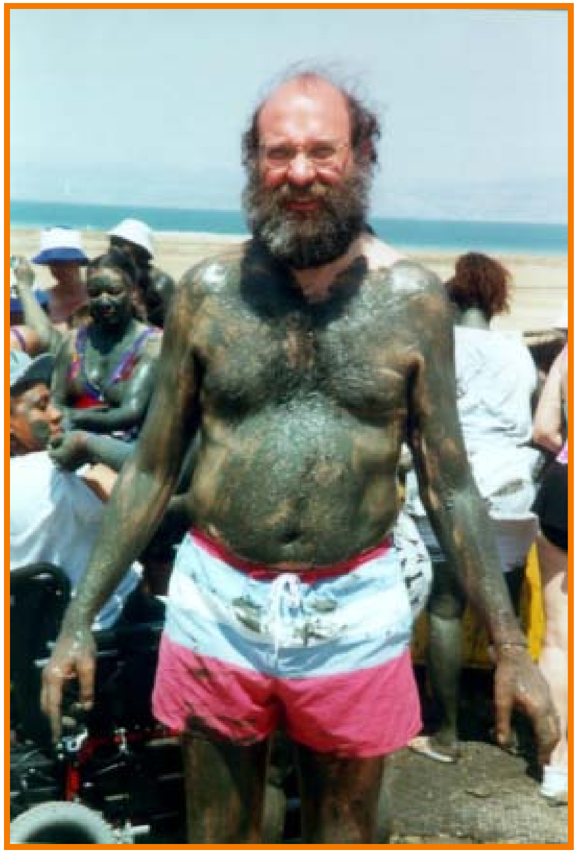
Image Warping

COS 426

Image Warping

- Move pixels of an image



Source image

Warp

Destination image

Image Warping

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

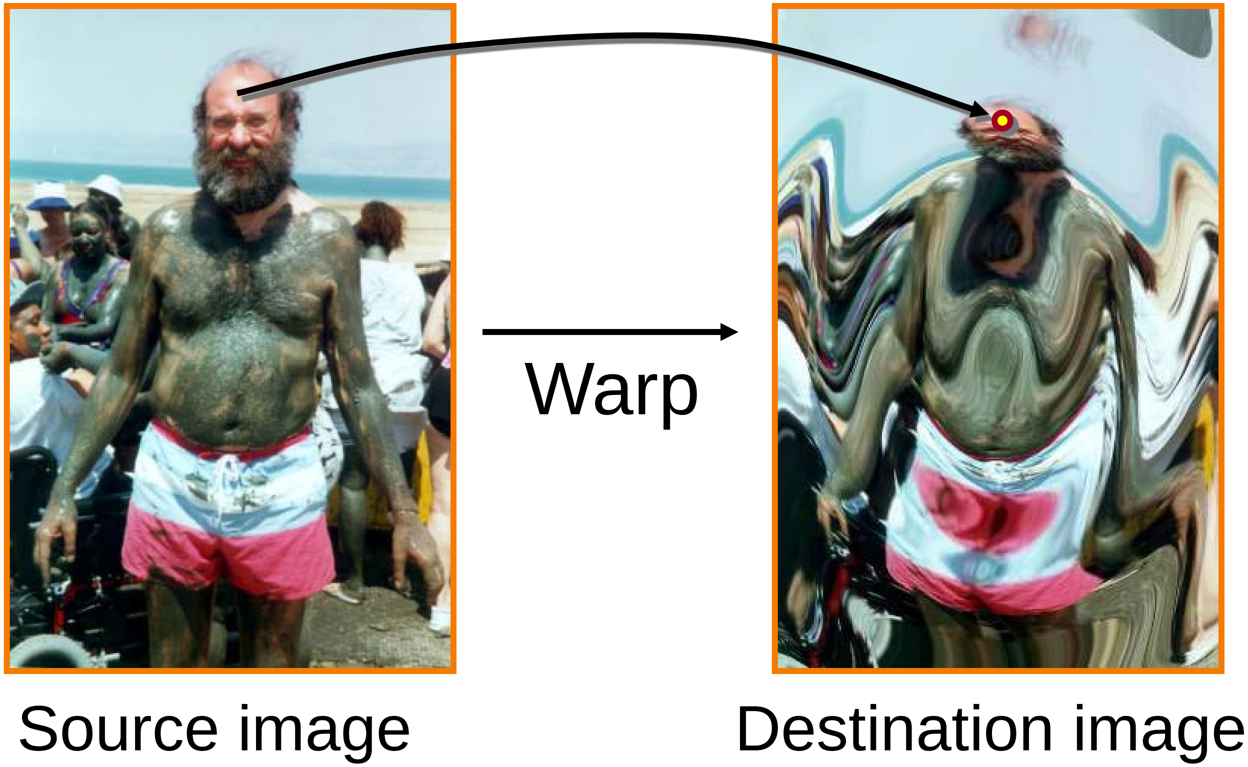


Image Warping

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

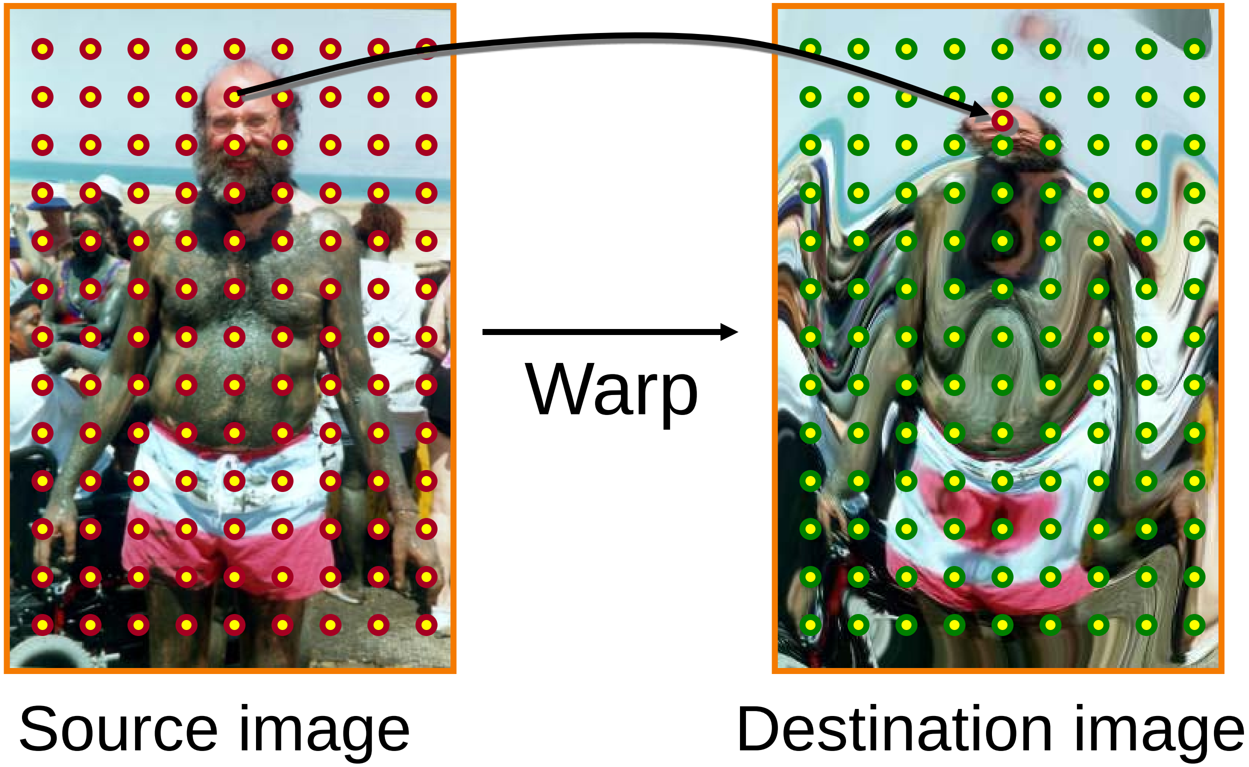
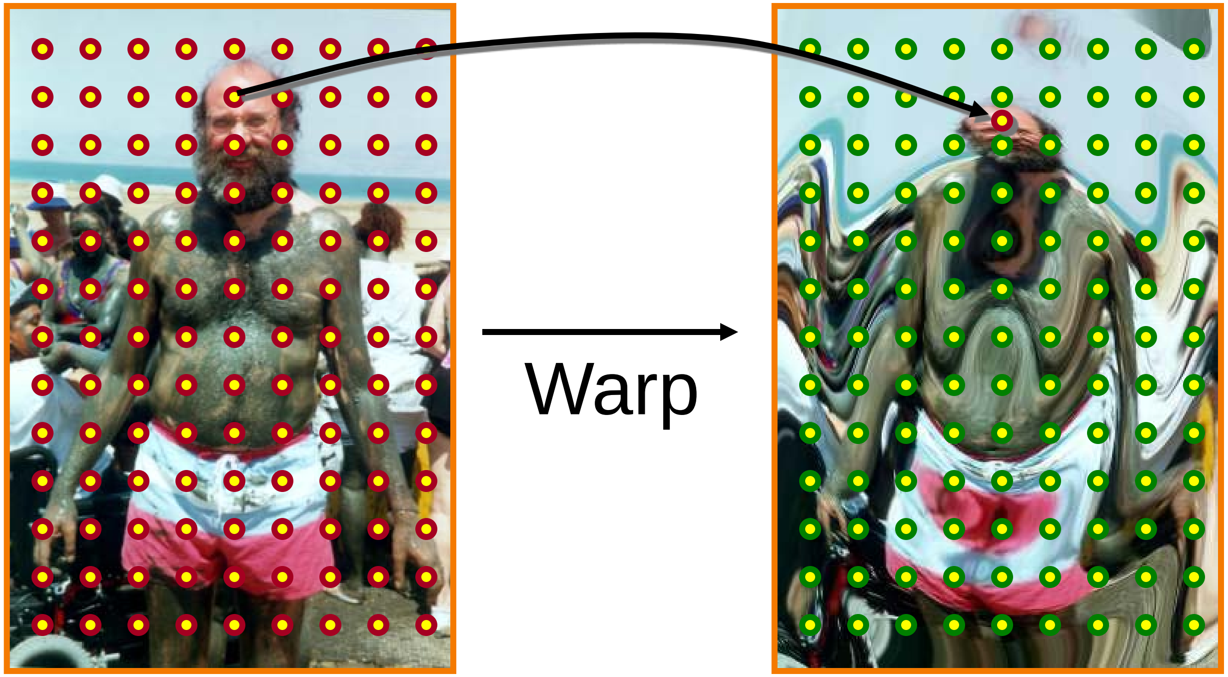


