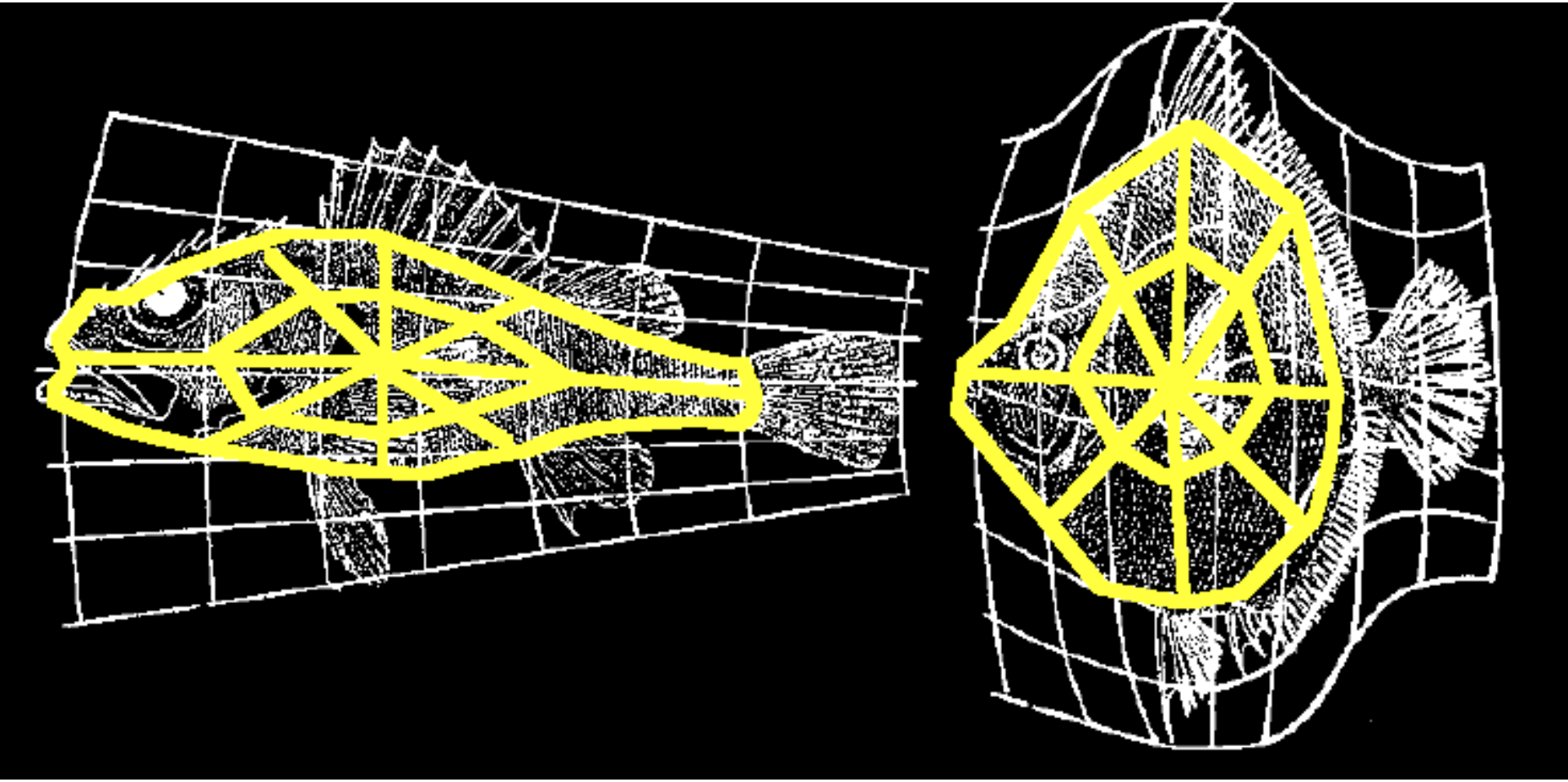
Krivljenje slike - warping

Princip 2D krivljenja



Demo

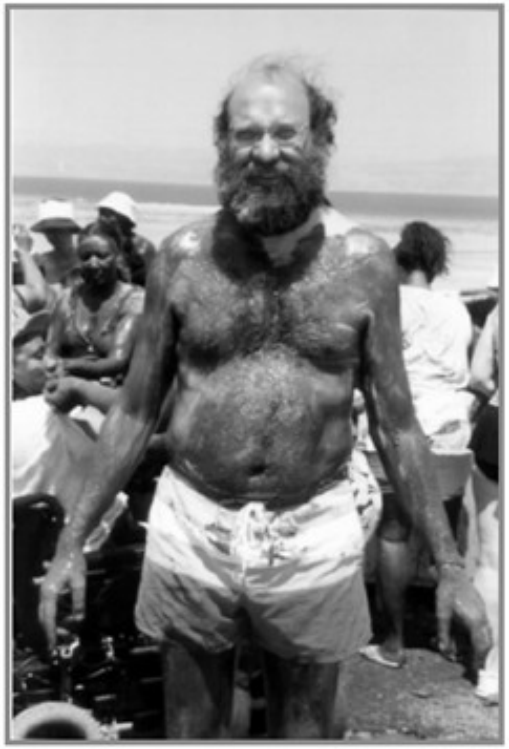
Krivljenje (Warping)

- A warp is a 2-D geometric transformation and generates a distorted image when it is applied to an image.

- **Warping** an image means : apply a given deformation to it.

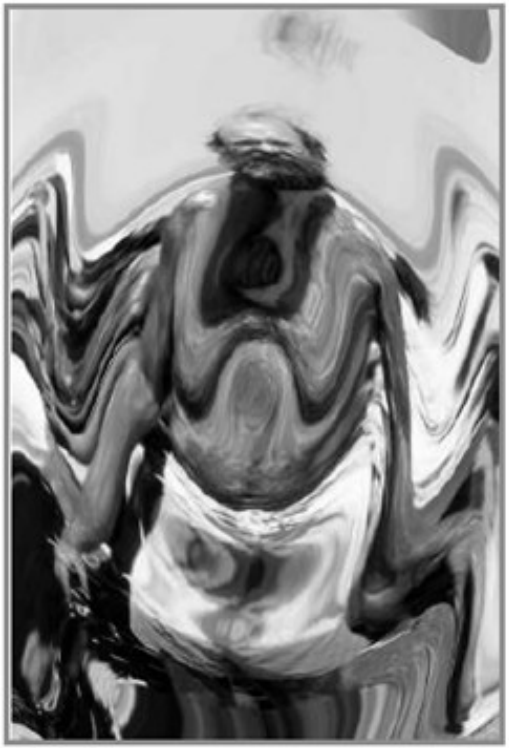
- Two ways to warp an image:-
  - → Forward mapping.
  - → Reverse mapping.

Krivljenje (Warping)



