Chapter 9

Image Warps

The word warp brings mental images of large shape distortions, but the technical term *image warp* is used to refer to any transformation on an image that changes any of its geometric properties. Thus a transformation that simply changes the size of an image and a transformation that makes complex twists and bends to an image are both referred to as warps. Other image operations, even ones that make drastic changes, are not referred to as warps as long as they only change non-geometric attributes such as color, texture, graininess, etc.

We often refer to the operation that does the warp as a map or function that sends an input image into a warped output image. To nail this idea down more precisely, let us start by measuring the input image in coordinates (u, v), and our resulting warped image in coordinates (x, y). Then any warp can be thought of as a mapping that sends each coordinate pair (u, v) into a corresponding pair (x, y). Any such map can be expressed as two functions, X that determines the transformed x coordinate from (u, v), and Y that determines the transformed y coordinate from (u, v). This concept is illustrated in Figure 9.1, and is written mathematically as

$$x = X(u, v),$$
$$y = Y(u, v),$$

or in vector notation as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X(u,v) \\ Y(u,v) \end{bmatrix}. \tag{9.1}$$

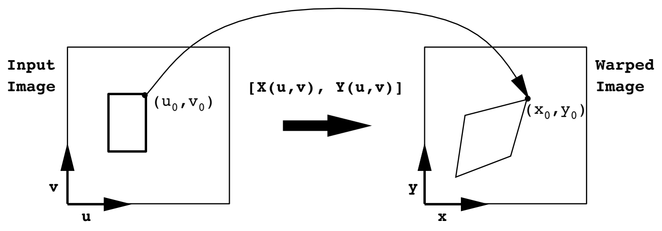


Figure 9.1: Image Map defined by functions $X(.,.)$ and $Y(.,.)$

9.1 Forward Map

The function pair $[X(), Y()]$ defines what is known as a *forward map* from the input image to output image. If the input image were continuous, and the forward map were reasonably smooth (i.e. it had no discontinuities), it would suffice to allow one to "paint in" the warped image from the input image. For each coordinate pair (u, v) in the input image, we would simply color the point $[X(u, v), Y(u, v)]$ in the output image the same color as the point (u, v) in the input image. However, if you reflect a bit you will see that this process is only of hypothetical interest, as it implies that we paint an infinitely finely represented image into another infinitely finely represented image using an arbitrarily large number of painting steps – *it works great but it takes forever!*

Realistically, digital images are not infinitely fine continuous representations but in fact consist of a finite number of discrete samples. We view an image by spreading the color of each sample over a small area – so that each sample corresponds with a pixel when we view the image. This seems to solve our problem, and allows us to write the simple algorithm

```
for(v = 0; v < in_height; v++)
  for(u = 0; u < in_width; u++)
    Out[round(X(u,v))][round(Y(u,v))] = In[u][v];
```

that will perform the forward map with only one operation per pixel in the input image. Note that the `round()` operations in the assignment statement are necessary to provide integer pixel coordinates. Unfortunately, the application of this simple forward map algorithm to a sampled image will result in the kinds of situations shown in Figure 9.2. The output image will be left with "holes" (i.e. pixels with unknown values) where the output is scaled up compared with the input, and multiple pixel overlaps where the output is scaled down with respect to the input.

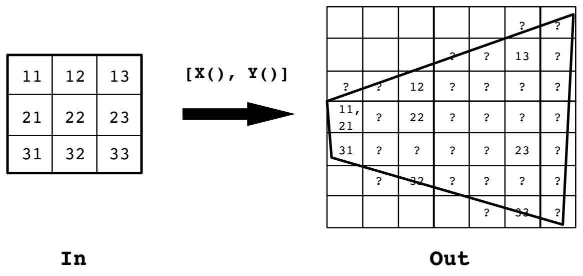


Figure 9.2: Forward Map Leaves Holes and Overlaps

The problem is that each pixel in the input image represents a finite (non-zero) area and actually projects an area of the input image onto the output, as shown in Figure 9.3. If our forward mapping algorithm carefully projected each pixel onto the output, we would see that each input pixel might fully cover some output pixels and partially cover several other output pixels, or it might only partially cover a single pixel. If we did this kind of projection, saving the percentage coverage of each output pixel by the input pixels, we could use these percentages to compute a weighted average color for each output pixel. This would give us an output image without holes or overlaps. But, the calculation of the correct percentages could get very complex, especially if the forward map is highly nonlinear (i.e. straight lines get mapped to curved lines). If the map is linear, a simple way to do the projection of a pixel is to project each of its four corners and then connect these projected corners by straight lines. However, if the mapping is non-linear, we would have to resort to other methods to get accurate results, such as mapping many sample points around the pixel contour. No matter what, this method is bound to be slow to compute and much harder to implement than our original simple pixel-to-pixel algorithm.