Image Warping

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

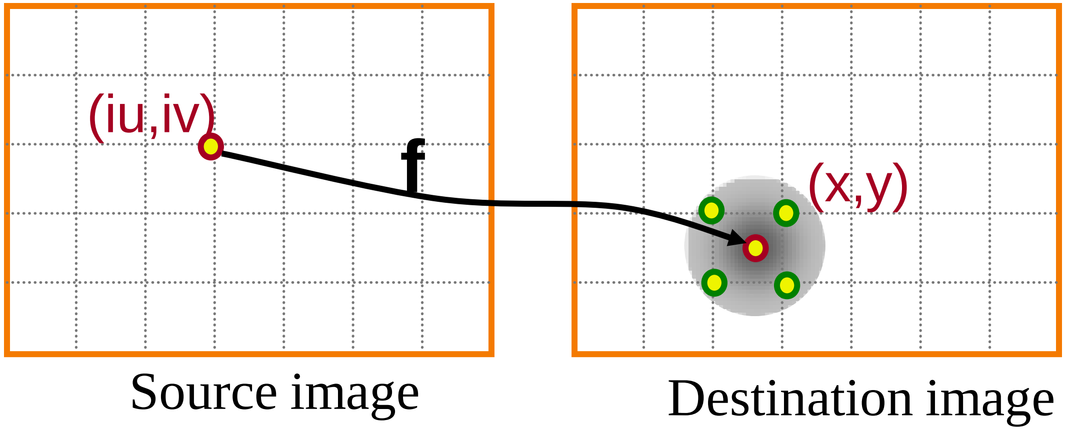


Source image

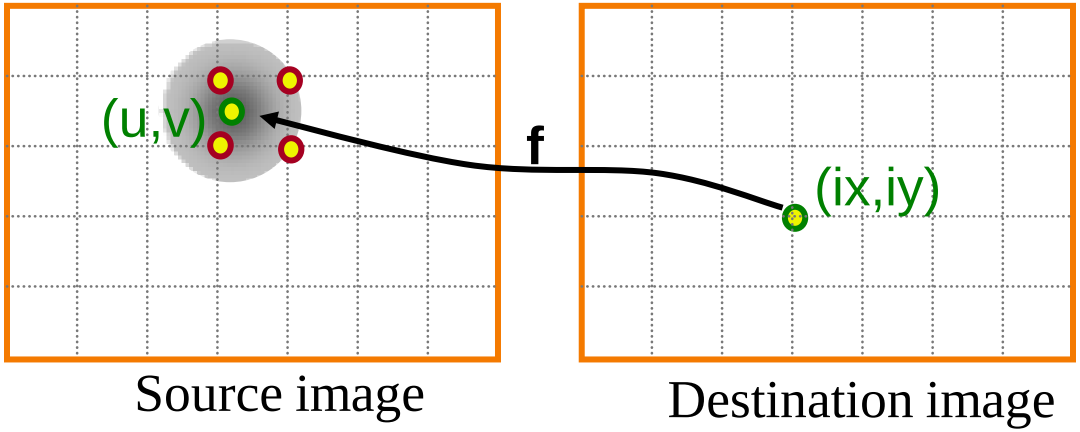
Destination image

Two Options

- Forward mapping

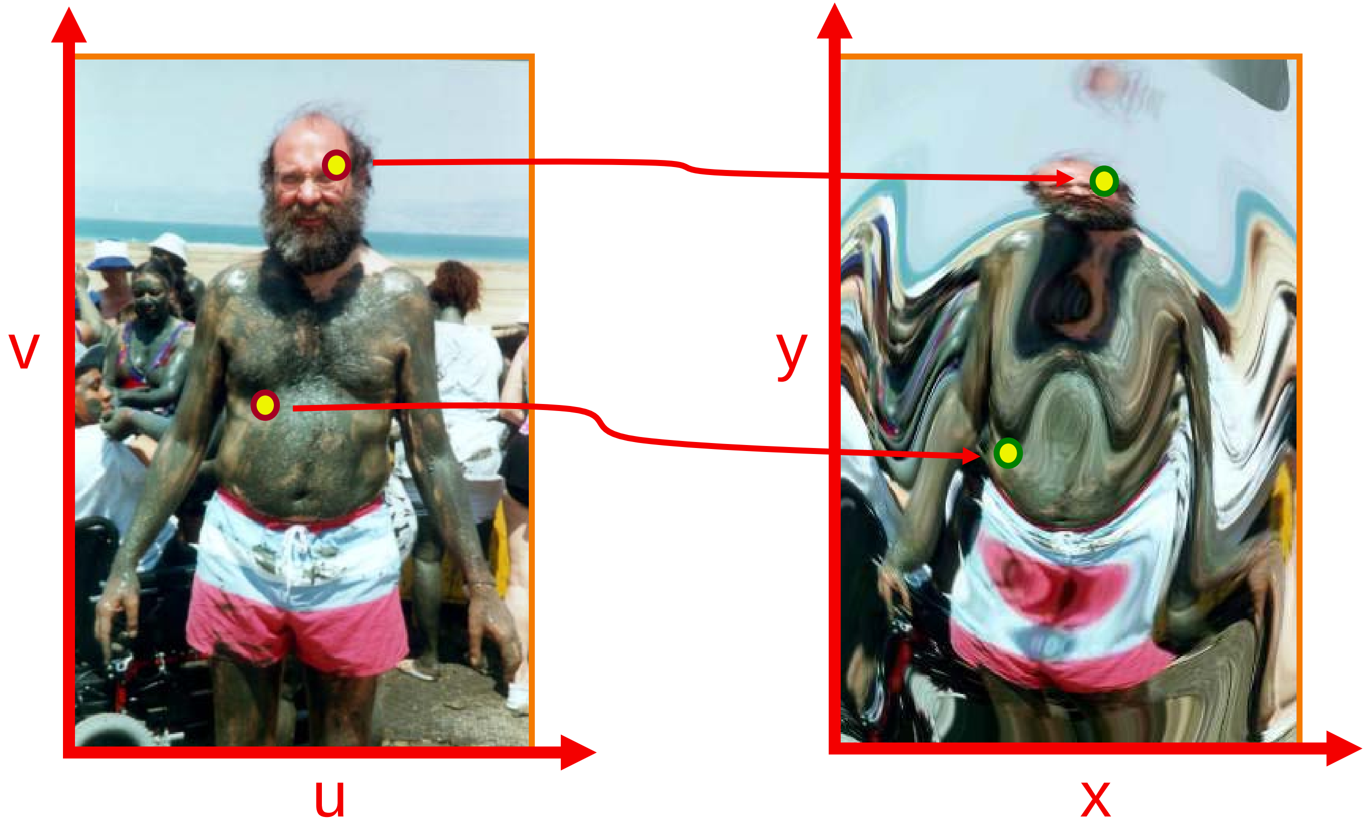


- Reverse mapping



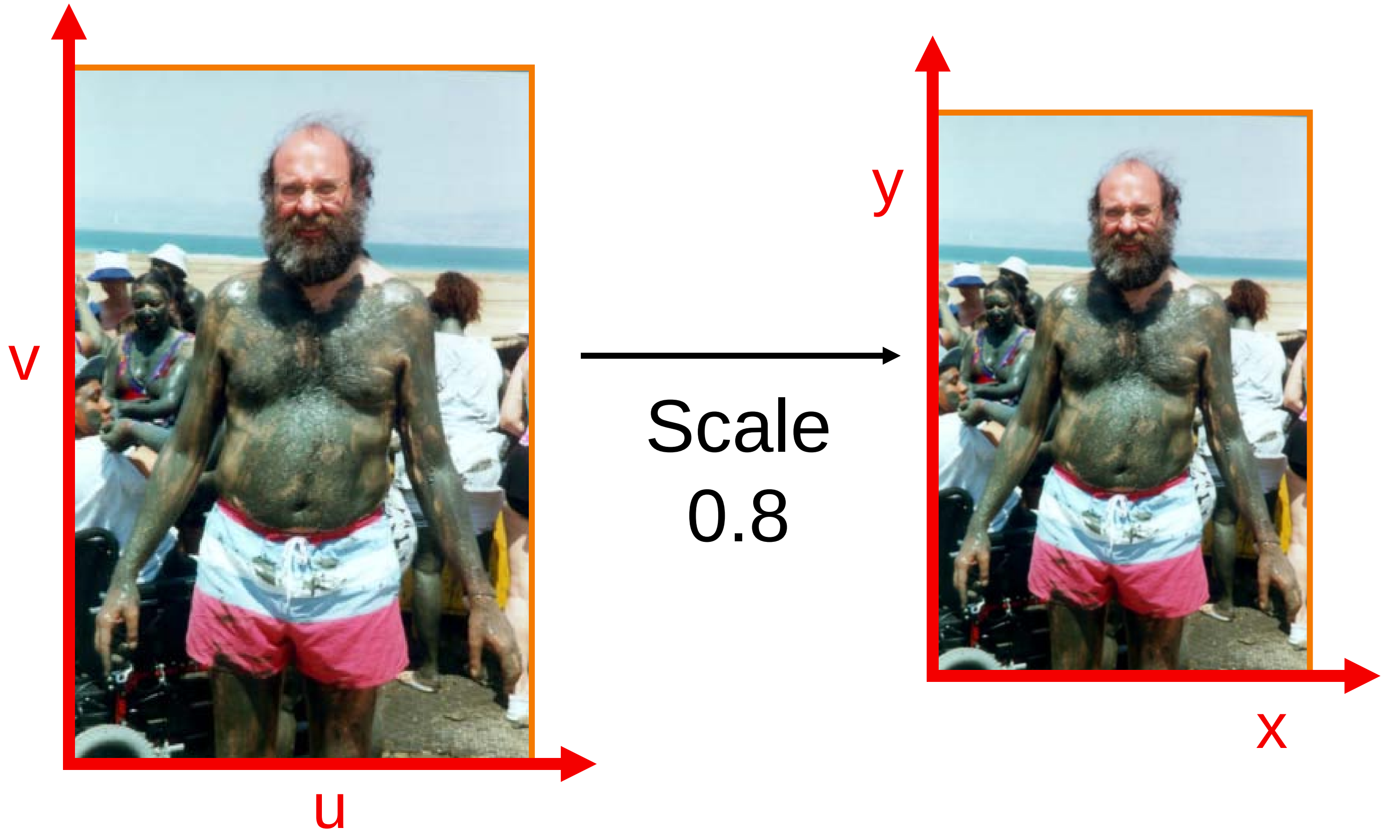
Mapping

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



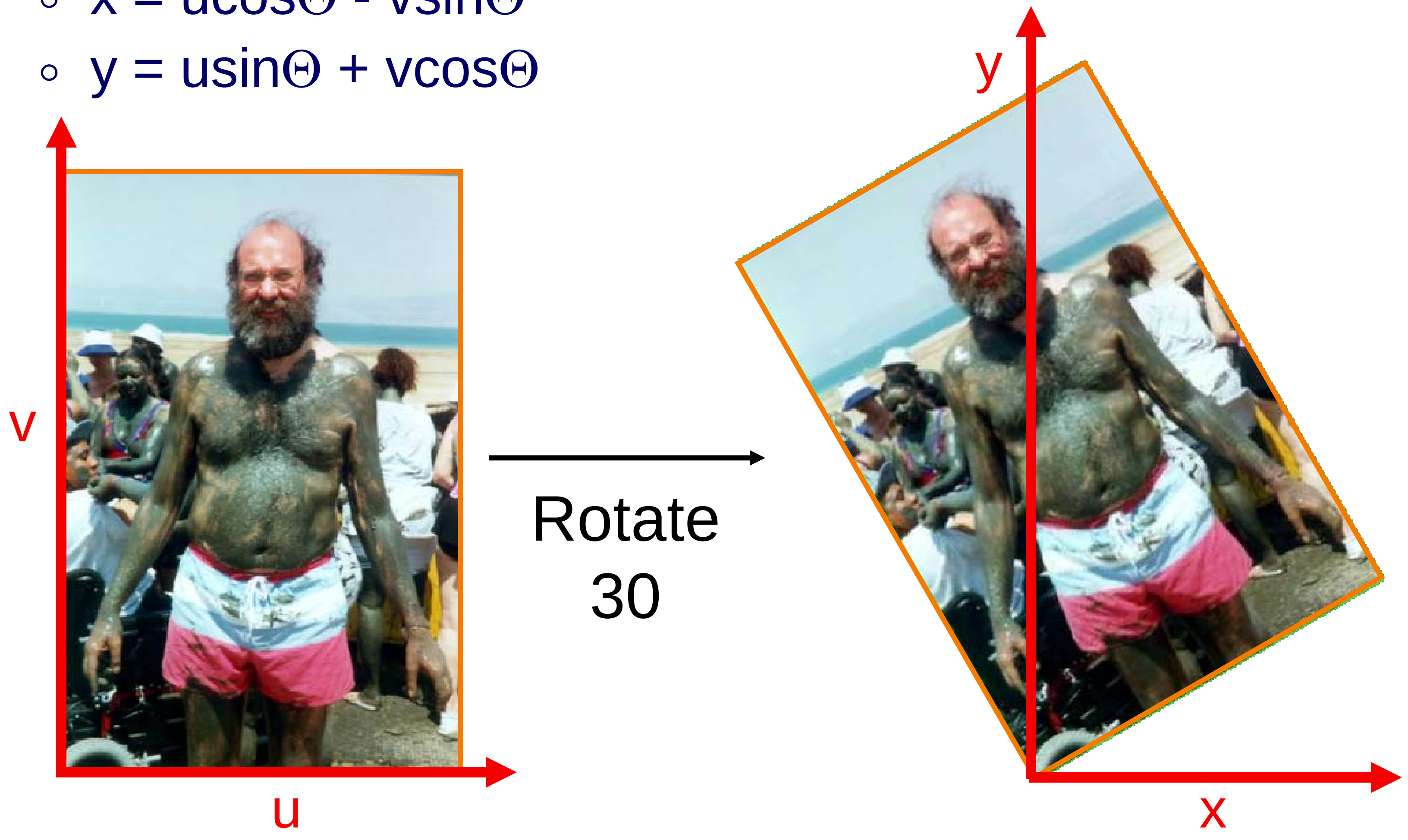