Source image

warp

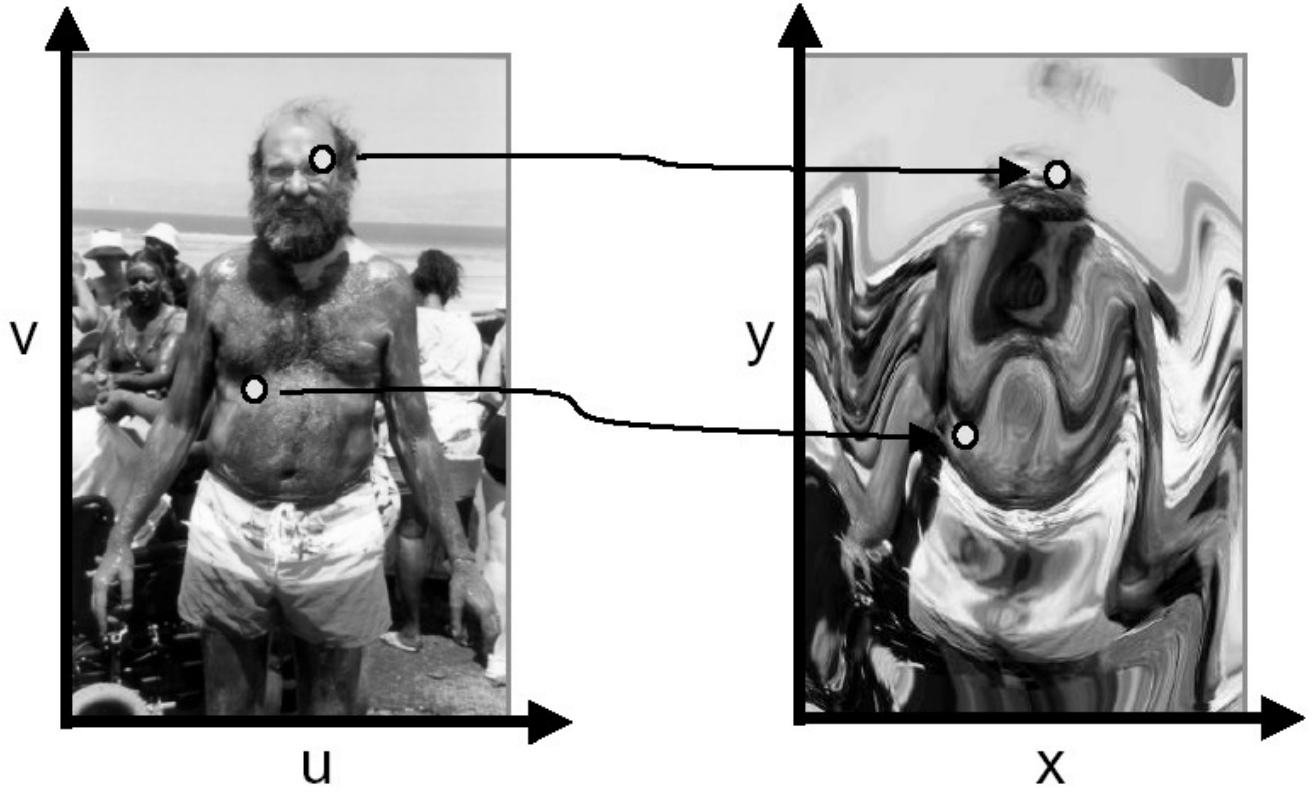


Destination image

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

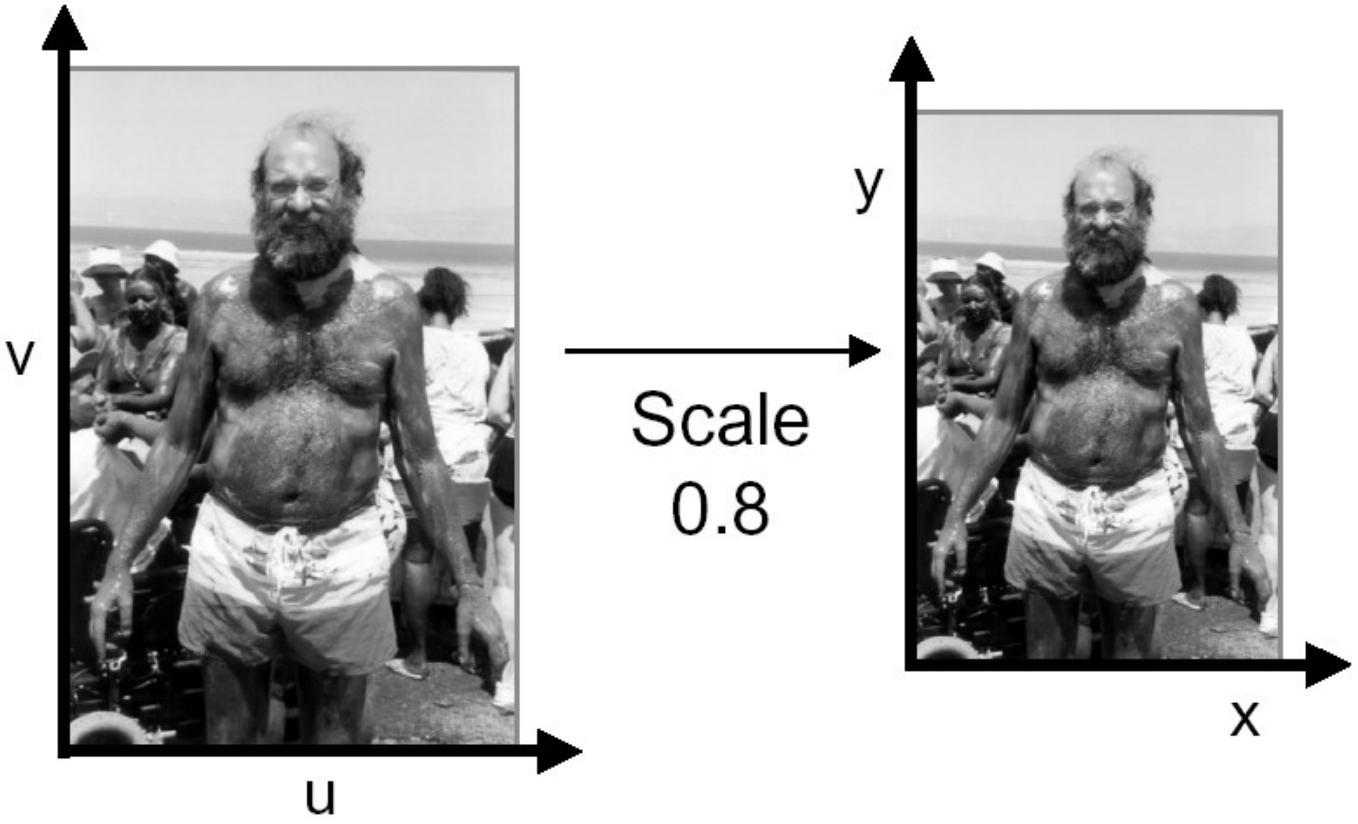
Preslikava (mapping)