9.2 Inverse Map

The problems with the forward image mapping process can be solved by simply inverting the problem, turning it into an *inverse map.* Instead of sending each input pixel to an output pixel, we look at each output pixel and determine what input pixel(s) map to it, as shown in Figure 9.4. To do this we need to invert the mapping functions X and Y. We will name these inverse mapping functions $U(x, y)$ and $V(x, y)$, and define them as

$$u = U(x, y),$$

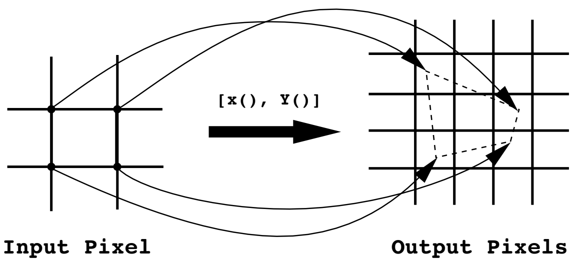


Figure 9.3: Projection of pixel area onto output raster

$$v = V(x, y),$$

or in vector notation as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} U(x,y) \\ V(x,y) \end{bmatrix}, \tag{9.2}$$

with U and V chosen such that

$$\begin{aligned} u &= U[X(u,v), Y(u,v)], \\ v &= V[X(u,v), Y(u,v)]. \end{aligned} \tag{9.3}$$

In other words, the pair $[U(), V()]$ is chosen such that it exactly inverts or "undoes" the mapping due to the pair $[X(), Y()]$. Thus, if one samples the output image at point (x, y), then $(u, v) = [U(x, y), V(x, y)]$ gives the coordinates (u, v) in the input image that map to position (x, y) in the output. As in the forward map, we have to round to give integer pixel coordinates, but here we round the coordinates into the input image, not the output image. Thus, even with a sampled digital image, we can compute our output image from the input image using the inverse map, without fear of holes or overlaps in the output image. The mapping algorithm is simply

```
for(y = 0; y < out_height; y++)
  for(x = 0; x < out_width; x++)
    Out[x][y] = In[round(U(x,y))][round(V(x,y))];
```

The inverse mapping method is simple and fast to compute, and thus is used extensively in most image warping tasks such as morphing and 3D texture mapping. It also has some nice features, such as the ability to sample through a mask to provide automatic clipping of the output image, as shown in Figure 9.5.

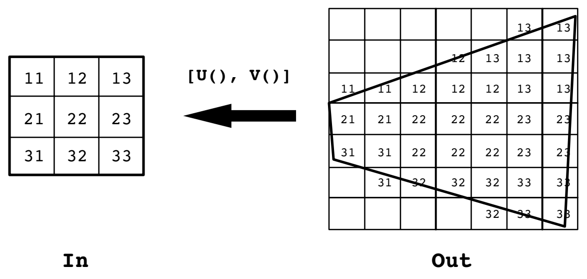


Figure 9.4: Inverse Map gives Complete Covering

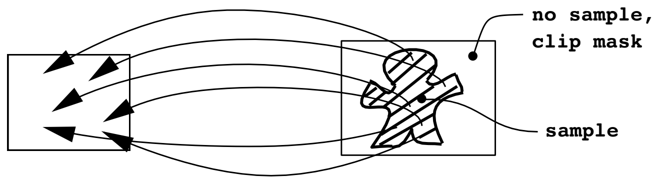


Figure 9.5: Inverse Mapping Through a Clip Mask

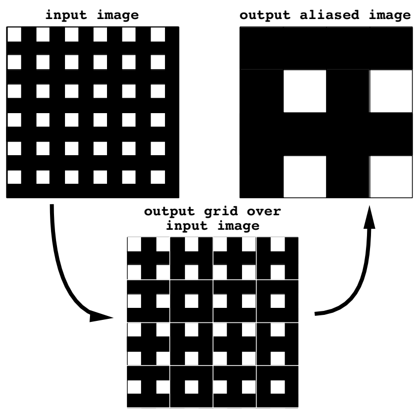
9.3 Warping Artifacts

However, nothing comes for free and in fact the inverse mapping process, while solving the serious problems of the forward map, still leaves us with artifacts that can degrade output image quality. These artifacts are due to the fact that in dealing with digital images we are dealing with sampled, not continuous, data. We will treat this issue in detail later, but for now it is enough to know that they fall into two important categories:

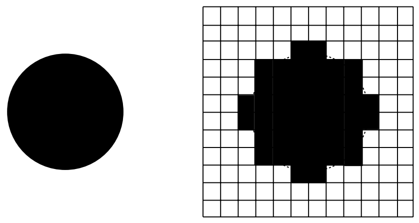
- **aliasing artifacts** caused by sampling the input too coarsely in some places, leading to missed features and unwanted patterning (Figure 9.6a) – aliasing will occur where a region of the output image is *minified* (scaled down) with respect to the corresponding region of the input image.

- **reconstruction artifacts** caused by using too crude a method (psf) to reconstruct the area around the input pixel, leading to staircasing or "jaggies" in the output (Figure 9.6b) – reconstruction artifacts will appear where a region of the output image is *magnified* (scaled up) with respect to the corresponding region of the input image.

We can minimize aliasing by doing a careful job of smoothing the input image before we sample it to produce the output image. We can minimize reconstruction artifacts by using a more sophisticated reconstruction scheme than the square flat areas that we get with pixels. However, these issues will require a careful treatment and we will not begin to build up the necessary tools until the next chapter.



a) aliasing artifacts



b) reconstruction artifacts

Figure 9.6: Aliasing and Reconstruction Artifacts

~~9.4 Affine Maps or Warps~~

~~An affine map on an image is of the general form~~

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}u + a_{12}v + a_{13} \\ a_{21}u + a_{22}v + a_{23} \end{bmatrix}.$$