Parametric Mappings

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



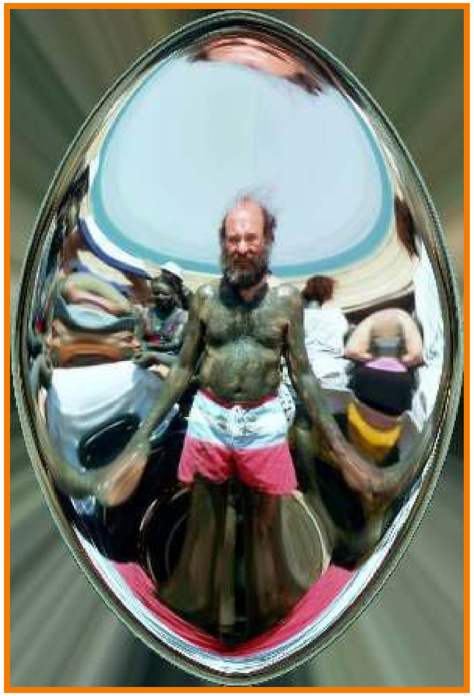
Parametric Mappings

- Rotate by Θ degrees:
  - $x = u\cos\Theta - v\sin\Theta$
  - $y = u\sin\Theta + v\cos\Theta$



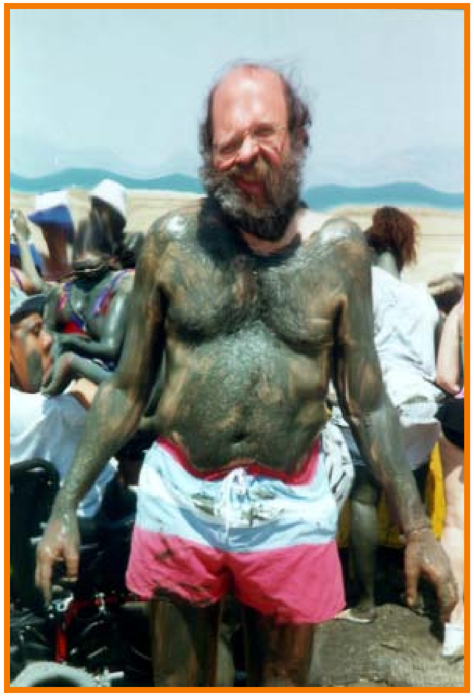
Other Parametric Mappings

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



Fish-eye

"Swirl"



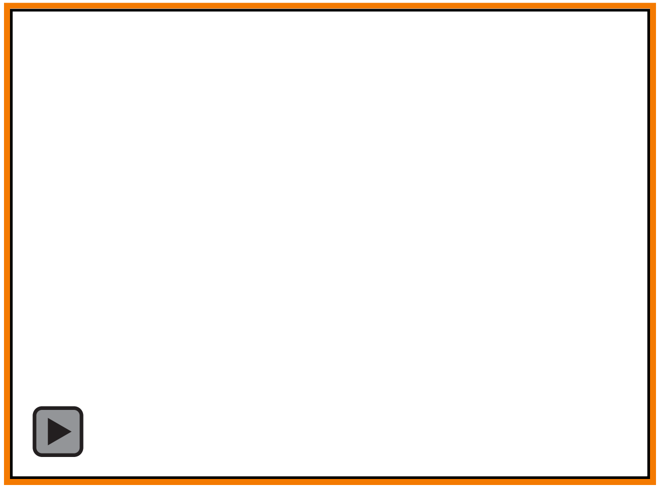
"Rain"

