Chapter 16

Avoiding Artifacts in Warped Images

16.1 The Warping Process

Figure 16.1 depicts a typical image warp, and shows how the same warp can produce both maginification and minification in the output image. Where the warped image is magnified compared with the input, the artifacts are the result of oversampling, i.e. too many output pixels depend on a single input pixel. Where the warped image is minified compared with the input, the artifacts are aliasing errors due to undersampling, i.e. pixels in the input image are skipped because there are more input pixels than there are output pixels. In order to understand what kind of errors are introduced by magnification and minification, we need to have a sound conceptual view of what we are doing when we warp a digital image.

16.2 Sampling and Reconstruction

First, remember that the original image consists of samples of some real world (or virtual) scene. Each pixel value is simply a point sample of the scene. When we view such an image on a screen, we are really viewing a reconstruction that is done by spreading each sample out over the rectangular area of a pixel, and then tiling the image with these rectangles. Figures 16.2a-c show the steps in the process from sampling to reconstruction, looked at in one-dimensional form. When viewed from a distance, or when the scene is sufficiently smooth, the

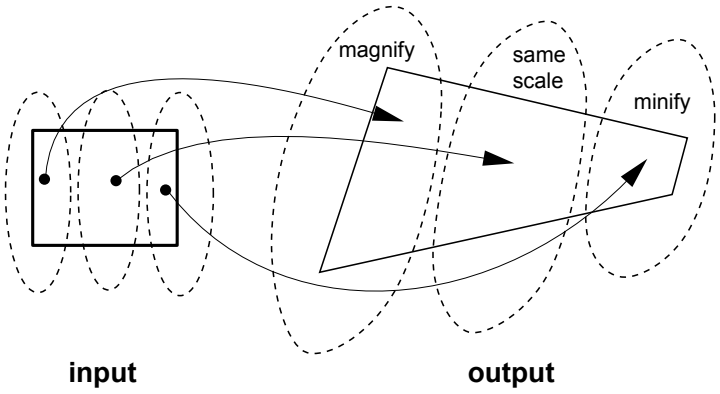


Figure 16.1: Magnification and Minification in the Same Warp

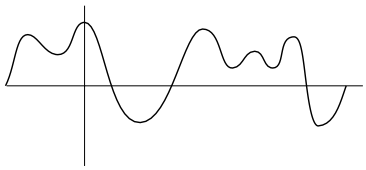
rectangular artifacts in the reconstruction are negligible. However, under magnification they become very noticeable as *jaggies* and *blockiness* in the output. Figure 16.2d shows how the extra resampling when magnifying results in multiple samples with the same value, effectively spreading single pixel values in the original image over multiple pixels in the magnified image, creating a jagged or blocky look.

> **Lesson 1**: To reduce magnification artifacts we need to do a better job of reconstruction.

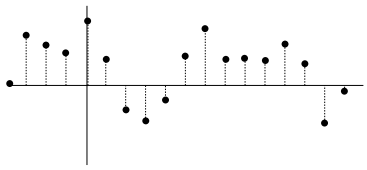
16.3 Resampling and Aliasing

When we minify an image, however, the resampling process causes us to miss many of the reconstructed samples in the original image, leading to missing details, *ropiness* of fine lines, and other aliasing artifacts which will appear as patterning in the output. In the worst case, the result can look amazingly unlike the original, like the undersampled reconstruction shown in Figure 16.2e. The problem here is that the resampling is being done too coarsely to pick up all the detail in the reconstructed image, and worse, the regular sampling can often pick up high frequency patterns in the original image and reintroduce them as low frequency patterns in the output image.

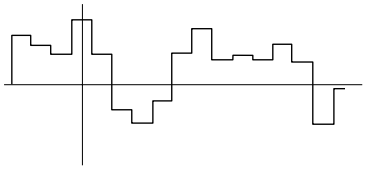
> **Lesson 2**: To reduce minification artifacts we must either 1) sample more finely than once for each output pixel, or 2) smooth the reconstructed input before sampling.



