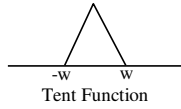
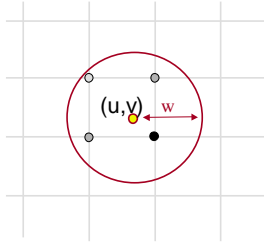
- Kernel is triangle function



Triangle Filtering



- Kernel is triangle function



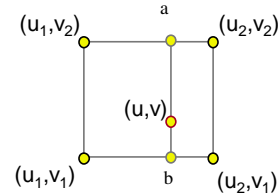
Width of filter affects blurriness

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"

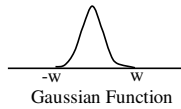
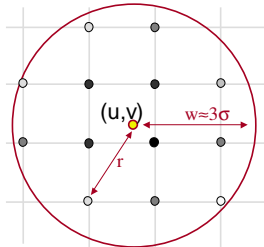


Filter Width = 1

Gaussian Filtering



- Kernel is Gaussian function



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

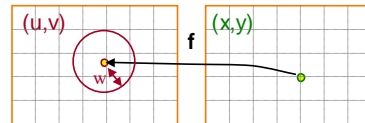
Filter Width = 2

Filter Size



- How choose w?

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



Source image

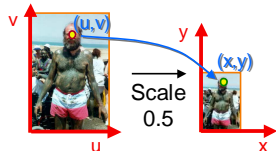
Destination image

Filter Size



- Scale (src, dst, sx, sy):

```
float w = max(1.0/sx, 1.0/sy);
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = x / sx;
    float v = y / sy;
    dst(x, y) = Resample(src, u, v, w);
  }
}
```



Filtering Methods Comparison



- Trade-offs
 - Aliasing versus blurring
 - Computation speed



Point

Bilinear

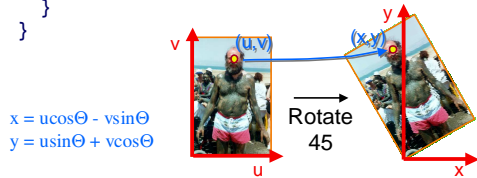
Gaussian

Example: Rotate



- Rotate (src, dst, theta):

```
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = x*cos(-θ) - y*sin(-θ);
    float v = x*sin(-θ) + y*cos(-θ);
    dst(x,y) = Resample(src,u,v,w);
  }
}
```

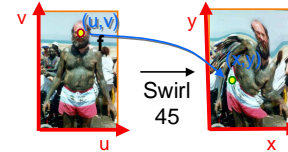


Example: Fun



- Swirl (src, dst, theta):

```
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = rot(dist(x,xcenter)*theta);
    float v = rot(dist(y,ycenter)*theta);
    dst(x,y) = Resample(src,u,v,w);
  }
}
```



COS426 Examples



Randy Carnevale



Jonathan Heinberg



Sid Kapur



Philip Wei

Image Processing



- Quantization
 - Uniform Quantization
 - Random dither
 - Ordered dither
 - Floyd-Steinberg dither
- Filtering
 - Blur
 - Detect edges
- Warping
 - Scale
 - Rotate
 - Warp
- Combining
 - Composite
 - Morph

Image Compositing



- Combine images
 - Separate image into "elements"
 - Generate independently
 - Composite together
- Applications
 - Cel animation
 - Chroma-keying
 - Blue-screen matting



Blue-Screen Matting



- Composite foreground and background images
 - Create background image
 - Create foreground image with blue background
 - Insert non-blue foreground pixels into background

Problem: no partial coverage!

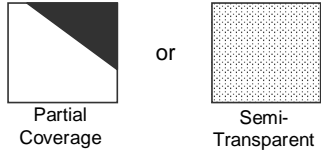


Alpha Channel



- Encodes pixel coverage information
 - $\alpha = 0$: no coverage (or transparent)
 - $\alpha = 1$: full coverage (or opaque)
 - $0 < \alpha < 1$: partial coverage (or semi-transparent)

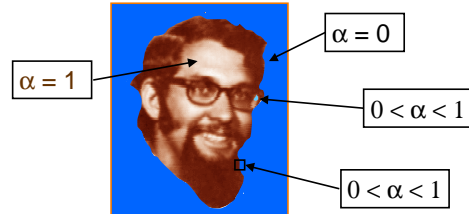
- Example: $\alpha = 0.3$



Compositing with Alpha



Controls the linear interpolation of foreground and background pixels when elements are composited.



Semi-Transparent Objects



- Suppose we put A over B over background G



- How much of B is blocked by A?

$$\alpha_A$$
- How much of B shows through A?