COS426 Examples

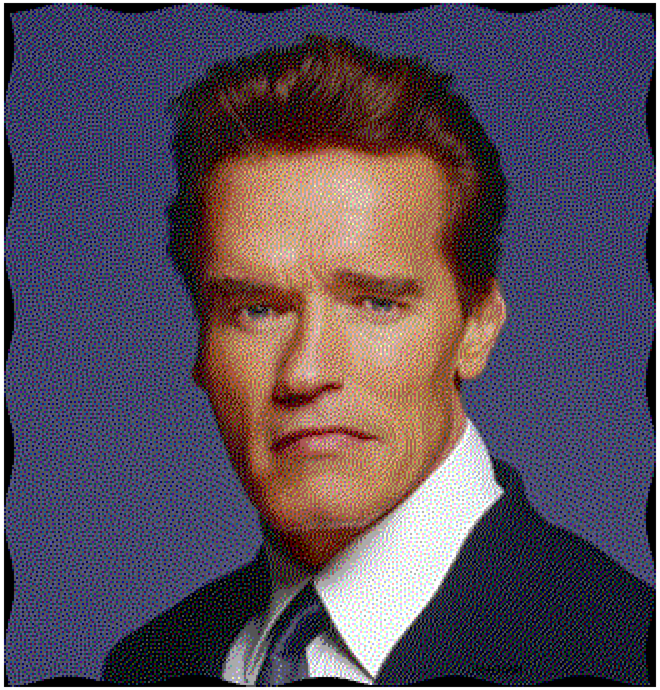
Aditya Bhaskara

Wei Xiang

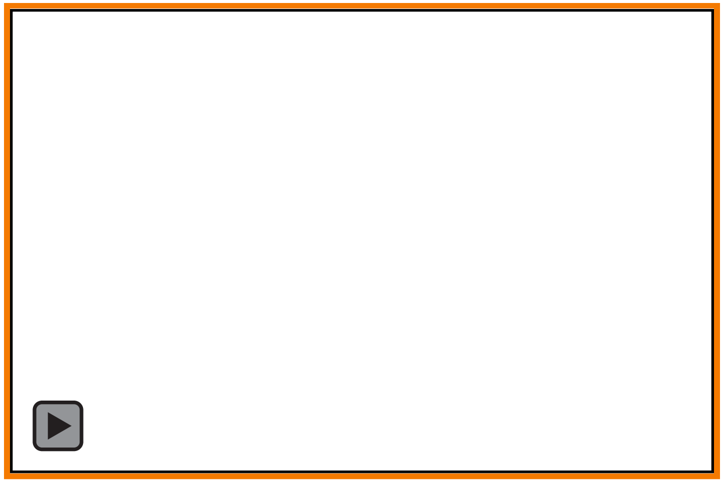
More COS426 Examples



Sid Kapur



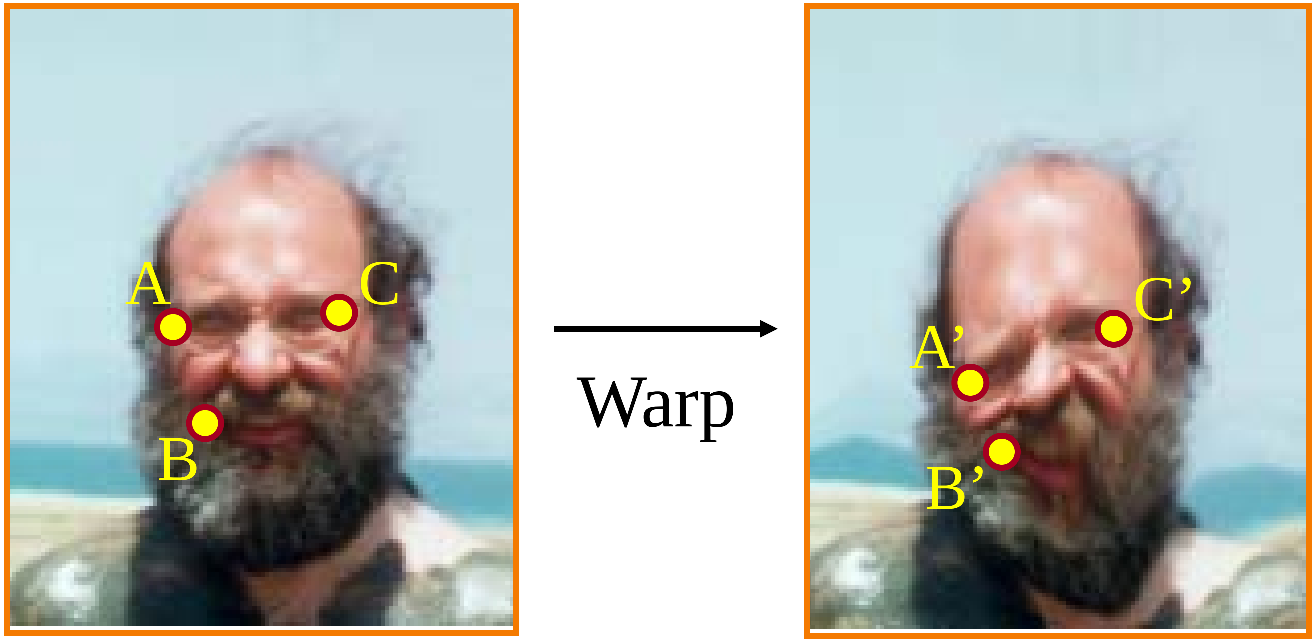
Eirik Bakke



Michael Oranato

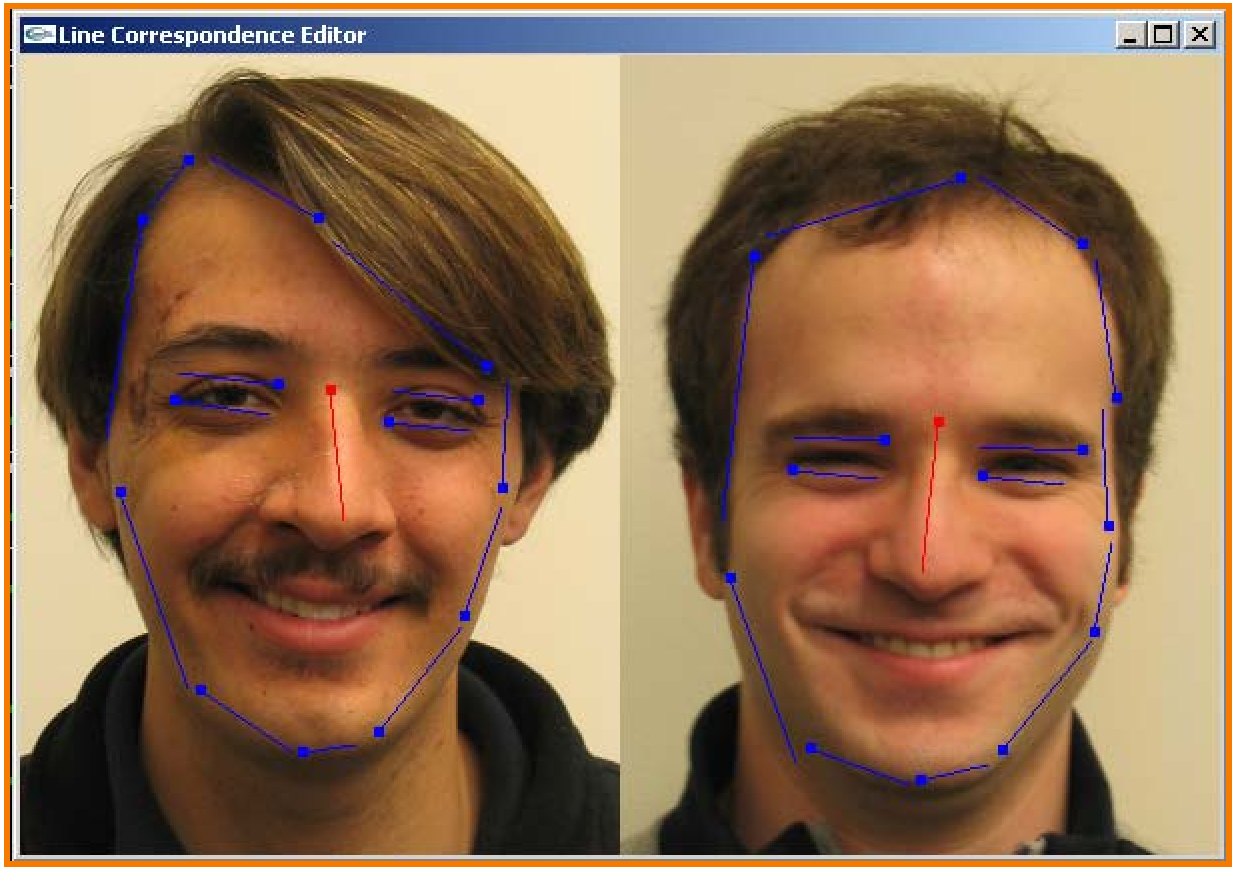
Point Correspondence Mappings

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



Line Correspondence Mappings

- Beier & Neeley use pairs of lines to specify warp



Beier & Neeley
SIGGRAPH 92

Image Warping

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

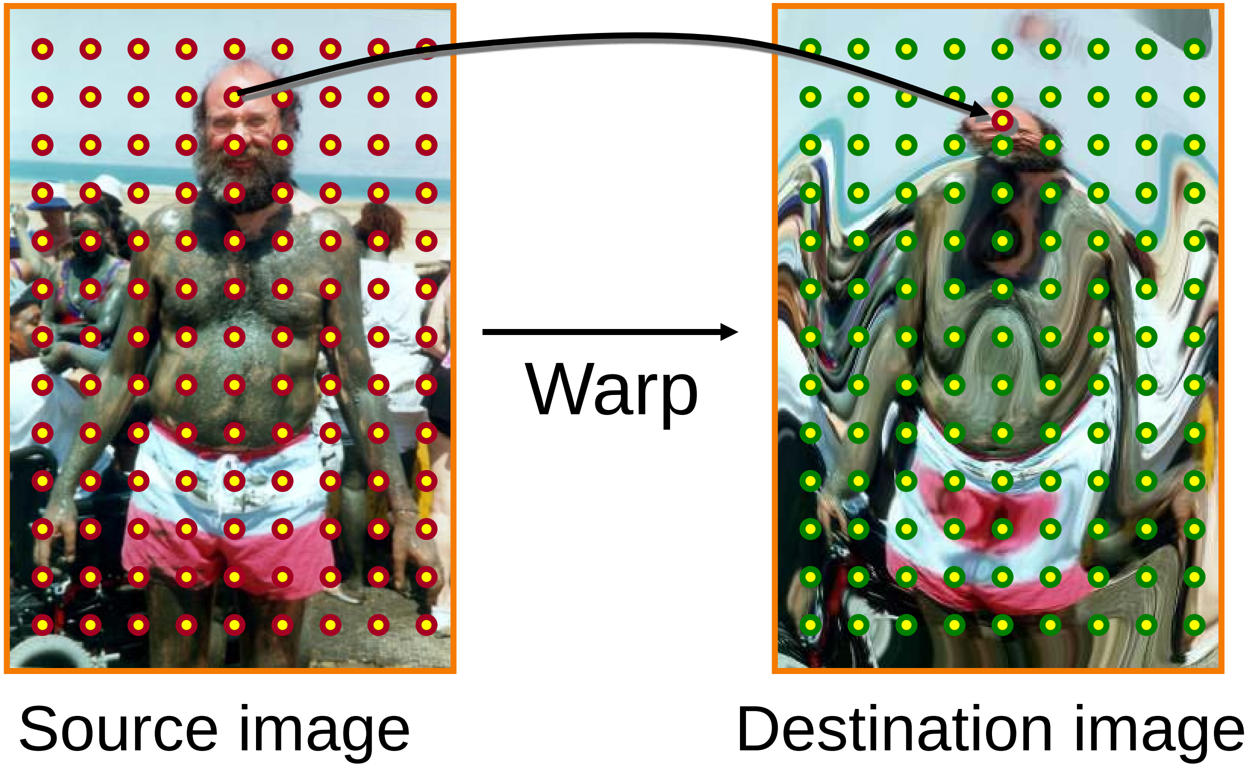
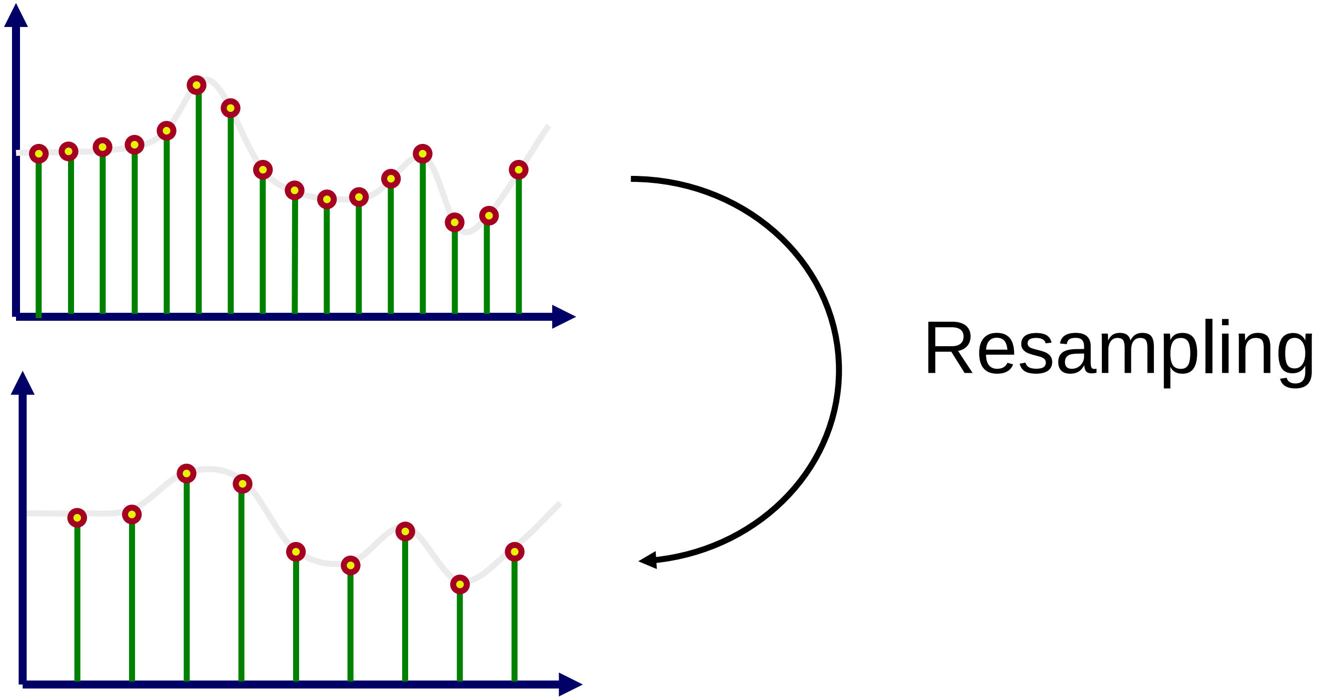


Image Warping

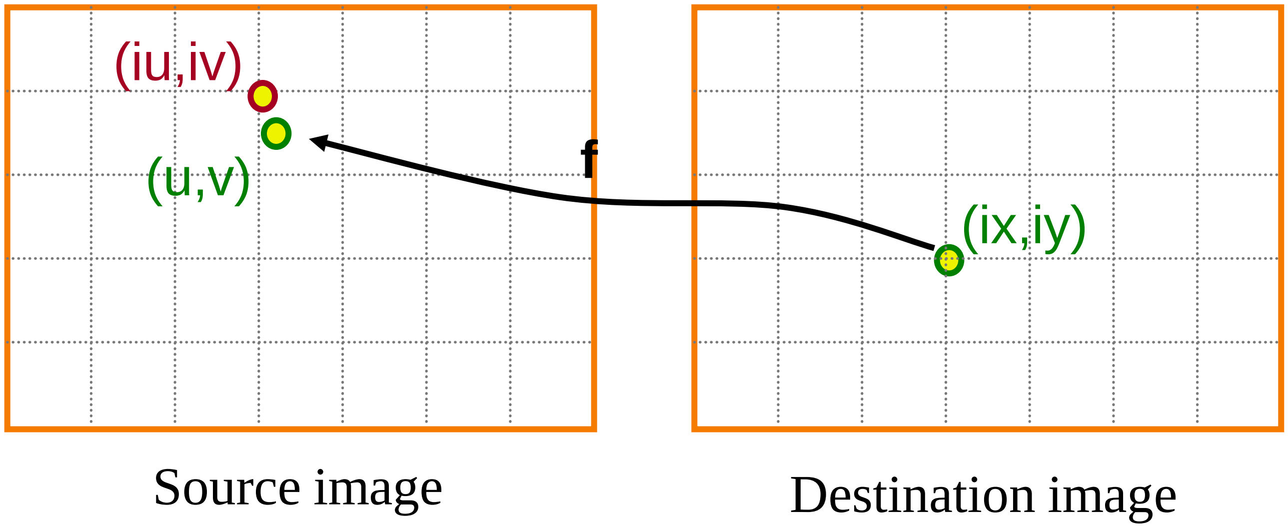
- Image warping requires resampling of image



