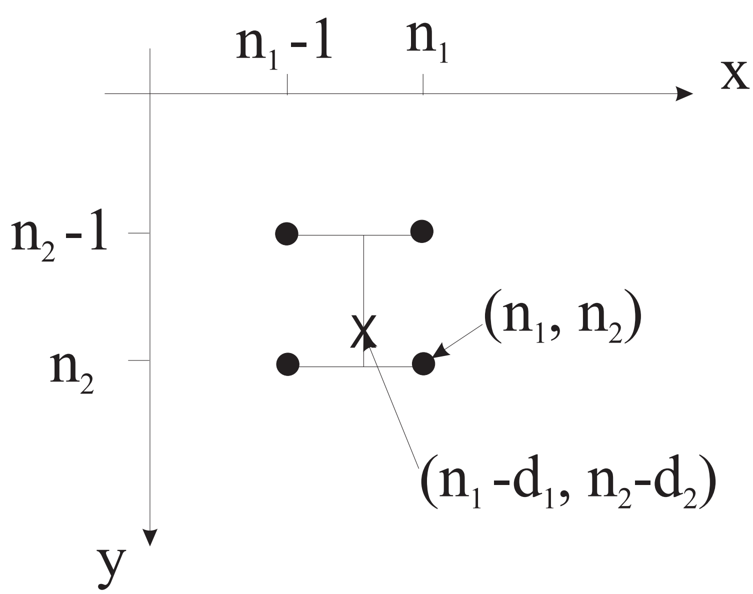
Image Interpolation

Problem: Given a sampled image $I[n_1, n_2]$ we wish to interpolate it at non-integer position, say $I[n_1 - d_1, n_2 - d_2]$, where $\vec{d} = (d_1, d_2)^T$ is a real-valued vector.

Separable Approach: We use a separable filter kernel to reduce 2D interpolation to several 1D interpolation problems.



Reuse: We reuse our 1D interpolation filters for the separable kernels (see upSample.m). The Fourier properties of the 1D kernels carry directly over to this 2D approach.

Matlab: imageTutorial.m demonstrates simple image translations and warps.

Example: Image Translation

Suppose we wish to translate an image $I[n_1, n_2]$ by a (constant) displacement $\vec{d}$. That is, the new image $R[n_1, n_2]$ is

$$R[n_1, n_2] \;=\; I[n_1 - d_1, n_2 - d_2]. \tag{1}$$

We assume the new image R is the same size as I, with pixels for which $(n_1 - d_1, n_2 - d_2)$ is out of bounds in the original image I set to some special value (eg. `nan` in Matlab). (Alternatively, the range for the pixels in R might be enlarged to include the range over which the translated image $I[n_1 - d_1, n_2 - d_2]$ is defined.)

Strictly speaking, $I[n_1, n_2]$ is only defined at integer-valued sample points. The term $I[n_1 - d_1, n_2 - d_2]$ in (1) is short-hand for the sample of some interpolating image $C(x, y)$ at the intermediate location $(n_1 - d_1, n_2 - d_2)$. In general, we could define $C(x, y) = g * I$, where g is some 2D interpolation kernel $g(x, y)$.

We will use a separable interpolation kernel $g(x, y) = g_1(x)g_2(y)$. Here g_k can be any 1D interpolation kernel, such as the triangular filter, or the Catmull-Rom filter considered previously.