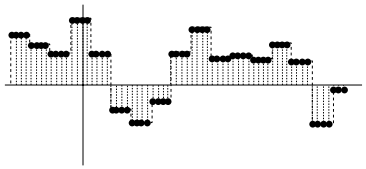
a) brightness along a scanline across the original scene



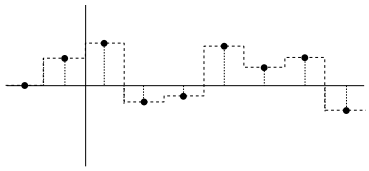
b) brightness samples along the same scanline



c) pixel-like reconstruction of original line from the samples



d) resampling under magnification



e) resampling under minification

Figure 16.2: The Sampling, Reconstruction and Resampling Processes

16.4 Sampling and Filtering

Although we have not yet developed the mathematics to state this precisely, it can be shown that under certain conditions it is theoretically possible to exactly recover an original unsampled image given only the samples in the pixmap. It will be possible to do this when the original image is smooth enough, given the sampling density that we used to sample the original image to capture it in the pixmap. The criterion, loosely stated, is that the original image should not have any fluctuations that occur at a rate greater than 1/2 of the sampling rate. In other words, if the original image has an intensity change that from dark to light or light to dark there should be at least two samples taken in the period of this fluctuation. If we have more samples, that is even better. Another way of saying this is that any single dark to light or light to dark transition should be wide enough that at least two samples are taken in the transition. This criterion must hold over the whole image, for the shortest transition period in the image.

If we let the sampling period (i.e. the space between samples) be T_S, and the resampling period be T_R, then in considering the image warping problem:

1. A perfect reconstruction could be obtained by prefiltering the sampled image with a filter that smooths out all fluctuations that occur in a space smaller than $2T_S$.

2. Aliasing under resampling could be avoided by filtering the reconstructed image to remove all fluctuations that occur in a space smaller than $2T_R$, before doing the resampling.

In simple terms, we need to: 1) Do a nice smooth reconstruction, not the crude one obtained by spreading samples over a pixel area, and 2) possibly further smooth the reconstruction so that when we resample, we are always doing the resampling finely enough to pick up all remaining detail.

16.5 The Warping Pipeline

Conceptually, what we are trying to do when we warp a digital image is the process diagrammed in Figure 16.3. It consists of three steps:

1. reconstruct a continuous image from the sampled input image, by filtering out all fluctuations with period less than twice the original sampling period T_S,
2. warp the continuous reconstruction into a new shape,
3. filter the warped image to eliminate fluctuations with period less than twice resampling period to be used in the next step, and
4. resample the warped image with a resampling period T_R to form a new discrete digital image.

However, in fact we do not really have any way in the computer of reconstructing and manipulating a continuous image, since by its nature everything in the computer is discrete. Now, since the reconstructed and warped continuous versions of the image never really get constructed, it makes sense to think of combining the reconstruction and low-pass filtering operations of steps 1 and 3 into a single step, that somehow would take into account the warp done in step 2. To understand just how this might work, it would help to have a view of the filtering process that would operate over the original *spatial domain* image.

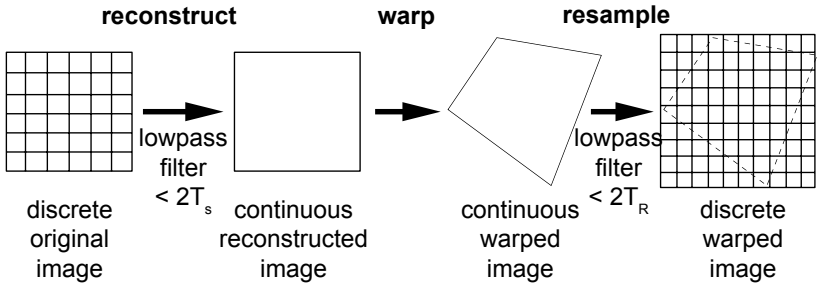


Figure 16.3: A Conceptual View of Digital Image Warping

16.6 Antialiasing: Spatially Varying Filtering and Sampling

16.6.1 The Aliasing Problem

Remember that in the image warping problem there are two places where filtering is important:

1. reconstruction filtering (critical when magnifying)
2. antialiasing filtering (critical when minifying).