Point Sampling

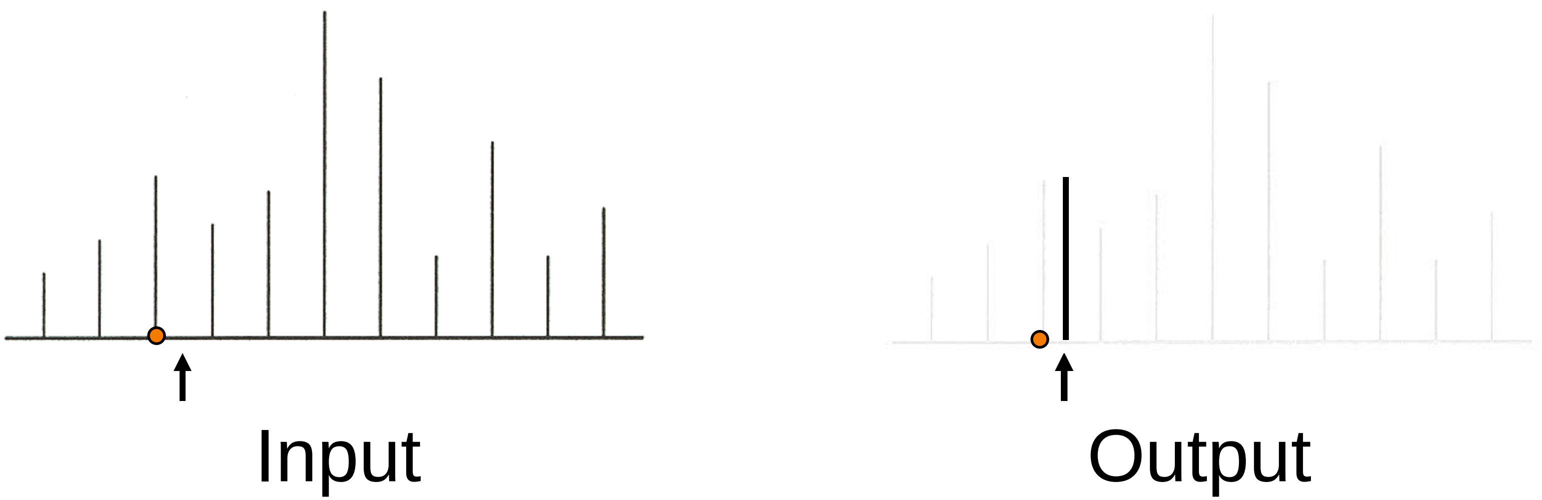
- Possible (poor) resampling implementation:

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



Point Sampling

- Use nearest sample



Point Sampling

Point Sampled: Aliasing!

Correctly Bandlimited

Image Resampling Pipeline

- Ideal resampling requires correct filtering to avoid artifacts
- Reconstruction filter especially important when magnifying
- Bandlimiting filter especially important when minifying

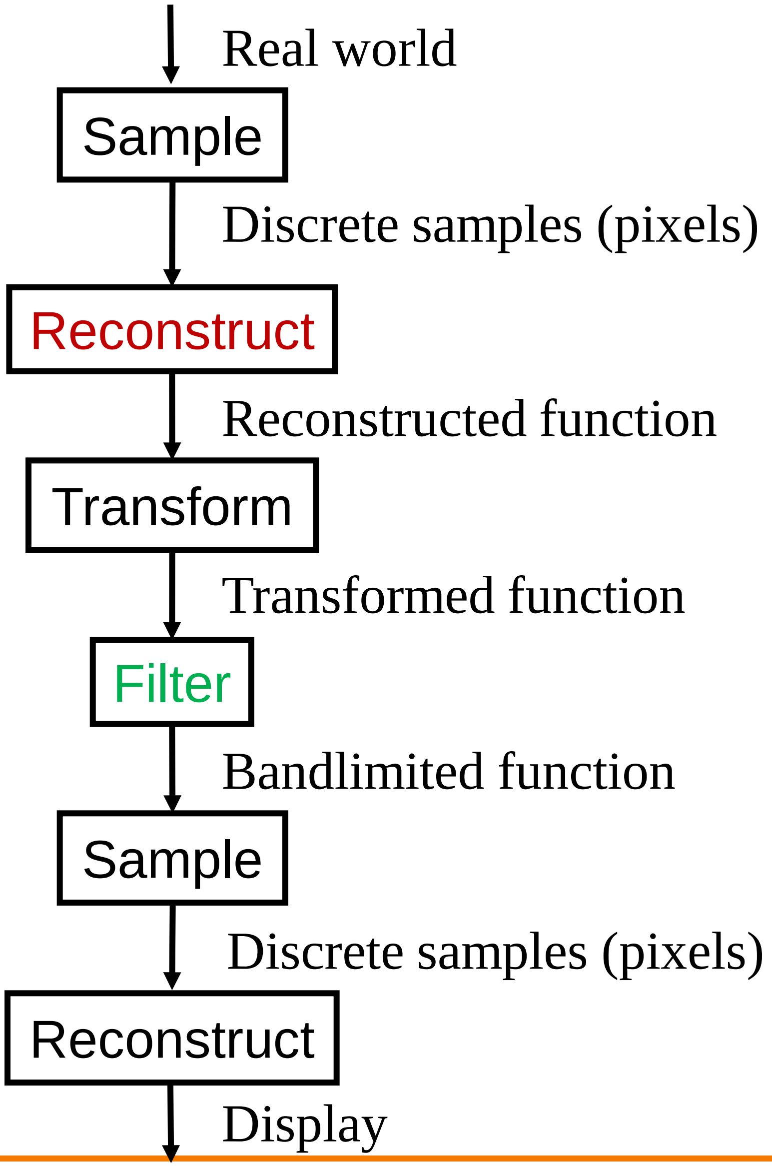
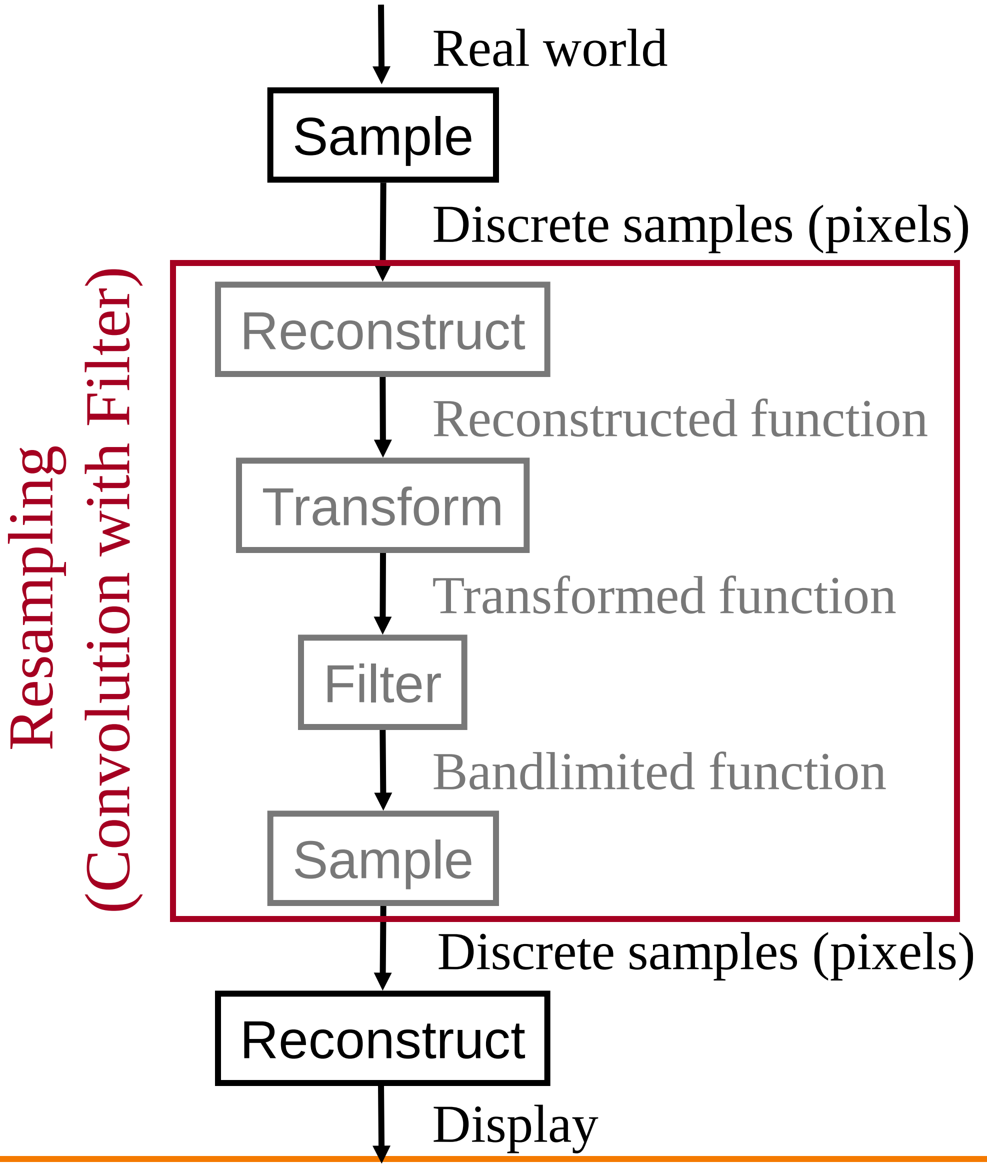


Image Resampling Pipeline

- In practice: Resampling with low-pass filter in order to reduce aliasing artifacts



Resampling with Filter

- Output is weighted average of inputs:

```
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;
}
```