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



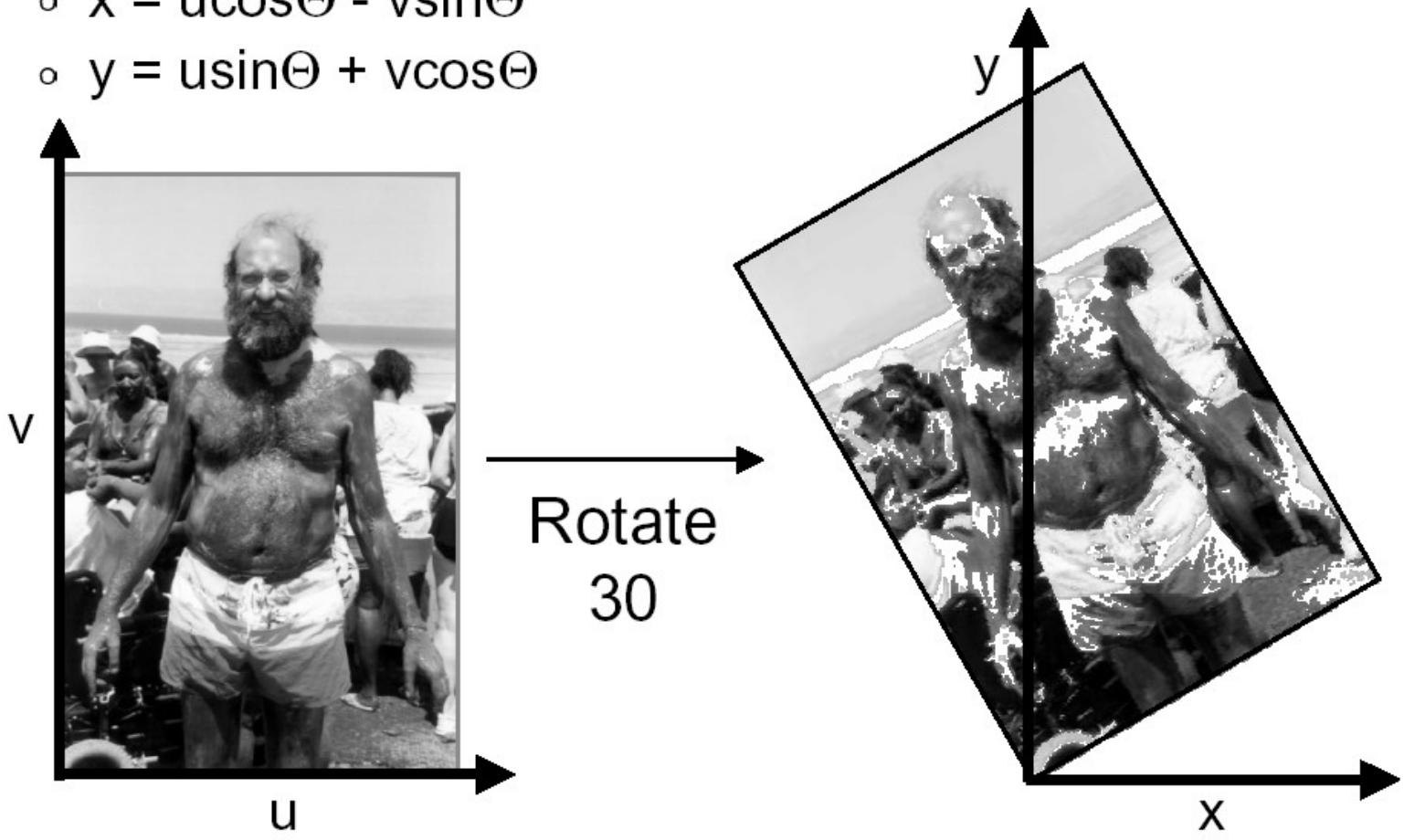
Primer preslikave

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



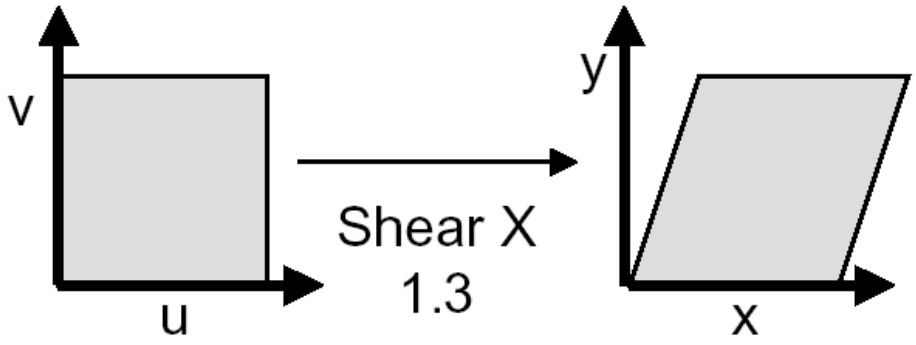
Primer preslikave

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

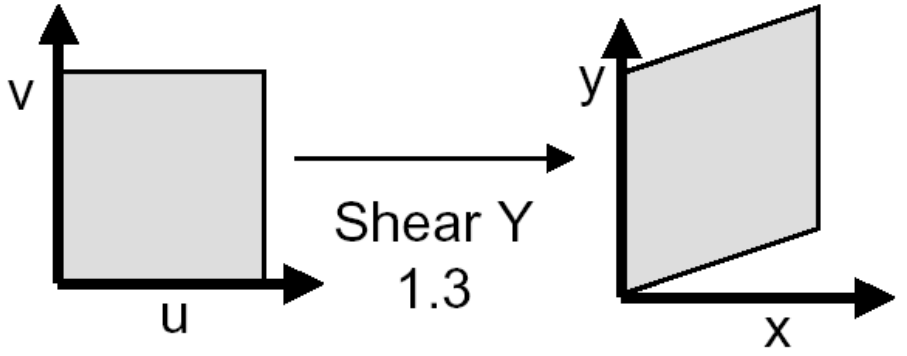


Primer preslikave

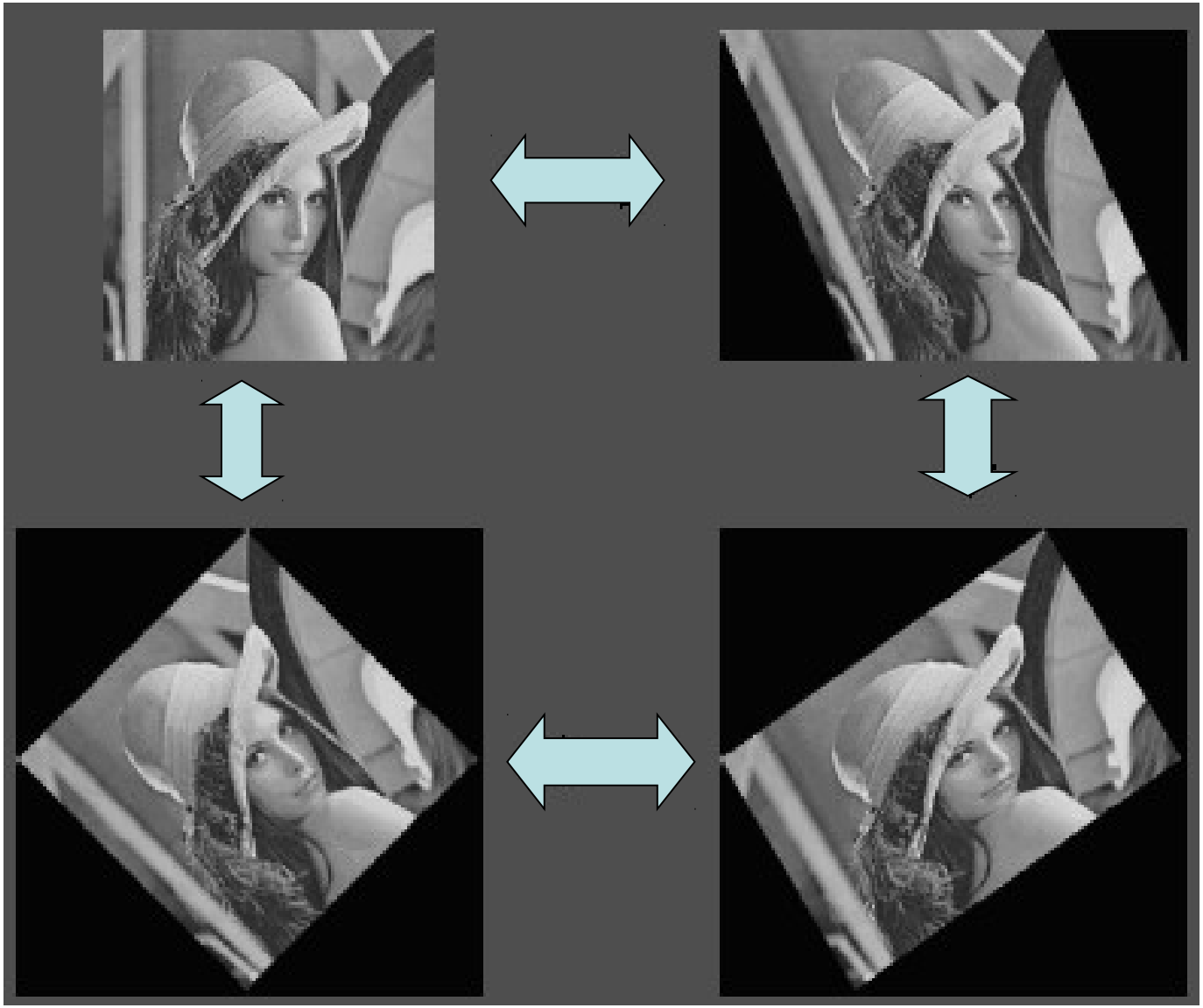
- Shear in X by *factor:*
  - x = u + *factor* * v
  - y = v



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

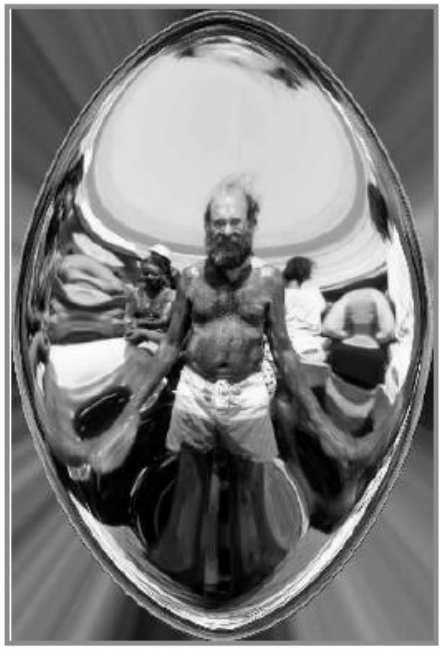


Primer krivljenja



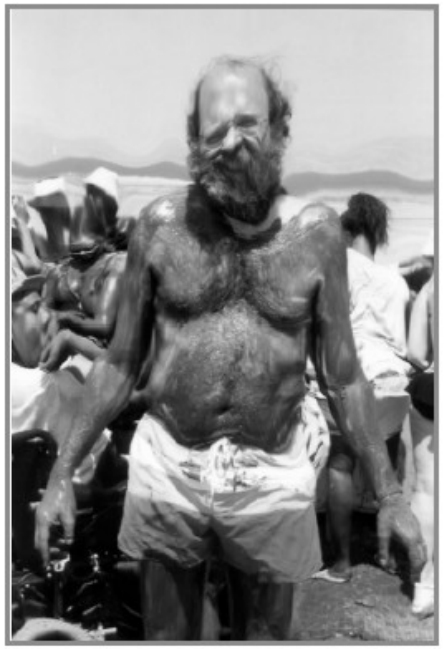
Other Mappings

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



Fish-eye

"Swirl"

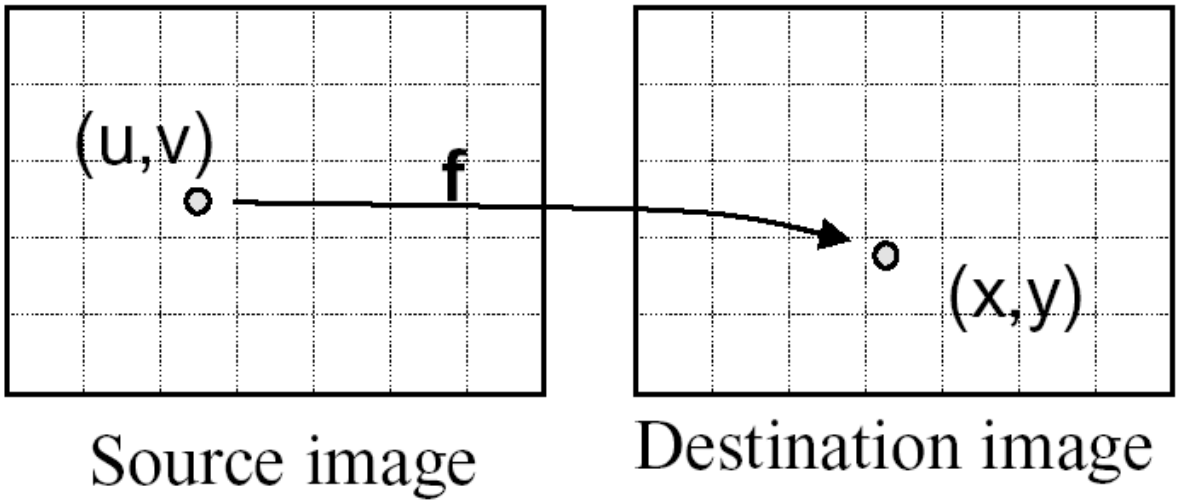


"Rain"

Image Warping Implementation I

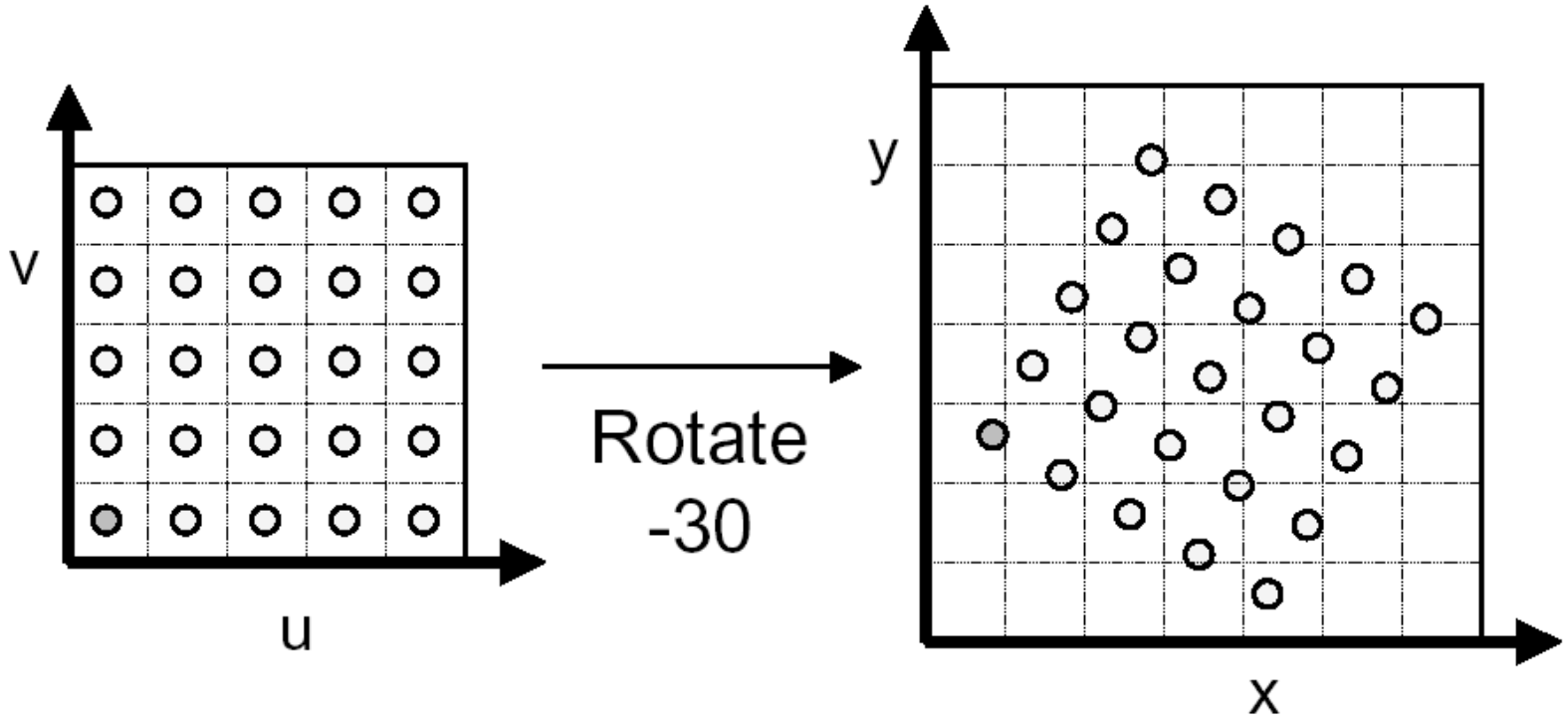
- Forward mapping:

```
for (int u = 0; u < umax; u++) {
  for (int v = 0; v < vmax; v++) {
    float x = f_x(u,v);
    float y = f_y(u,v);
    dst(x,y) = src(u,v);
  }
}
```