In the last section we looked at the general idea of practical filtering, with special attention paid to the reconstruction problem. Here we will focus specifically on the aliasing problem.

Recall that aliasing will occur when resampling an image whose warped reconstruction has frequency content above 1/2 the resampling frequency. To prevent this we can either:

1. filter the reconstructed and warped image so that its highest frequencies are below 1/2 the resampling rate,
2. adjust the resampling rate to be at least twice the highest frequency in the reconstructed warped image.

In general, the problem In both cases is that a warp is non-uniform, meaning that the highest frequency will vary across the image, as schematized in Figure 16.4.

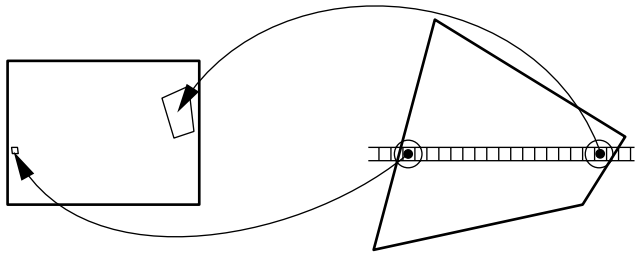


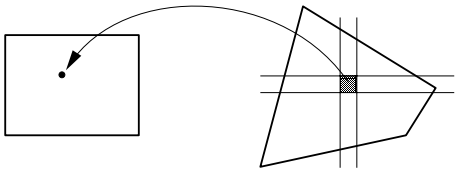
Figure 16.4: Area Sampling, Pixel Area Projected Back Into Input Image

We can deal with this by filtering the entire image to reduce the highest frequencies found anywhere in the image, or sample everywhere at a rate twice the highest frequency found anywhere in the image. Either solution can be very wasteful (sometimes horribly wasteful) of computation time in those areas of the image that do not require the heavy filtering or excess sampling. The answer is to use either spatially varying filtering or adaptive sampling.

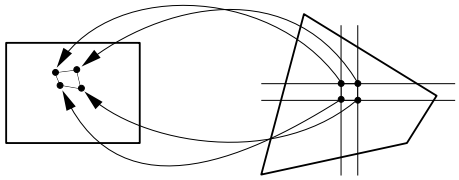
16.6.2 Adaptive Sampling

We can think of the inverse mapping process described in Chapter 7, and shown in Figure 16.5a, as a point sampling of the input image, over the inverse map, guided by a uniform traversal over the output image. This technique is simple to implement but, as we have discovered, leaves many artifacts, and we need to look at more powerful approaches.

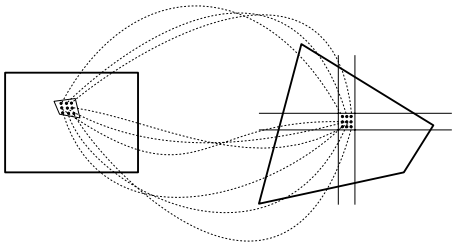
One method for doing sampling without aliasing is *area sampling*. This technique is diagrammed in Figure 16.5b. Area sampling projects the area of an output pixel back into the input image and attempts to take a weighted average of covered and partially covered pixels in the input. Area sampling produces good results under minification, i.e. aliasing is eliminated. However, to do area sampling accurately is very time consuming to compute and hard to implement. Fortunately, experience shows that good success can be had with simpler, less expensive approximations to area sampling.