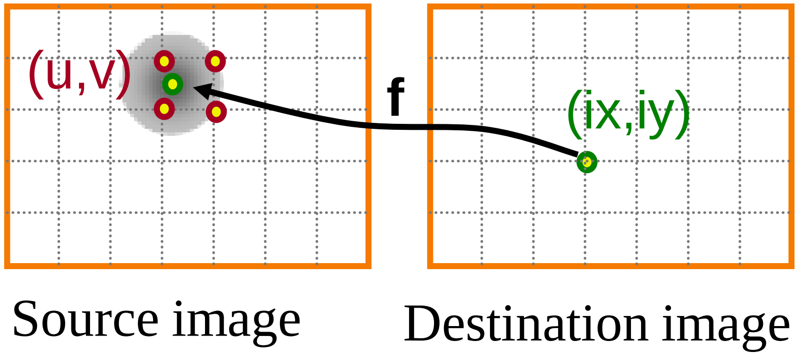
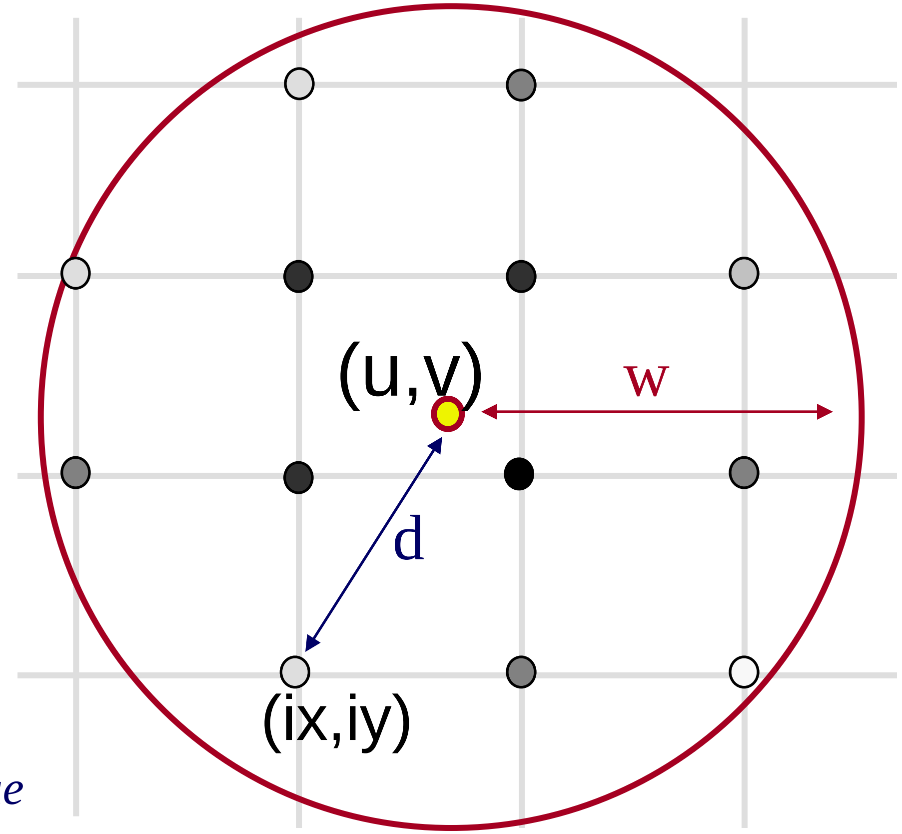


Image Resampling

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

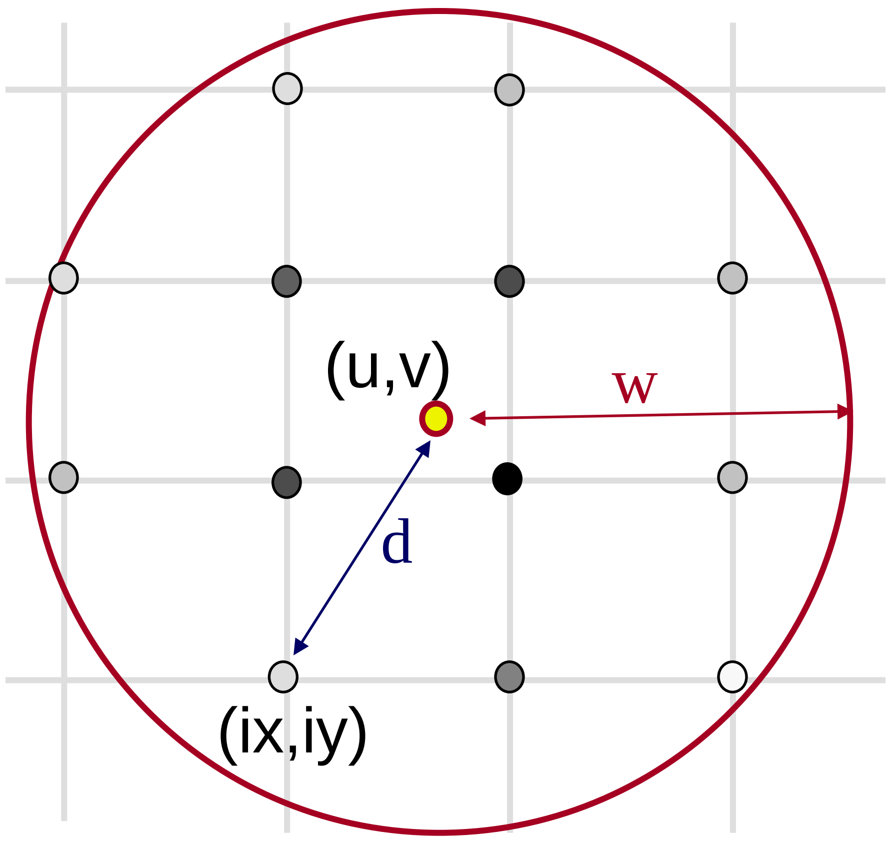
```
dst(ix,iy) = 0;
for (ix = u-w; ix <= u+w; ix++)
  for (iy = v-w; iy <= v+w; iy++)
    d = dist (ix,iy)↔(u,v)
    dst(ix,iy) += k(ix,iy)*src(ix,iy);
```

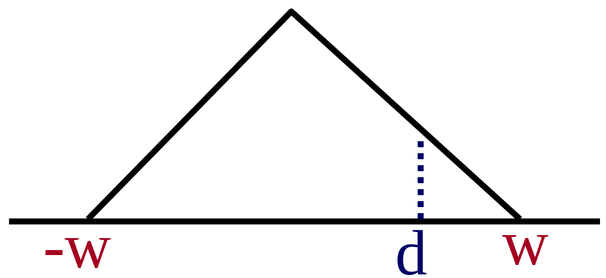


k(ix,iy) represented by gray value

Image Resampling

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





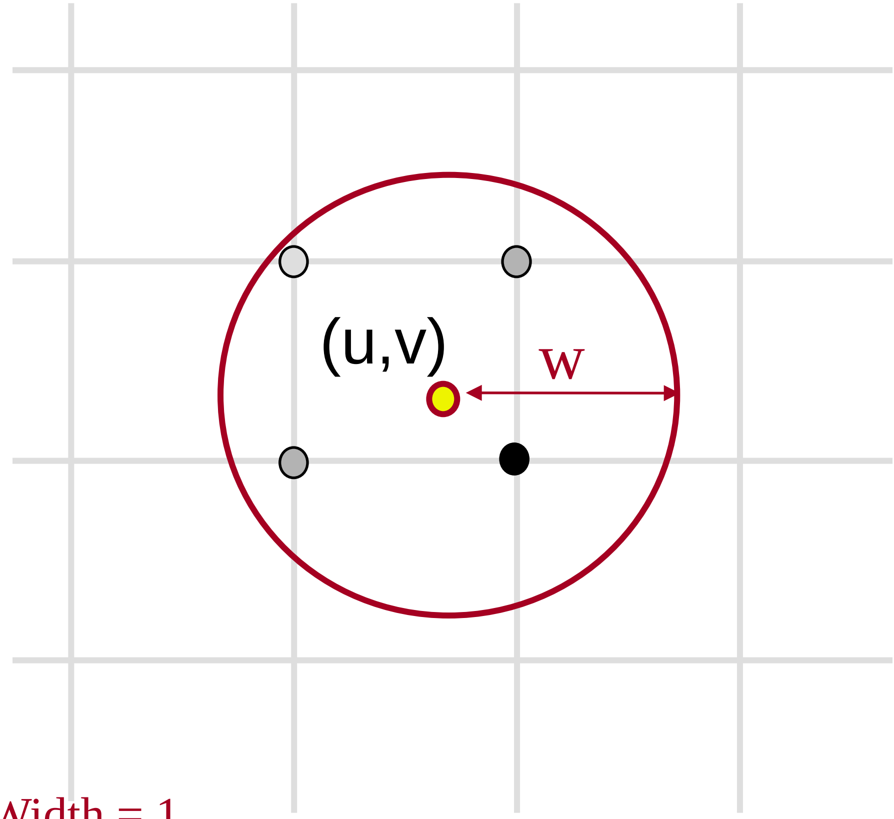
Triangle filter

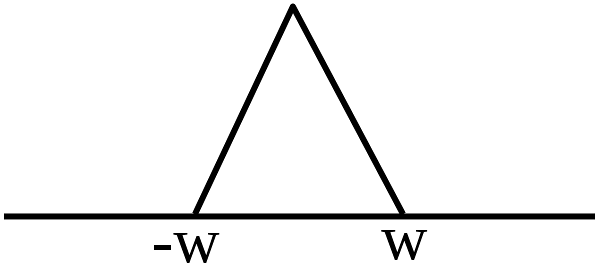
$$k(i,j)=\max(1 - d/w, 0)$$

Filter Width = 2

Image Resampling

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





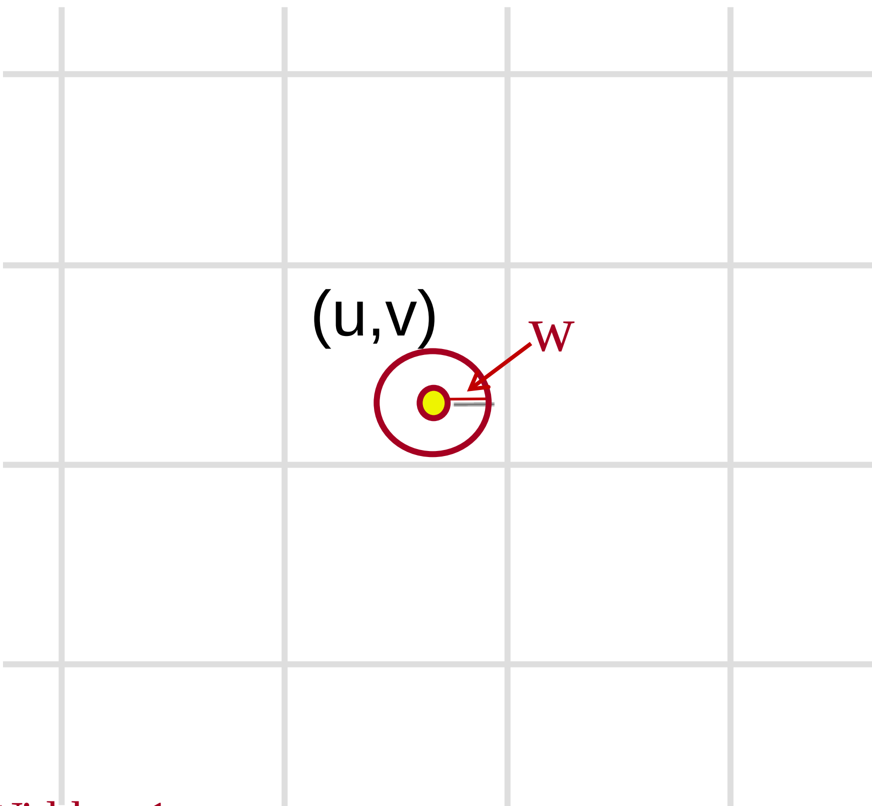
Triangle filter

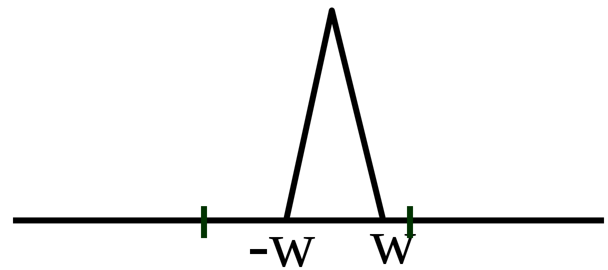
Width of filter affects blurriness

Filter Width = 1

Image Resampling

- What if width (w) is small than sample spacing?