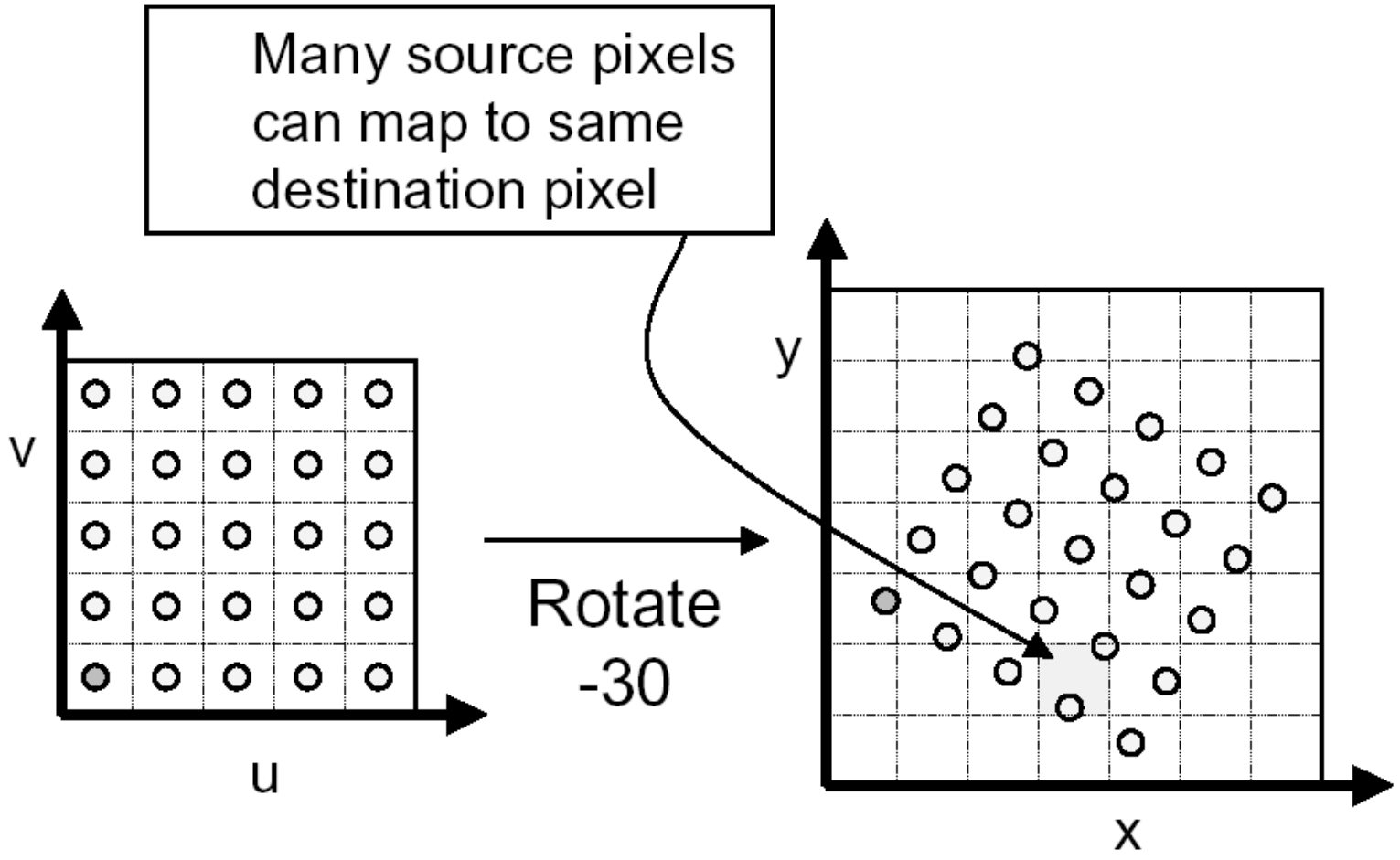
Forward Mapping

- Iterate over source image



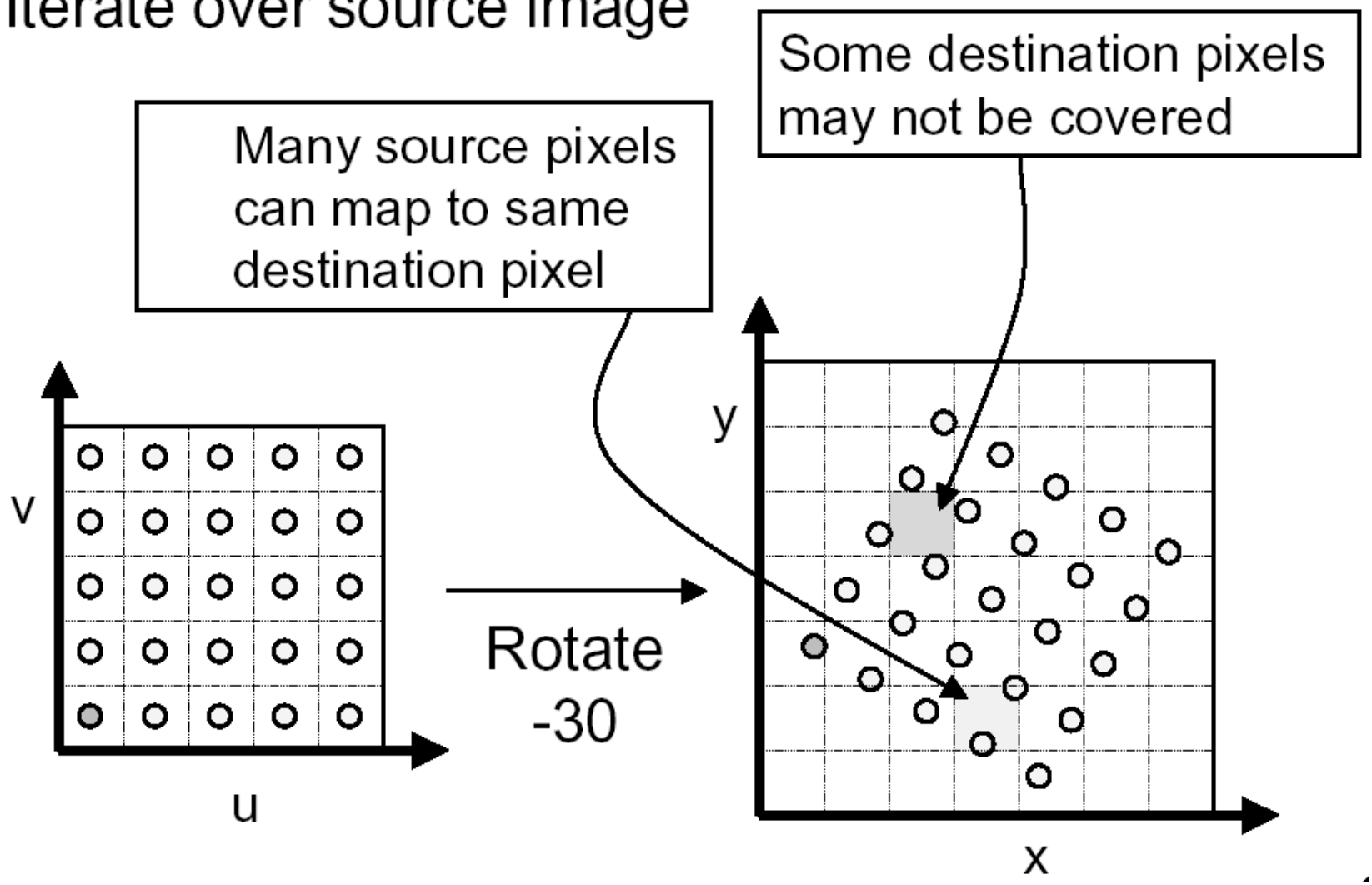
Forward Mapping

- Iterate over source image



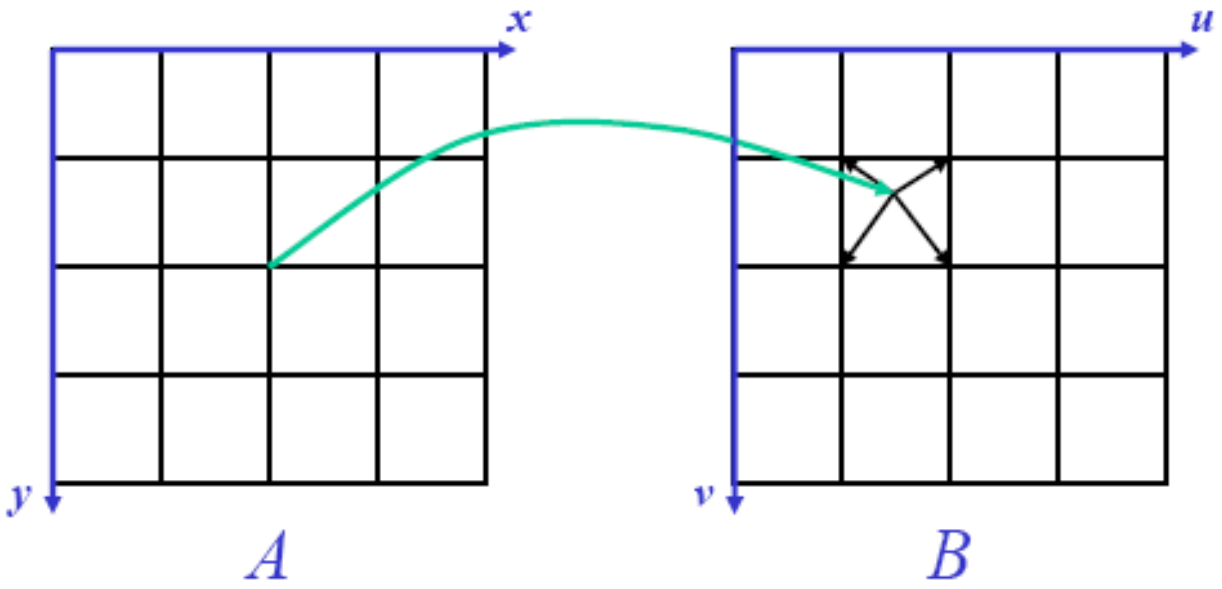
Forward Mapping

- Iterate over source image



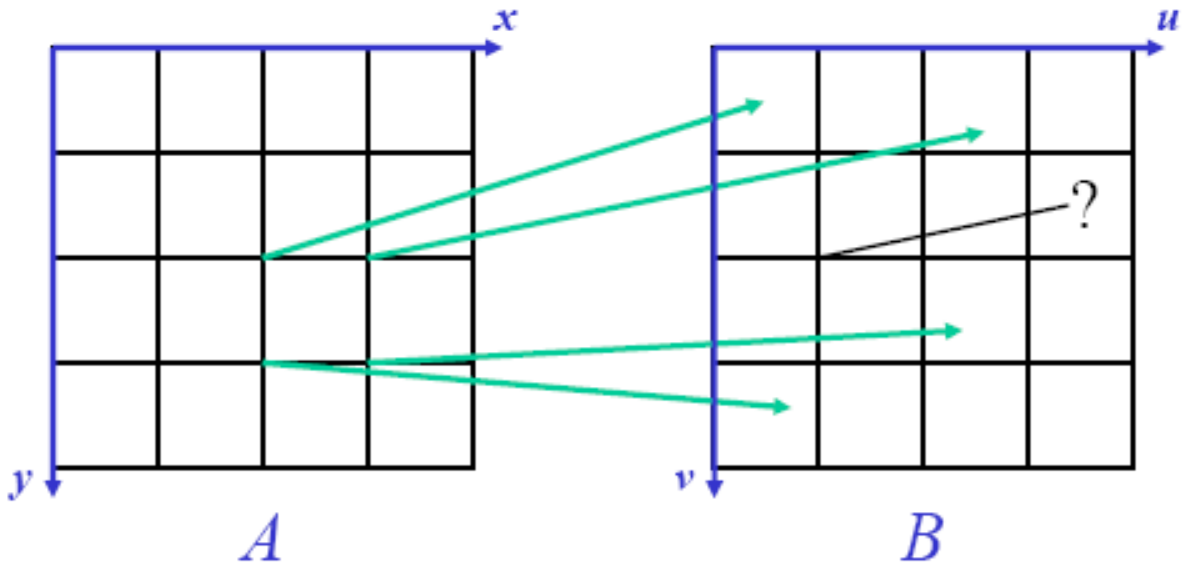
Forward Mapping: Problems

- Doesn't always map *to* pixel locations
- Solution: spread out effect of each pixel, e.g. by bilinear interpolation