a) point sampling



b) area sampling



c) supersampling

Figure 16.5: Sampling Techniques Under Inverse Mapping

A common approximation to area sampling is known as *supersampling*. Here,

we still take only point samples, but instead of sampling the input only once, or over an entire area as with area sampling, multiple samples are taken per output pixel. This process is diagrammed in Figure 16.5c. A sum or weighted average of all of these samples is computed before storing in the output pixel. This gives results that can be nearly as good as with area sampling, but results vary with the amount of minification. The problem with this method is that if you take enough samples to eliminate aliasing everywhere, you will be wasting calculations over parts of the image that do not need to be sampled so finely. Although supersampling can be wasteful, some economies can be had. Figure 16.6 shows how some careful bookkeeping can be done to minimize the number of extra samples per pixel. If samples are shared at the corners and along the edges of pixels, there is no need to recompute these samples when advancing to a new pixel.

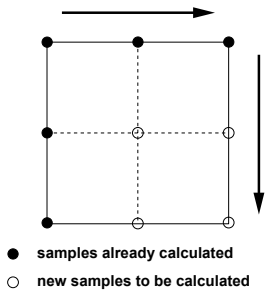


Figure 16.6: Sharing of Samples Under Inverse Mapping. Scan is Left to Right, Top to Bottom

Adaptive supersampling improves on simple supersampling by attempting to adjust the sampling density for each pixel to the density needed to avoid aliasing. The process used in adaptive supersampling is as follows:

1. Do a small number of test samples for a pixel, and look at the maximum difference between the samples and their average.

2. If any samples differ from the average by more than some predefined threshold value, subdivide the pixel and repeat the process for each subpixel that has an extreme sample.

3. Continue until all samples are within the threshold for their sub-pixels(s) or some maximum allowable pixel subdivision is reached.

4. Compute the output pixel value as an area-weighted average of the samples collected above.

Any regular sampling approach will be susceptible to aliasing artifacts introduced by the regular repeating sampling pattern. A final refinement to the adaptive supersampling technique is to *jitter* the samples so that they no longer are done on a regular grid. As shown in Figure 16.8, an adaptive sampling

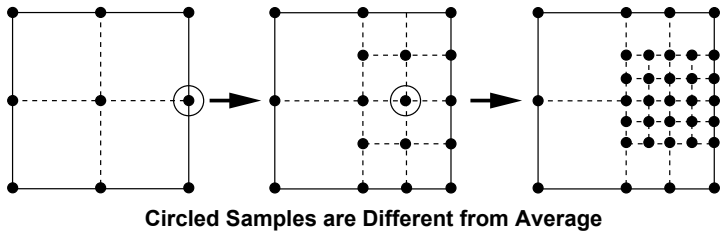


Figure 16.7: Adaptive Supersampling

approach is taken but each sample is moved slightly by a random perturbation from its regular position within the output pixel before the inverse map is calculated. Irregular sampling effectively replaces the low frequency patterning artifacts that arise from regular sampling with high frequency *noise*. This noise is usually seen as a kind of graininess in the image and is usually much less visually objectionable than the regular patterning from aliasing.

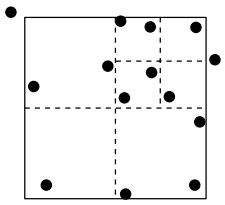


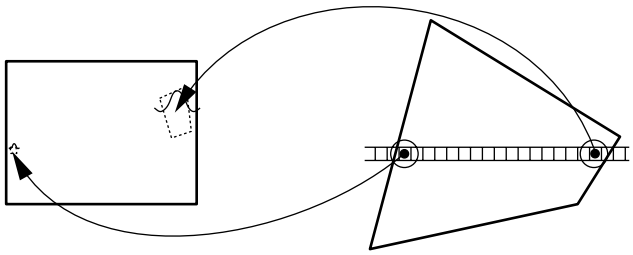
Figure 16.8: Jittered Adaptive Supersampling

16.6.3 Spatially Variable Filtering

Antialiasing via filtering is a fundamentally different approach from antialiasing via adaptive supersampling, and leads to another basic problem. Since the frequency content of an image will vary due to the warp applied, efficiency issues require that whatever filtering is done be adapted to this changing frequency content. The idea behind spatially varying filtering is to filter a region of a picture only as much as needed, varying the size of the filter convolution kernel with the variance in frequency content.

