Geometrical Transformations

Linear systems

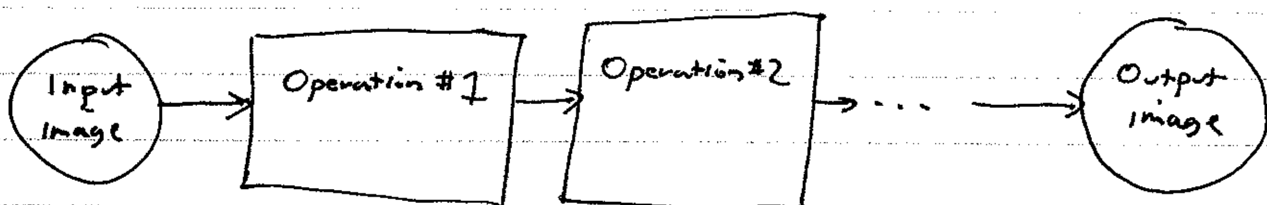
We often use linear systems ideas and concepts to provide a mathematical basis for manipulation of digital images. With it we can derive functions that operate on the numbers comprising our image data to allow us to better use the information contained therein.

A function is <u>linear</u> if it obeys the property

$$f(ax + by) = a \cdot f(x) + b \cdot f(y)$$

This is often expressed as the principle of superposition: the result of applying a linear function $f(\,)$ to the scaled sum of two values is the same as first applying the function to each value, then scaling and adding the result. Knowing that a system is linear helps us in many types of image processing, such as in the development of filters. It also lets us summarize the properties of an imaging system in terms of a single function, the impulse response. We will return to the impulse response when we discuss digital filtering.

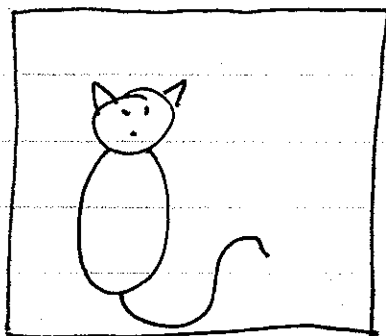
We may model an image processing system (linear or non-linear) as a series of operations



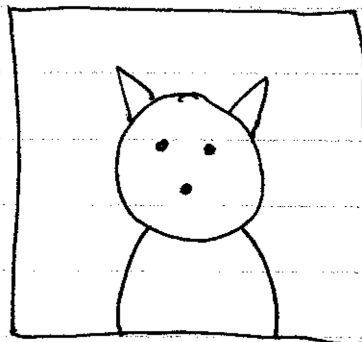
If each operation is linear, we may combine all the steps to one equivalent step using superposition and greatly simplify our model. Since each step requires some computer processing, combining steps also increases computer efficiency.

<u>Some simple transformations</u>

Zooming - When we enlarge a portion or all of an image, we denote the effect <u>zooming</u>, borrowing from the photographic term.



Original Image



Zoomed Portion

Because we have lost some of the original image, we say that the new image has been <u>cropped</u> as well as zoomed.

While it is clear pictorially what we have done here, how do we handle such data in a digitized computer image? Let's look at a small (3x3) subset of a digital image before and after several types of zooms.

Original matrix : $\begin{bmatrix} 8 & 4 & 8 \\ 4 & 8 & 4 \\ 8 & 2 & 8 \end{bmatrix}$

Suppose we simply insert blank rows and columns in between the original values:

$$\begin{bmatrix} 8 & 0 & 4 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 8 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 2 & 0 & 8 \end{bmatrix}$$

This certainly is larger and contains all the information of the original. But it won't produce a nice-looking image as a result. You can see this more easily if we zoom again:

$$\begin{bmatrix} 8 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 8 \end{bmatrix}$$

This image is mostly zero, with a few sparse points shown. One way to get a better picture is to use a <u>zero-order hold</u>, where we copy each value to the right and down with a zero-order function (a constant) based on each value:

Zero-order hold
Zoomed image

$$\begin{bmatrix} 8 & 8 & 4 & 4 & 8 \\ 8 & 8 & 4 & 4 & 8 \\ 4 & 4 & 8 & 8 & 4 \\ 4 & 4 & 8 & 8 & 4 \\ 8 & 8 & 2 & 2 & 8 \end{bmatrix}$$

The zero-order is commonly used as it is easy to compute and doesn't leave holes. But sometimes the very large-appearing pixels (actually resels) are not aesthetic, so we can instead use a <u>first-order hold</u>, which uses a first-order function (a line) to get intermediate values:

first-order hold zoomed image

$$\begin{bmatrix} 8 & 6 & 4 & 6 & 8 \\ 6 & 6 & 6 & 6 & 6 \\ 4 & 6 & 8 & 6 & 4 \\ 6 & 5.5 & 5 & 5.5 & 6 \\ 8 & 5 & 2 & 5 & 8 \end{bmatrix}$$

Note here we had to derive the intermediate values in two stages, first getting the direct intermediate values, then the remaining ones.