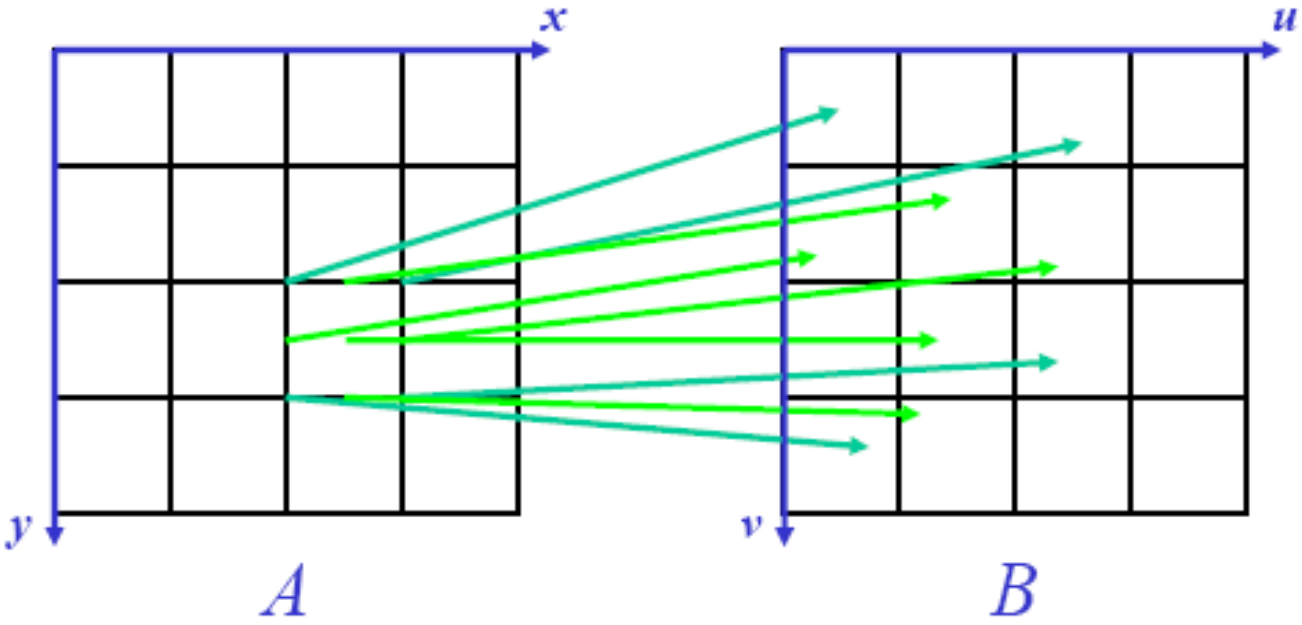
Forward Mapping: Problems

- May produce holes in the output



Forward Mapping: Problems

- May produce holes in the output
- Solution: sample source image (A) more often
  - Still can leave holes



Backward Mapping

Let $x(u, v)$ and $y(u, v)$ be an inverse mapping from location (x, y) to (u, v):

$$B[u, v] = A[x(u, v), y(u, v)]$$

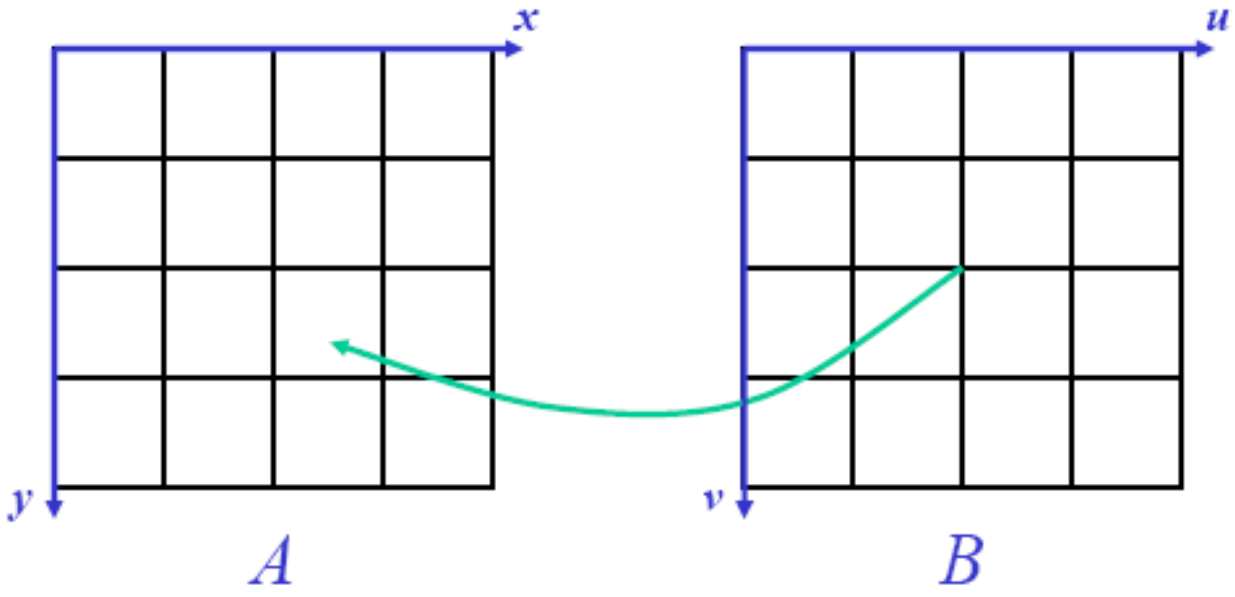
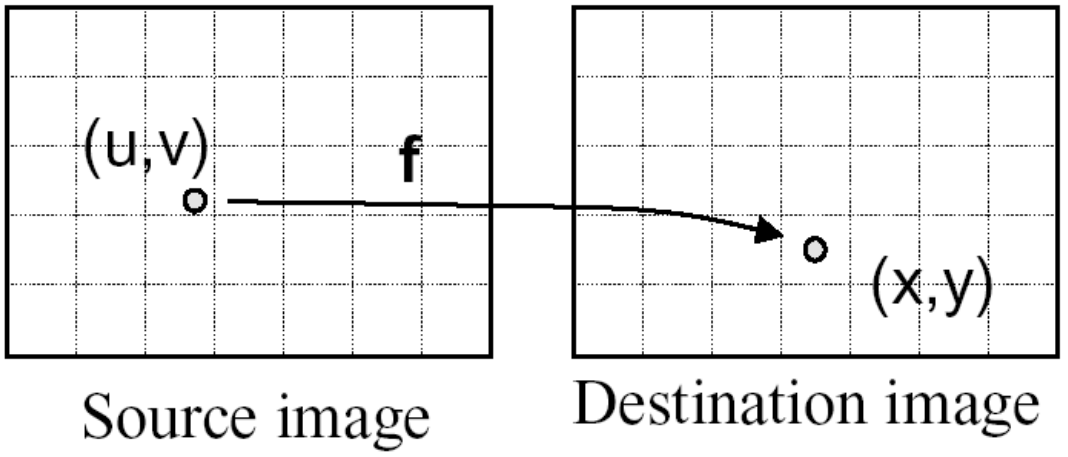