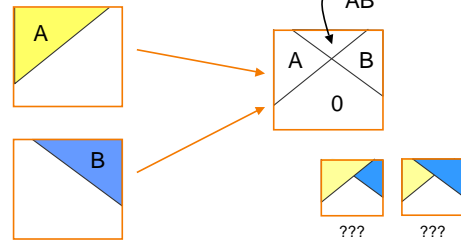
$$(1 - \alpha_A)$$
- How much of G shows through both A and B?

$$(1 - \alpha_A)(1 - \alpha_B)$$

Opaque Objects



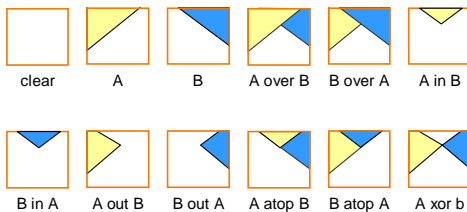
- How do we combine 2 partially covered pixels?
 - 3 possible colors (0, A, B)
 - 4 regions (0, A, B, AB)



Composition Algebra



- 12 reasonable combinations

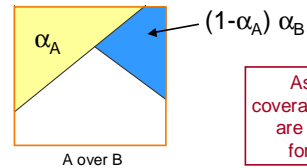


Porter & Duff '84

Example: C = A Over B

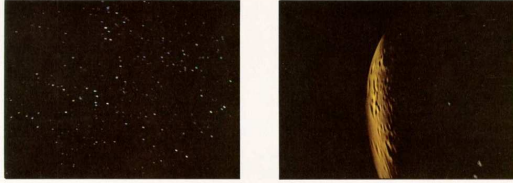


- Consider the areas covered:
 - $C = \alpha_A A + (1 - \alpha_A) \alpha_B B$
 - $\alpha = \alpha_A + (1 - \alpha_A) \alpha_B$



Assumption: coverages of A and B are uncorrelated for each pixel

Image Composition Example

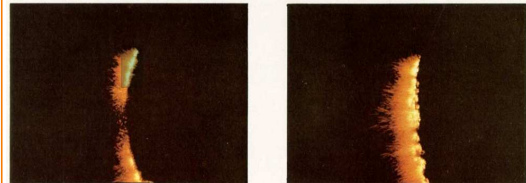


Stars

Planet

[Porter&Duff Computer Graphics 18:3 1984]

Image Composition Example

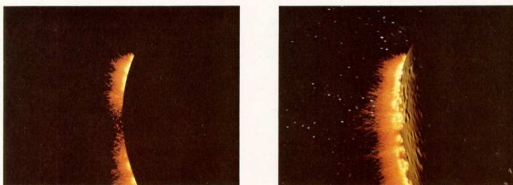


BFire

FFire

[Porter&Duff Computer Graphics 18:3 1984]

Image Composition Example



BFire out Planet

Composite

[Porter&Duff Computer Graphics 18:3 1984]

Image Composition Example

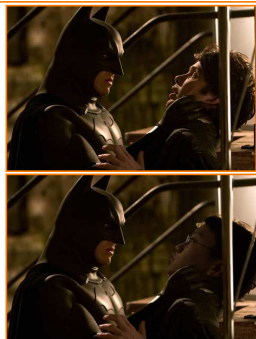


Jurassic Park

COS426 Examples



Darin Sleiter



Kenrick Kin

Even CG folks Can Win an Oscar



Smith Duff Catmull Porter

Image Processing



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

Image Morphing



- Animate transition between two images

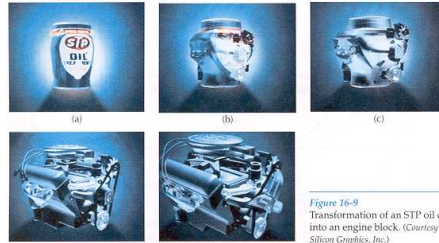


Figure 16-9 Transformation of an STP oil can into an engine block. (Courtesy of Silicon Graphics, Inc.)

H&B Figure 16.9

Cross-Dissolving



- Blend images with "over" operator
 - alpha of bottom image is 1.0
 - alpha of top image varies from 0.0 to 1.0

$$\text{blend}(i,j) = (1-t) \text{src}(i,j) + t \text{dst}(i,j) \quad (0 \leq t \leq 1)$$

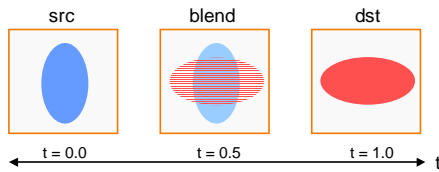


Image Morphing



- Combines warping and cross-dissolving

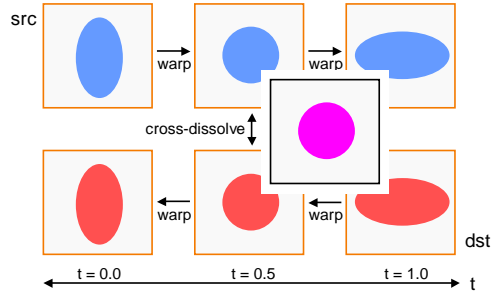
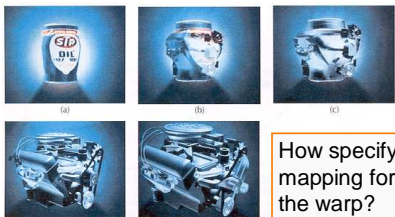


Image Morphing



- The warping step is the hard one
 - Aim to align features in images



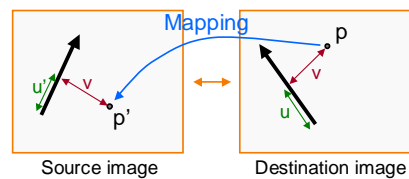
How specify mapping for the warp?