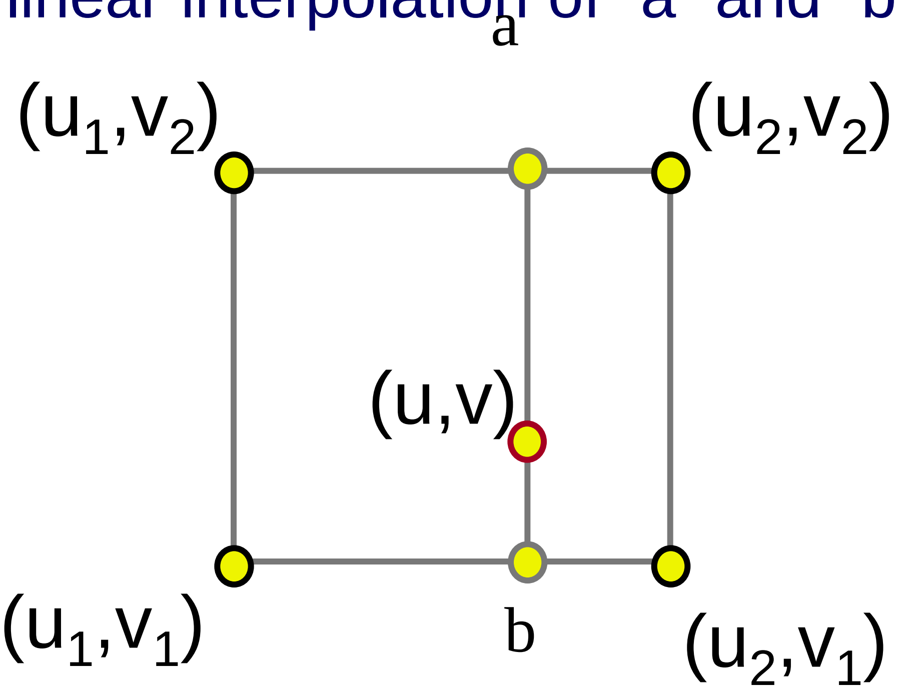


Triangle filter

Filter Width < 1

Image Resampling (with width <= 1)

- Reconstruction filter:
  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”

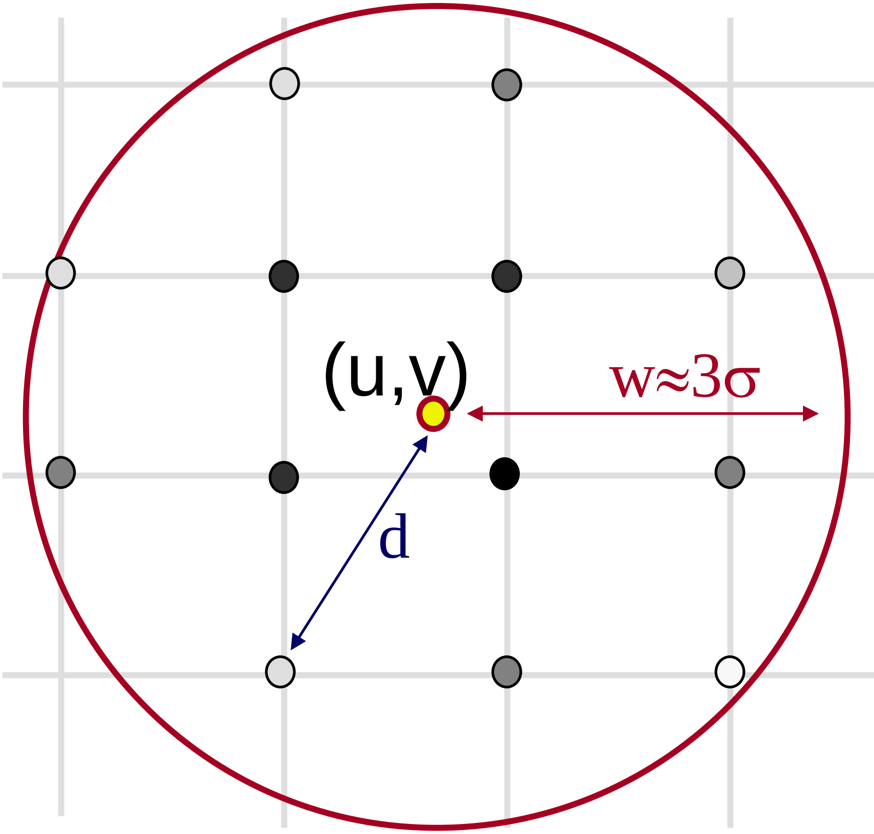


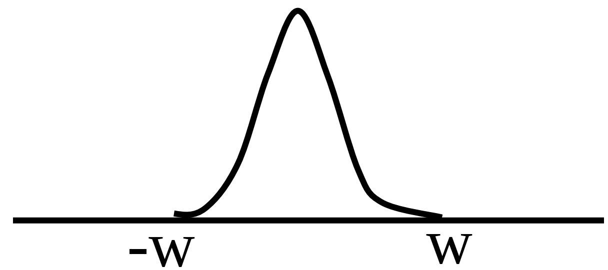
Filter Width < 1

Gaussian Filtering

- Kernel is Gaussian function

$$G(d,\sigma) = e^{-d^2/(2\sigma^2)}$$





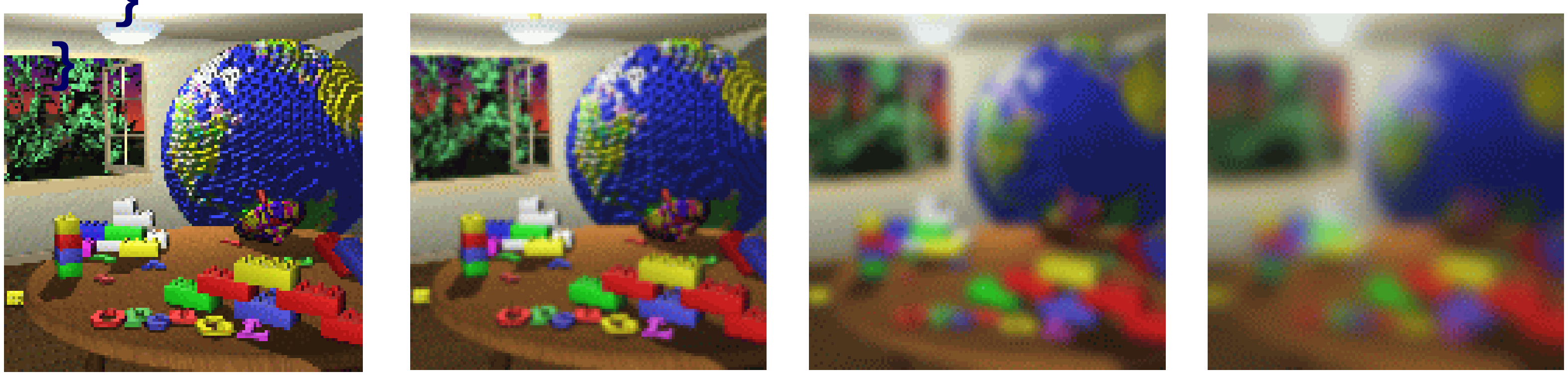
Gaussian Function

- Drops off quickly, but never gets to exactly 0
- In practice: compute out to w ~ 3σ

Putting it All Together

- Possible implementation of image blur:

```
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);
    }
  }
}
```

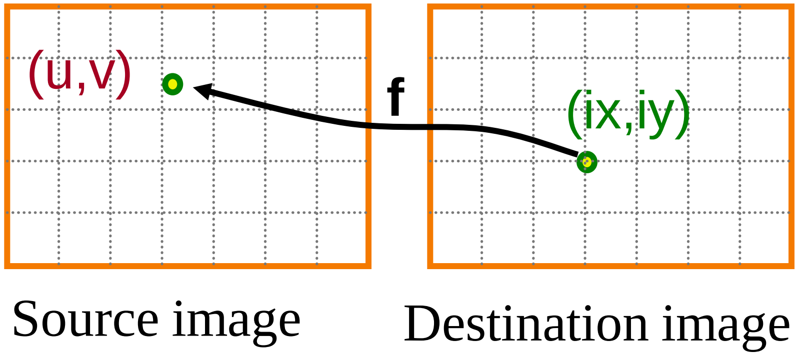


Increasing sigma

Putting it All Together

- Possible implementation of image scale:

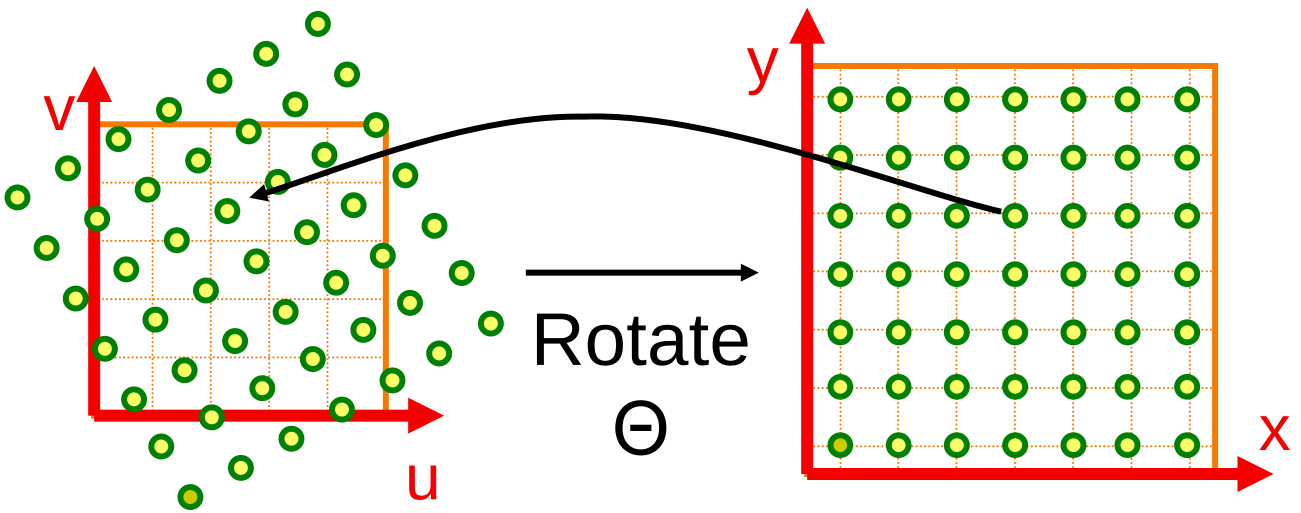
```
Scale(src, dst, sx, sy) {
  w ≈ max(1/sx,1/sy);
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix / sx;
      float v = iy / sy;
      dst(ix,iy) = Resample(src,u,v,k,w);
    }
  }
}
```



Putting it All Together

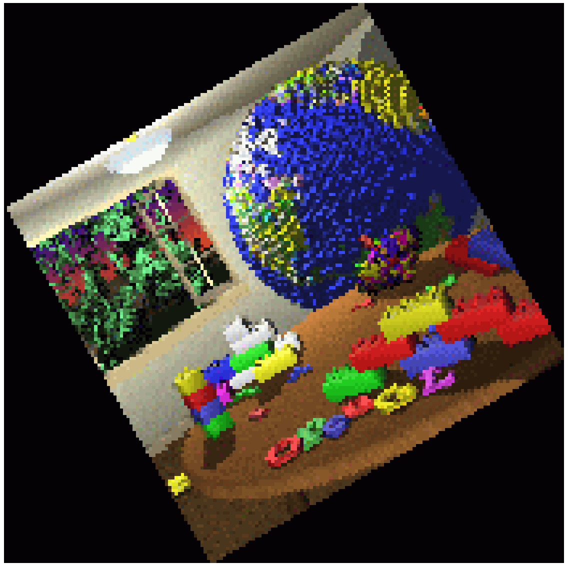
- Possible implementation of image rotation:

```
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,k,w);
      }
   }
}
```