Spatially variable filtering schemes attempt to pre-filter each area of the input picture, using only the level of filtering necessary to remove frequencies above either 1/2 the sampling rate or 1/2 the resampling rate, whichever is lower under the inverse map. The idea here is that the same convolution kernel is used across the whole image but it changes scale to accommodate the resampling rate. Where the output is minified, the scale of the kernel being convolved with the input image is increased. Where the output is magnified, it is decreased. Re-

member, of course, that the bigger the kernel the more computation is involved, so the problem with this approach is that it gets very slow over minification.



Kernel size expands to match sampling rate. For high sampling rate, kernel size is small, for low rate it is large.

Figure 16.9: Spatially Varying Filtering

A crude but efficient way of implementing spatially varying filtering is the *summed area table* scheme. It is crude in that it only allows for a box convolution kernel, but fast in that it does a 128×128 convolution as fast as a 4×4 convolution! The trick is to first transform the input image into a data structure called a summed area table (SAT). The SAT is a rectangular array that has one cell per pixel of the input image. Each cell in the SAT corresponds spatially with a pixel in the input, and contains the sum of the color primary values in that pixel and all other pixels below and to the left of it. An example of a 4×4 SAT is shown in Figure 16.10.

2	1	3	2
1	3	2	1
3	2	1	1
1	2	1	3

input image

$\Longrightarrow$

7	15	22	29
5	12	16	21
4	8	10	14
1	3	4	7

summed area table

Figure 16.10: Summed Area Table

The SAT can be computed very efficiently by first computing its left column and bottom row, and then traversing the remaining scan lines doing the following calculation:

```
SAT[row][col] = Image[row][col] + SAT[row-1][col] +
                SAT[row][col-1] - SAT[row-1][col-1];
```

where `SAT` is the summed area table array and `Image` is the input image pixmap, and array indexing out of array bounds is taken to yield a result of 0.

Once the SAT is built, we can compute any box-filtered pixel value for any integer sized box filter with just 2 subtracts, 1 add and 1 divide. The computation of the convolution for one pixel is

```
BoxAvg = (SAT[row][col] - SAT[row-w][col] -
          SAT[row][col-w] + SAT[row-w][col-w]) / ^2;
```

where `w` is the desired convolution kernel width and `(row, col)` are the coordinates of the upper right hand corner of the filter kernel. Note, that the size `w` of the convolution kernel has no effect on the time to compute the convolution. The required size of the convolution can be approximated by back projecting a pixel and getting the size of its bounding rectangle in the input, as shown in Figure 16.11.

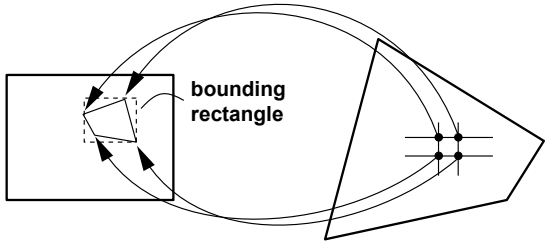


Figure 16.11: Finding the Input Bounding Rectangle for an Output Pixel

Looked at mathematically, if the inverse map is given by functions $u(x, y)$ and $v(x, y)$, then the bounding rectangle size is given by

$$du = \max(\partial u/\partial x, \partial u/\partial y)$$

in the horizontal direction, and

$$dv = \max(\partial v/\partial x, \partial v/\partial y)$$

in the vertical direction, as shown in Figure 16.12. Alternatively, the euclidian distances

$$(du)^2 = (\partial u/\partial x)^2 + (\partial u/\partial y)^2$$

and

$$(dv)^2 = (\partial v/\partial x)^2 + (\partial v/\partial y)^2$$

can be used.