Image Warping Implementation II

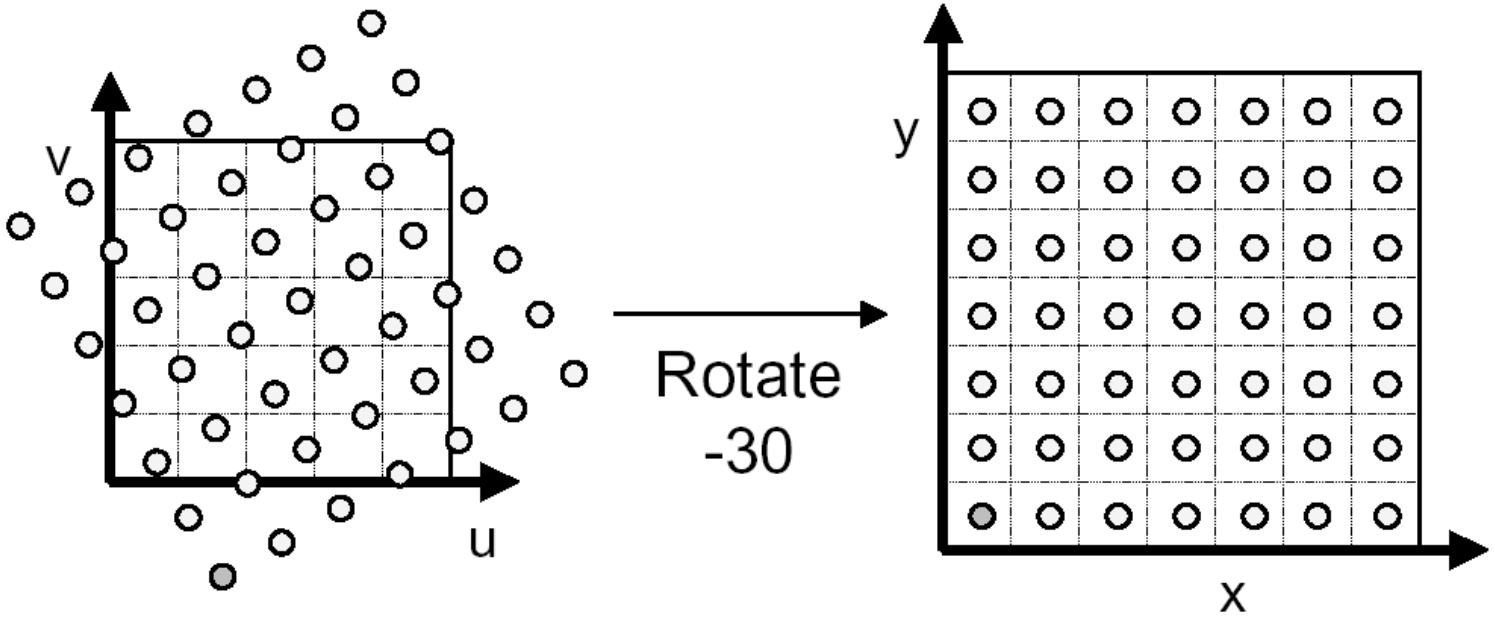
- Reverse mapping:

```
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = f_x^-1(x,y);
    float v = f_y^-1(x,y);
    dst(x,y) = src(u,v);
  }
}
```



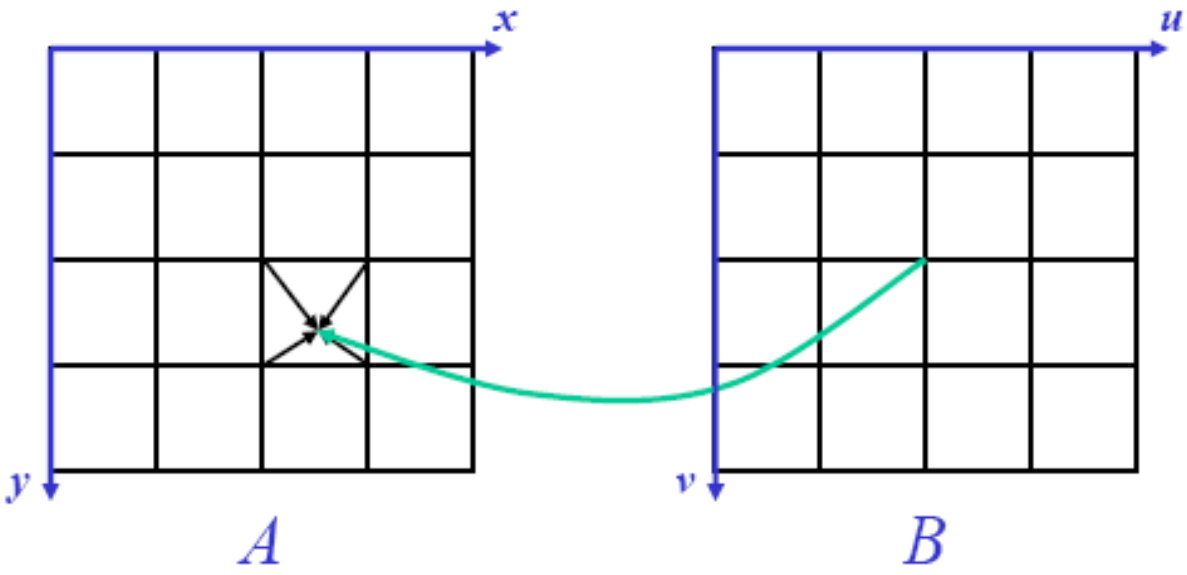
Reverse Mapping

- Iterate over destination image
  - Must resample source
  - May oversample, but much simpler!



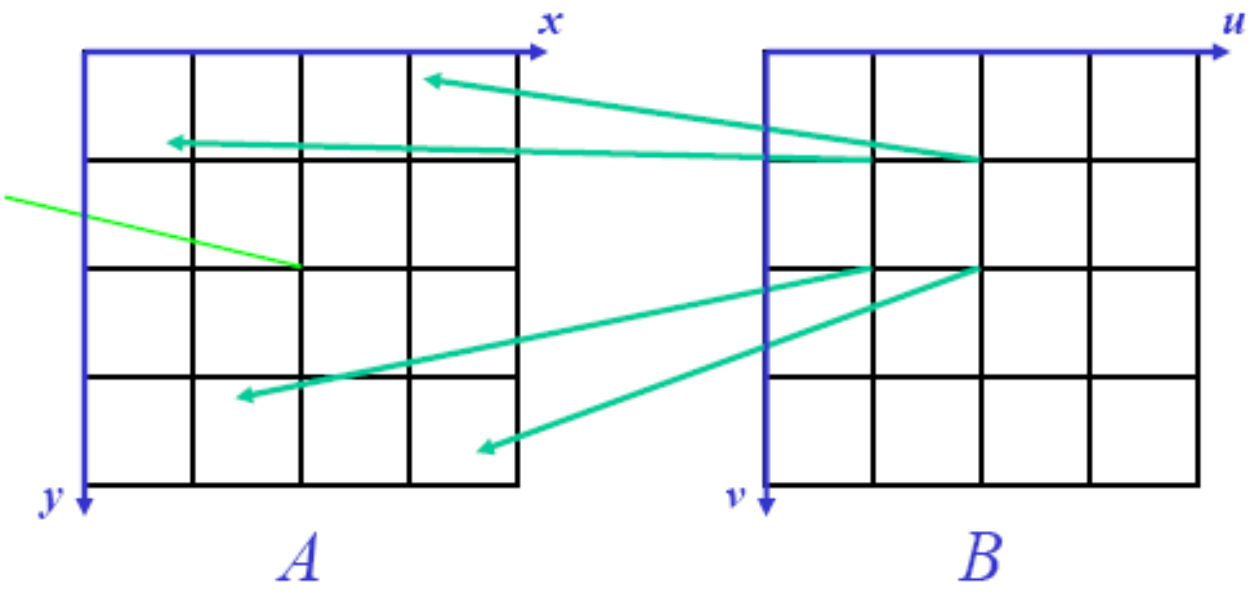
Backward Mapping: Problems

- Doesn't always map *from* a pixel
- Solution: Interpolate between pixels



Backward Mapping: Problems

- May produce holes in the input
- Solution: reduce input image (by averaging pixels) and sample reduced/averaged image → MIP-maps



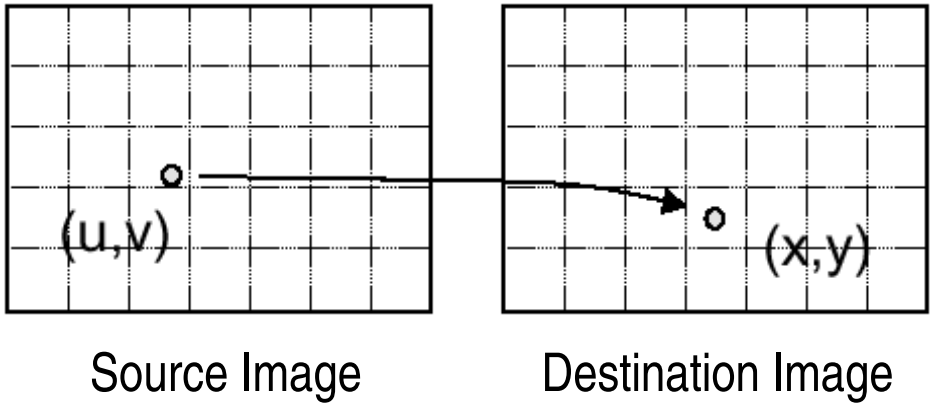
Forward and Reverse Mapping

- Forward → Some pixels in the destination might not get painted, and would have to be interpolated.

- Reverse → Every pixel in the destination image gets set to something appropriate.