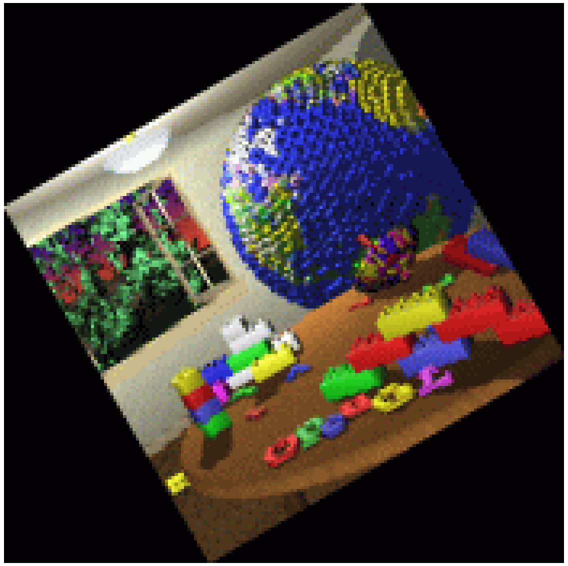


Sampling Method Comparison

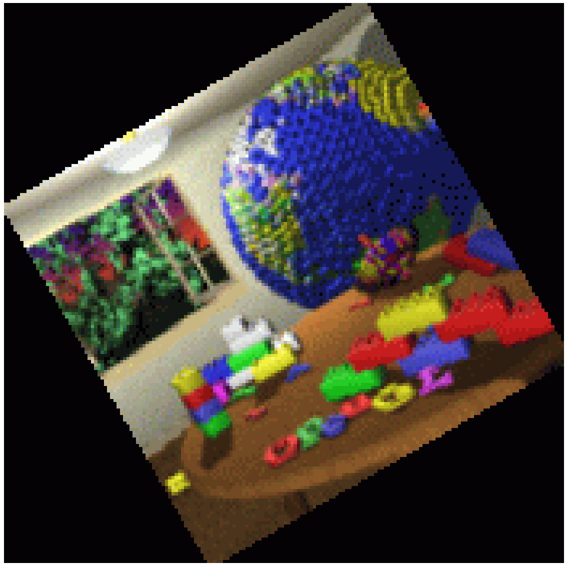
- Trade-offs
  - Aliasing versus blurring
  - Computation speed



Point



Triangle

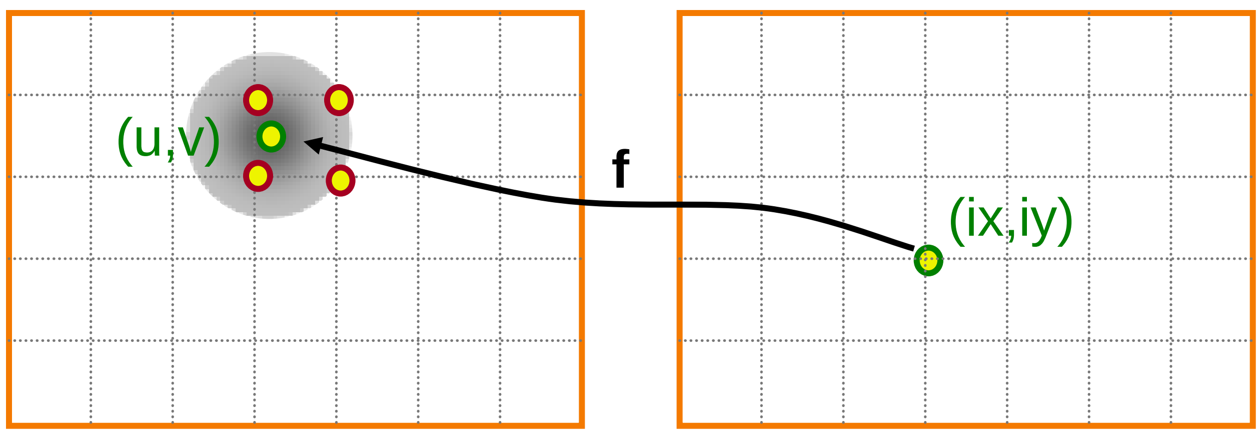


Gaussian

Forward vs. Reverse Mapping

- 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 = fx^-1(ix,iy);
      float v = fy^-1(ix,iy);
      dst(ix,iy) = Resample(src,u,v,w);
    }
  }
}
```

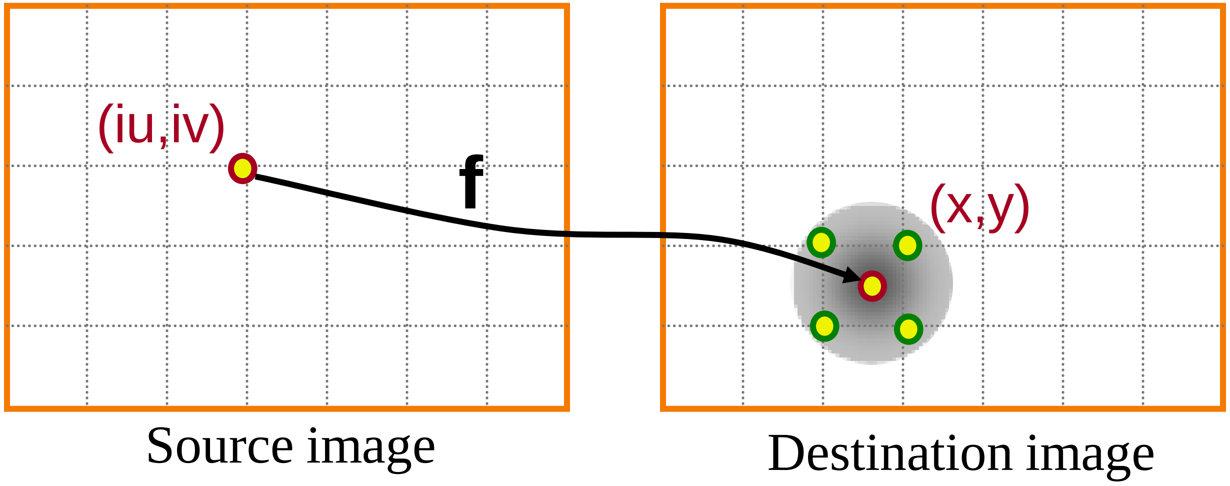


Source image Destination image

Forward vs. Reverse Mapping

- Forward mapping:

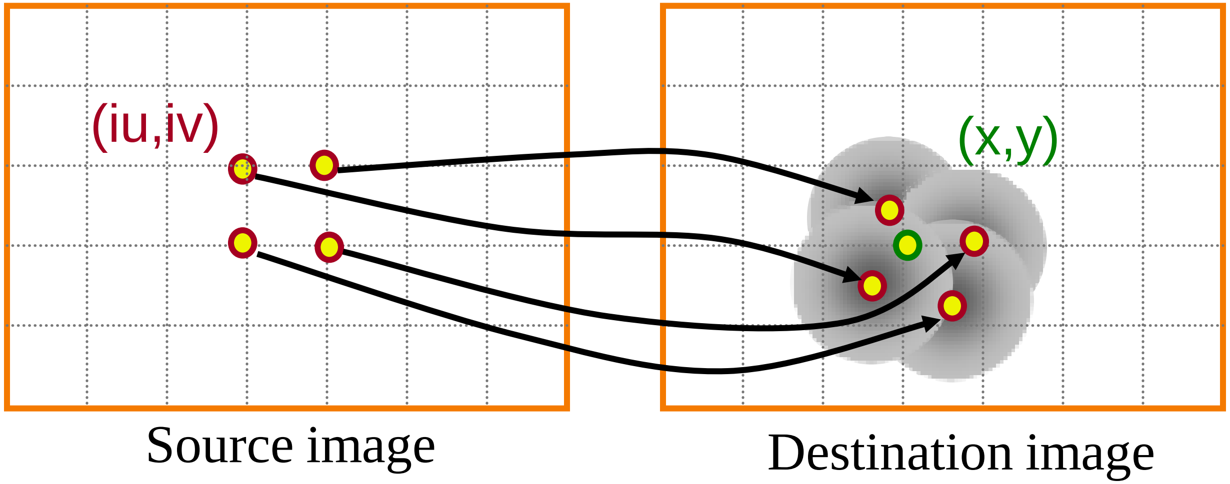
```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = fx(iu,iv);
      float y = fy(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv),x,y,k,w);
    }
  }
}
```



Forward vs. Reverse Mapping

- Forward mapping:

```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = fx(iu,iv);
      float y = fy(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv),x,y,k,w);
    }
  }
}
```

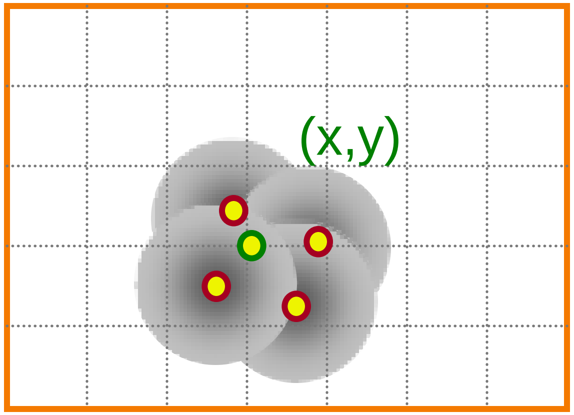


Forward vs. Reverse Mapping

- 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);
    float w ≈ 1 / scale(x, y);
    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);
      }
    }
  }
}
```

Problem?

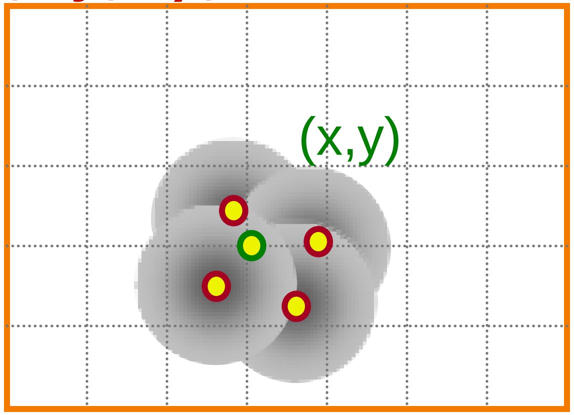


Destination image

Forward vs. Reverse Mapping

- 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);
    float w ≈ 1 / scale(x, y);
    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)
```



Destination image

Forward vs. Reverse Mapping

- Tradeoffs?

Forward vs. Reverse Mapping

- Tradeoffs:
  - Forward mapping:
    - Requires separate buffer to store weights
  - Reverse mapping:
    - Requires inverse of mapping function

Summary

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

Next Time…

- Changing intensity/color
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Add random noise
- Filtering over neighborhoods
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median
- Moving image locations
  - Scale
  - Rotate
  - Warp
- Combining images
  - Composite
  - Morph
- Quantization
- Spatial / intensity tradeoff
  - Dithering