H&B Figure 16.9

Feature-Based Warping



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



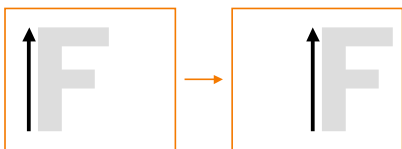
u is a fraction
 v is a length (in pixels)

Beier & Neeley
SIGGRAPH 92

Warping with One Line Pair



- What happens to the "F"?

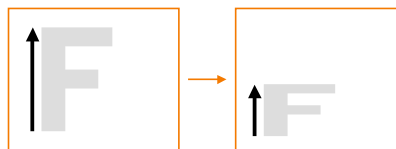


Translation!

Warping with One Line Pair



- What happens to the "F"?

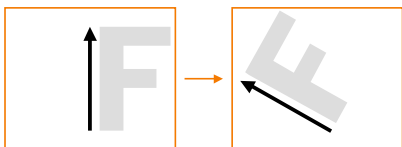


Scale!

Warping with One Line Pair



- What happens to the "F"?

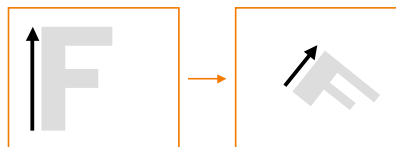


Rotation!

Warping with One Line Pair



- What happens to the "F"?



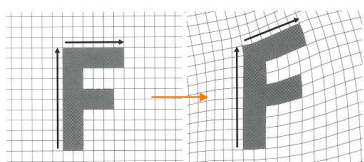
In general, similarity transformations

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

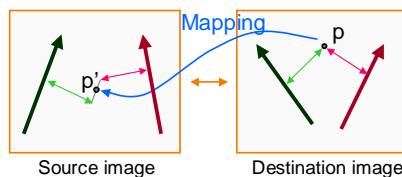


Beier & Neeley, Figure 4

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

Warping Pseudocode

```

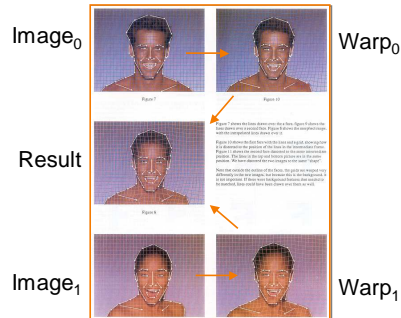
WarpImage(Image, L[...], L[...])
begin
  foreach destination pixel p do
    psum = (0,0)
    wsum = 0
    foreach line L[i] in destination do
      p'[i] = p transformed by (L[i], L'[i])
      psum = psum + p'[i] * weight[i]
      wsum += weight[i]
    end
    p' = psum / wsum
    Result(p) = Image(p')
  end
end
  
```

Morphing Pseudocode

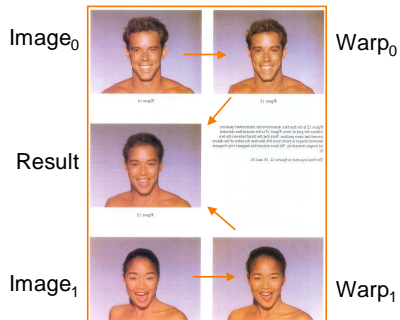
```

GenerateAnimation(Image0, L0[...], Image1, L1[...])
begin
  foreach intermediate frame time t do
    for i = 1 to number of line pairs do
      L[i] = line t-th of the way from L0[i] to L1[i]
    end
    Warp0 = WarpImage(Image0, L0, L)
    Warp1 = WarpImage(Image1, L1, L)
    foreach pixel p in FinalImage do
      Result(p) = (1-t) Warp0 + t Warp1
    end
  end
end
  
```

Beier & Neeley Example



Beier & Neeley Example



COS426 Examples



Summary



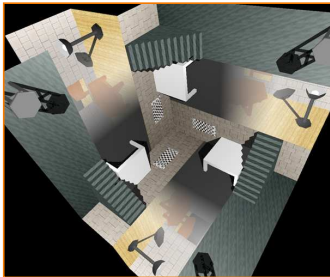
- Image warping
 - Mapping
 - Resampling
- Image compositing
 - Alpha channel
 - Porter-Duff compositing algebra
- Image morphing
 - Specifying correspondences
 - Warping
 - Compositing

Image Processing



- Quantization
 - Uniform Quantization
 - Random dither
 - Ordered dither
 - Floyd-Steinberg dither
- Filtering
 - Blur
 - Detect edges
- Warping
 - Scale
 - Rotate
 - Warp
- Combining
 - Composite
 - Morph

Next Time: 3D Rendering



Misha Kazhdan