Resampling

- Evaluate source image at arbitrary (u, v)
  - (u, v) does not usually have integer coordinates
- Some kinds of resampling
  - Point resampling
  - Triangle filter
  - Gaussian filter



Point Sampling

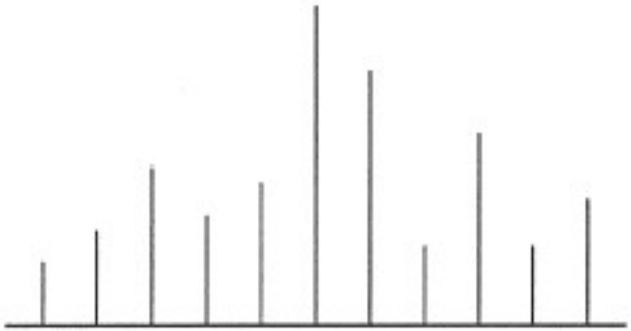
- Take value at closest pixel

```
int iu = trunc(u + 0.5);
int iv = trunc(v + 0.5);
dst(x, y) = src(iu, iv);
```

- Simple, but causes aliasing

Triangle Filter

- Convolve with triangle filter



Input

Output

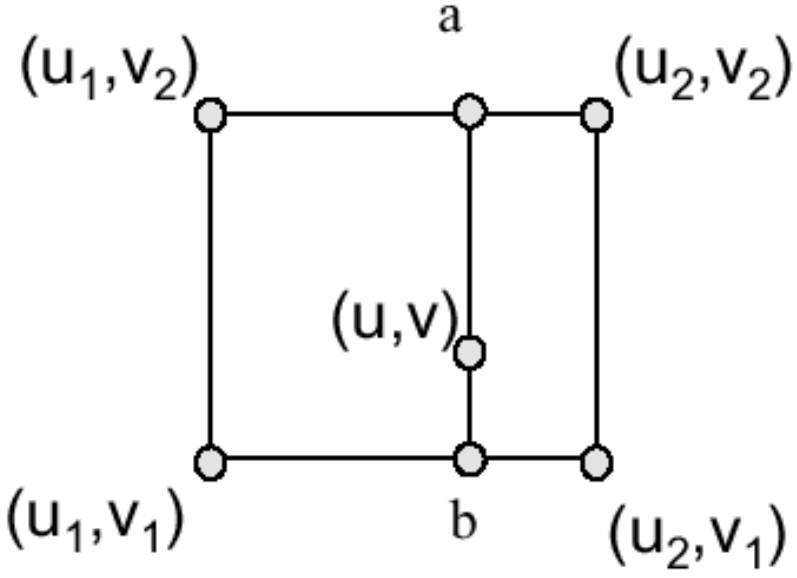
Triangle Filter

- Bilinearly interpolate four closest pixels

  a = linear interpolation of src(u_1, v_2) and src(u_2, v_2)

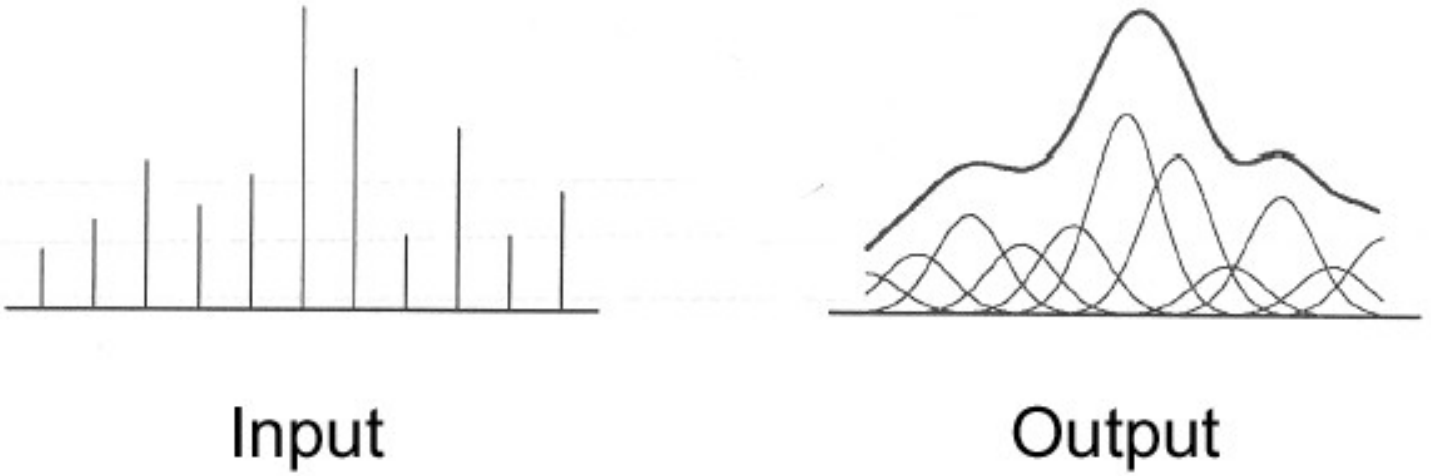
  b = linear interpolation of src(u_1, v_1) and src(u_2, v_1)

  dst(x, y) = linear interpolation of ‘a’ and ‘b’



Gaussian Filter

- Convolve with Gaussian filter



Width of Gaussian kernel affects bluriness

Filtering Method Comparison

- Trade-offs
  - Aliasing versus blurring
  - Computation speed

Image Warping Implementation

- Reverse mapping:

```
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = f_x^-1(x,y);
    float v = f_y^-1(x,y);
    dst(x,y) = resample_src(u,v,w);
  }
}
```

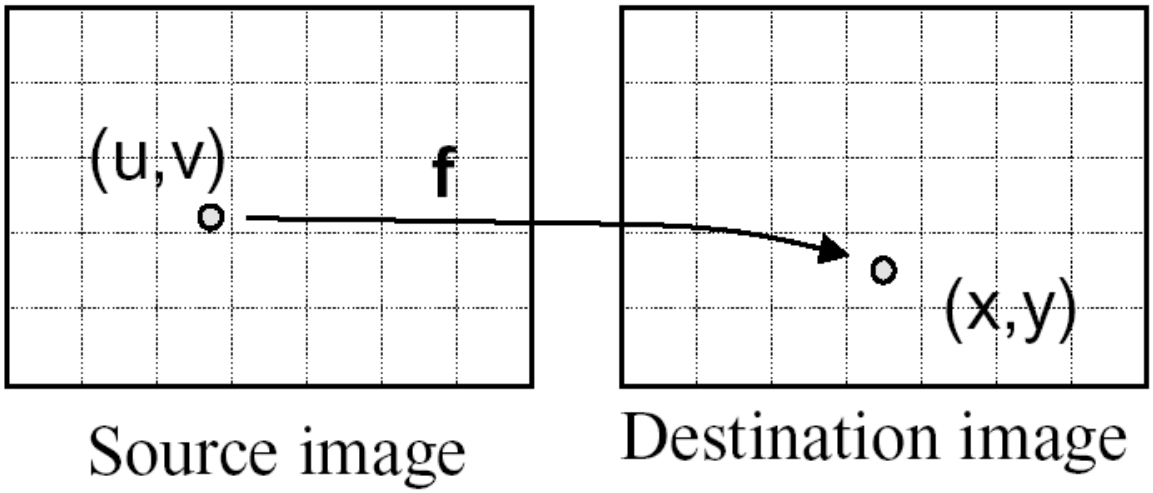
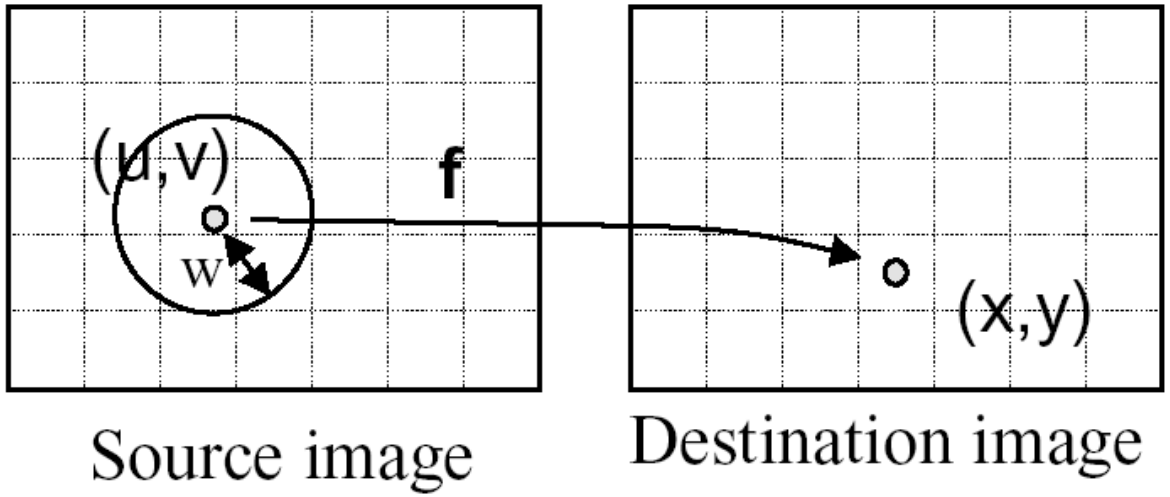


Image Warping Implementation

- Reverse mapping:

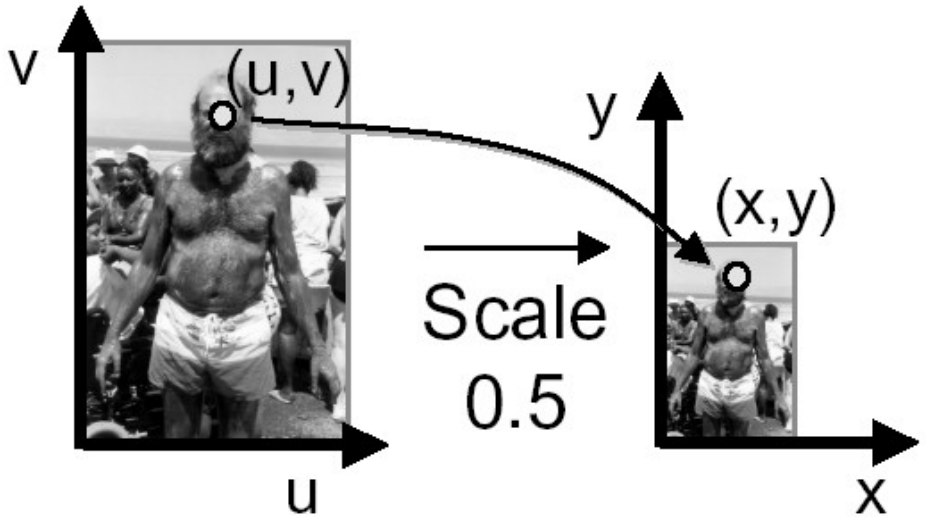
```
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = f_x^-1(x,y);
    float v = f_y^-1(x,y);
    dst(x,y) = resample_src(u,v,w);
  }
}
```