Separable Implementation

For a separable kernel, we have

$$C(x,y) = g * I = (g_2 *_y (g_1 *_x I))(x,y) \tag{2}$$

where $*_x$ and $*_y$ denotes convolution in the x and y directions, respectively. Using a temporary image $T(x, n_2)$ we can write (2) in two steps,

$$\begin{aligned} T(x,n_2) &= (g_1 *_x I)(x,n_2) \equiv \sum_{m_1} g_1(x-m_1) I[m_1,n_2], \\ C(x,y) &= (g_2 *_y T)(x,y) \equiv \sum_{m_2} g_2(y-m_2) T(x,m_2). \end{aligned}$$

Evaluating this at the desired point $(x, y) = (n_1 - d_1, n_2 - d_2)$, we find

$$T(n_1-d_1,n_2) = \sum_{m_1} g_1(n_1-d_1-m_1) I[m_1,n_2], \tag{3}$$

$$C(n_1-d_1,n_2-d_2) = \sum_{m_2} g_2(n_2-d_2-m_2) T(n_1-d_1,m_2). \tag{4}$$

Notice equation (4) provides the desired translated image $R[n_1, n_2] \equiv C(n_1 - d_1, n_2 - d_2)$. By defining $\hat{T}[n_1, n_2] \equiv T(n_1 - d_1, n_2)$ we can rewrite equations (3) and (4) as discrete 1D convolutions.

Image Translation by Separable Convolution

Define the discrete filter kernels

$$f_1[j] \;=\; g(j - d_1), \tag{5}$$
$$f_2[k] \;=\; g(k - d_2), \tag{6}$$

for integers j, k, where $g(x)$ is some 1D interpolation kernel (eg. Catmull-Rom). Then equations (3-4) can be written

$$\hat{T}[n_1, n_2] \;=\; f_1 *_x I = \sum_{m_1} f_1(n_1 - m_1) I[m_1, n_2], \tag{7}$$
$$R(n_1, n_2) \;=\; f_2 *_y \hat{T} = \sum_{m_2} f_2(n_2 - m_2) \hat{T}(n_1, m_2). \tag{8}$$

These operations were sketched on page 1. First I is interpolated along each row, and sampled at the intermediate locations $n_1 - d_1$. This forms the temporary image $\hat{T}$. Then this temporary image is interpolated down each column, and sampled at the intermediate locations $n_2 - d_2$.

Away from the image boundaries, if we swap the order of these two convolutions then we obtain the same translated image (why?).

When $g(x)$ is the triangular (piecewise linear) interpolation filter this is called **bilinear** interpolation. For piecewise cubic $g(x)$ it is called **bicubic** interpolation.

Image Warping

More general "rubber sheet" image deformations involve spatial displacements which vary with the image position $\vec{x}$. In particular, suppose $\vec{W}(\vec{x})$ is a warp function which takes a point $\vec{x}$ in the desired image $R(\vec{x})$, to the corresponding point $\vec{x}^{\,w} = \vec{W}(\vec{x})$ in the original image $I(\vec{x}^{\,w})$. Then the warped image is defined by

$$R[\vec{n}] \;=\; I(\vec{W}(\vec{n})), \quad \text{where } \vec{n} = (n_1,\; n_2)^T, \vec{W} = (W_1,\; W_2)^T. \tag{9}$$

In order to evaluate $I(\vec{W}(\vec{n}))$, we again need to use 2D interpolation.

However, due to spatial variation in $\vec{W}(\vec{x})$, the x, y interpolation kernels in (5-6) will in general depend on $\vec{x}$,

$$f_1[j;\vec{x}] \;=\; g(j - (x_1 - W_1(\vec{x}))), \tag{10}$$

$$f_2[k;\vec{x}] \;=\; g(k - (x_2 - W_2(\vec{x}))), \tag{11}$$

Equations (7-8) become

$$\hat{T}[n_1, m_2; \vec{n}] \;=\; \sum_{k_1} f_1[n_1 - k_1; \vec{n}] I[k_1, m_2], \tag{12}$$

$$R[\vec{n}] \;=\; \sum_{m_2} f_2[n_2 - m_2; \vec{n}] \hat{T}(n_1, m_2; \vec{n}). \tag{13}$$

This is **not** simply convolution since the filter kernels depend on the image location $\vec{n}$. Finally, if the warp $\vec{W}(\vec{x})$ involves a substantial shrinkage, then we need to blur $I[\vec{n}]$ before warping to avoid aliasing.