16.6.4 MIP Maps and Pyramid Schemes

A final and very popular technique for speeding the calculation of a varying filter kernel size is to use a Multiresolution Image Pyramid or *MIP-Map* scheme for

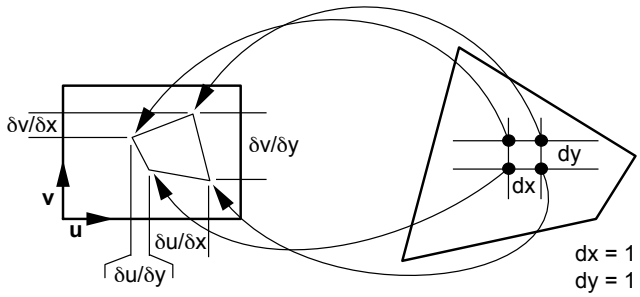


Figure 16.12: Measuring the Required Kernel Size Under an Inverse Map

storing the image. The idea here is to get quick access to the image prefiltered to any desired resolution by storing the image and sequence of minified versions of the image in a pyramid-like structure, as diagrammed in Figure 16.13. Each level in pyramid stores a filtered version of the image at 1/2 the scale of the image below it in the pyramid. In the extreme, the top level of the pyramid is simply a single pixel containing the average color value across the entire original image. The pyramid can be stored very neatly in the data structure shown in Figure 16.14, if the image is stored as an array of RGB values.

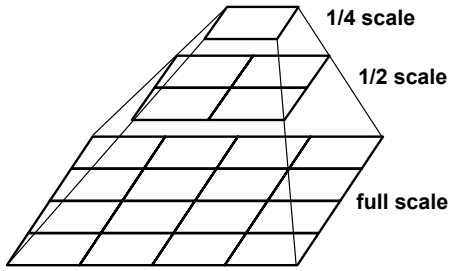


Figure 16.13: An Image Pyramid

The MIP-Map is somewhat costly to compute, so the scheme finds most use in dealing with textures for texture mapping for three-dimensional computer graphics rendering. Here, the MIP-Map can be computed once for a texture and stored in a texture library to be used as needed later. Then when the texture is used, there is very little overhead required to complete filtering operations to any desired scale.

From the MIP-Map itself, it may seem that it is only possible to do magnification or minification to scales that are powers of 2 from the original scale. For example if we want to scale the image to 1/3 its original size, the 1/2 scale image in the MIP-Map will be too fine, and the 1/4 scale image will be too coarse. The solution to this problem is to interpolate across pyramid levels, to get

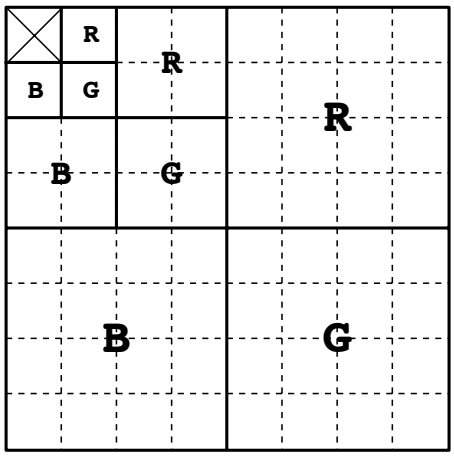


Figure 16.14: Data Structure for a MIP-Map

approximation to any resolution image. We would sample at the levels above and below the desired scale, and then take a weighted average of the color values retrieved. The idea is simply to find the pyramid level where an image pixel is just larger than the projected area of the output pixel, and the level where a pixel is just smaller than this area. Then we compute the pixel coordinates at each level, do a simple point sample at each level, and finally take the weighted average of the two samples.