Example: Scale

- 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);
  }
}
```



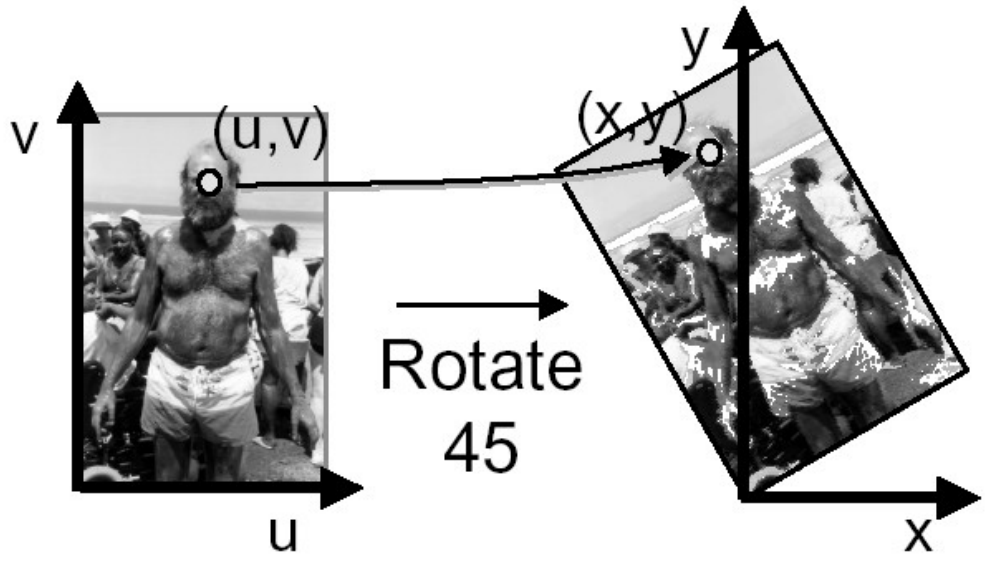
Example: Rotate

- 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 u = x*sin(-Θ) + y*cos(-Θ);
    dst(x,y) = resample_src(u,v,w);
  }
}
```

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

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



Example: Swirl

- 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);
  }
}
```

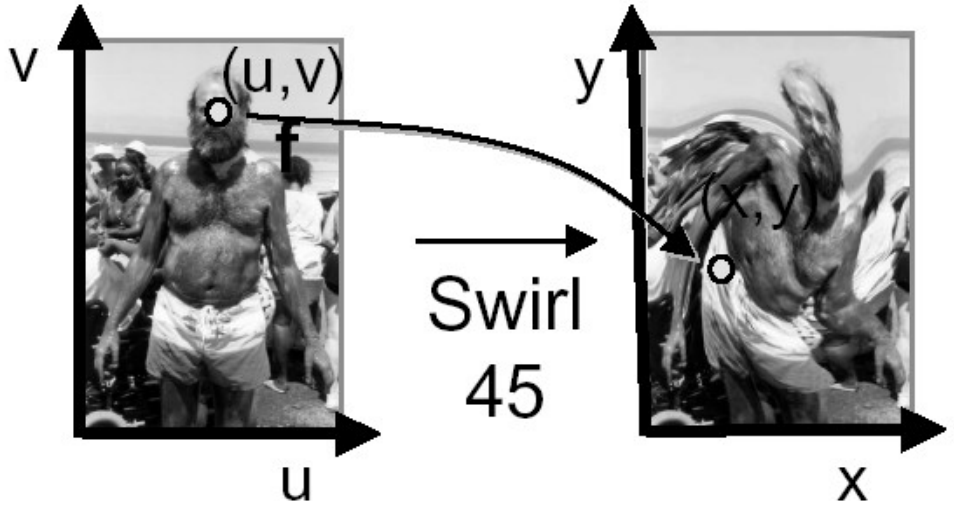


Image Warping: Summary

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

Reverse mapping
is simpler to implement

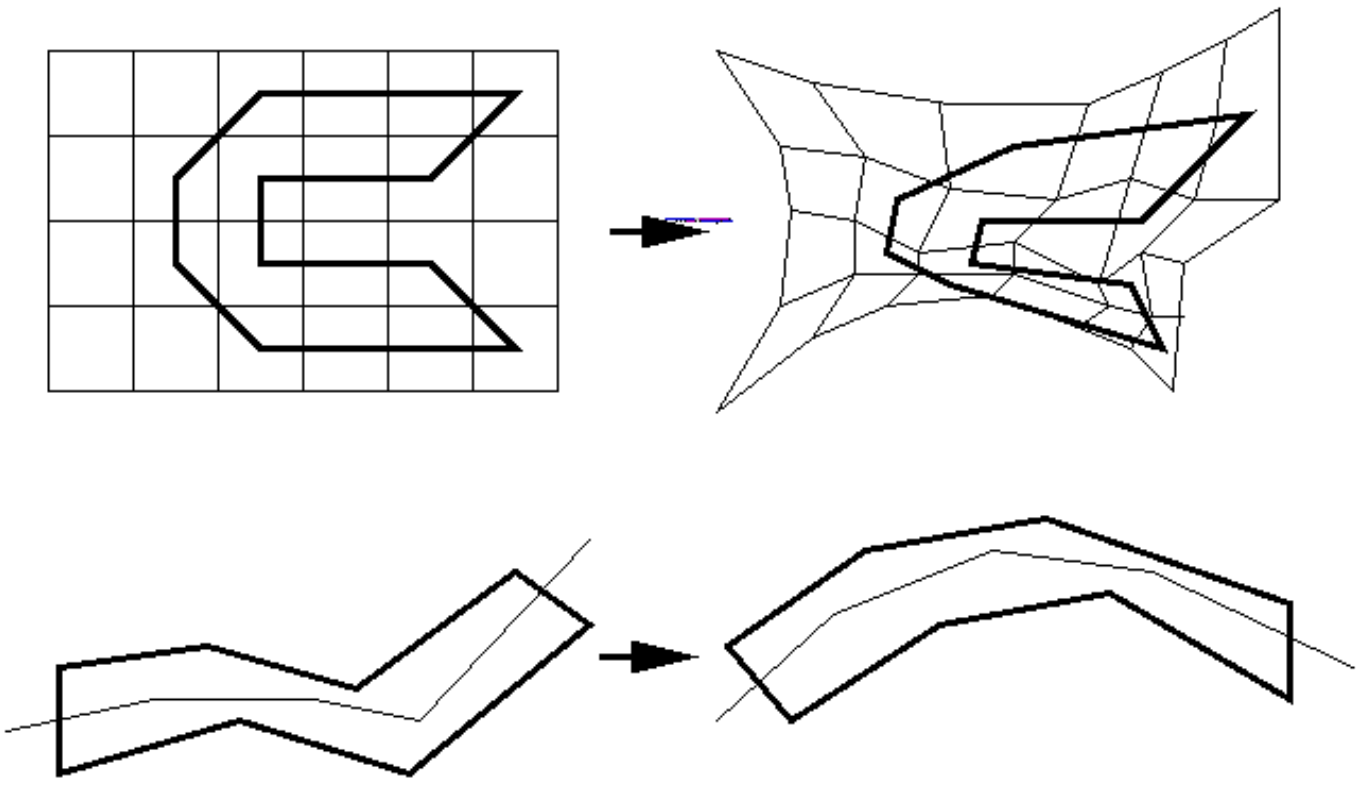
Different filters trade-off
speed and aliasing/blurring

Fun and creative warps
are easy to implement!

Forward and Reverse Mapping

- In either case, the problem is to determine the way in which the pixels in one image should be mapped to the pixels in the other image.
- So, we need to specify **how each pixel moves** between the two images.
- This could be done by specifying the mapping for a few important pixels.

Two Dimensional Object Warping

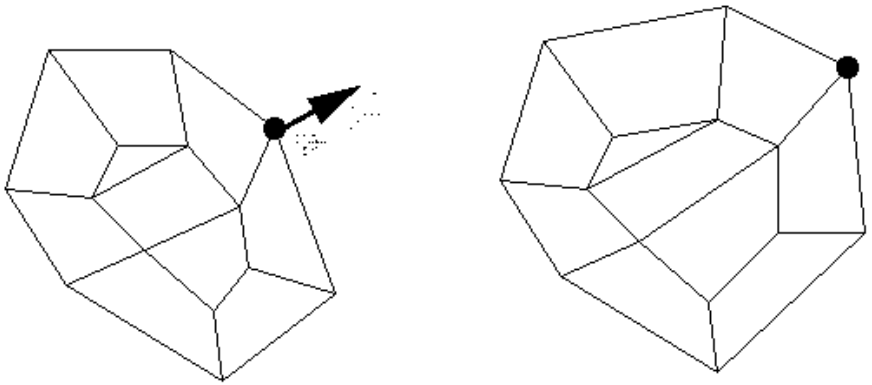


Two Dimensional Object Warping

- The shape modification can also be performed on a vertex-basis instead of on a space-basis.
- A displacement for a seed vertex can be specified by the user and this displacement can be propagated to nearby vertices.
- The displacement can be attenuated as a function of the distance that the vertex to be displaced is away from the seed vertex.

Two Dimensional Object Warping

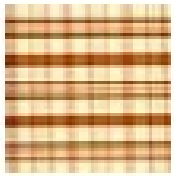
- The **distance function** can be chosen to trade-off quality of results and computational complexity.
  - Minimum number of edges connecting the vertex to be displaced from the seed vertex.
  - Minimum distance traveled over the surface of the object to get from the vertex to be displaced to the seed vertex.
  - Power functions to control the amount of attenuation.
  - The user could select the maximum distance at which the displacement would have an affect.



Texture Mapping

The process of "pasting" a 2D image on a 3D object.

When viewed after projection, this is just a 2D warp of the original texture with hidden surface elimination.



texture map

texture mapped object

Texture mapping