<u>Mathematical representation of scaling</u>

How can we describe what we have done (image scaling) using mathematical functions? Note that we have defined a new image that is a function of the old image, with various rules as to how we fill in missing points.

Our function maps one set of row and column coordinates to another, varying the relation between input and output coordinates as a simple scaling operation. Specifically,

$$out(i, j) = in(a \cdot i, a \cdot j)$$

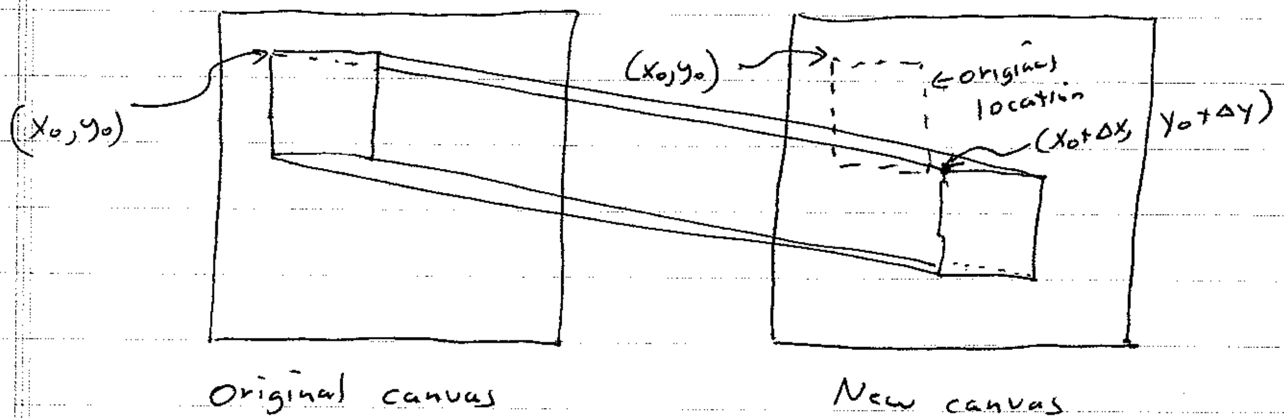
or

$$g(x, y) = f(ax, ay)$$

where g is the output image, f is the input image, and the zoom factor is a. Note we needn't scale by the same amount in both directions.

Shifting images - translation

Quite often we wish to move an image or a section of an image from one position to another, such as in illustration programs. We might want to select a portion of one image and translate it some pixels to the right and down in a new digital canvas:



Let's adopt the convention that the coordinates of an image begin at (1, 1) in the upper left corner, and increase across and down. Then the new coordinates increase in both dimensions for the move shown above. We might mathematically describe the new region using the notation

$$g(x, y) = f(x + \Delta x,\ y + \Delta y)$$

where x and y range over the selected area in the original canvas above, and Δx and Δy are the displacements for the move. The displacements needn't be positive, if they are negative the sign (direction) of the move is reversed.

Note that we can combine the two steps of a shift and a scaling easily:

$$g(x, y) = f(ax + \Delta x,\ by + \Delta y)$$

Another way to think about these geometrical operations is to consider specifically the functions relating the input and output image locations to their counterparts. Let's denote the coordinates in the output image plane as primed coordinates x' and y', and the input image coordinates as x and y. Then in the general case we want to express each primed coordinate as a function of the pair of unprimed coordinates:

$$x' = T_x(x,y)$$
$$y' = T_y(x,y)$$

where T_x and T_y are transformations (functions) that map the original locations to the output locations. The above simplifies more if we let (T_x, T_y) be a vector operator relating (x,y) to (x',y'):

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = T \begin{pmatrix} x \\ y \end{pmatrix}$$

For scaling, T reduces to a scalar value if the stretch in both axes is the same:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = a \begin{pmatrix} x \\ y \end{pmatrix}$$

For variable scaling in each dimension,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

where now a matrix must be used to represent the full transformation. But now adding the shift is trivial, by adding a constant vector to the matrix product:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

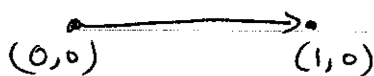
To check, simply expand the above to get

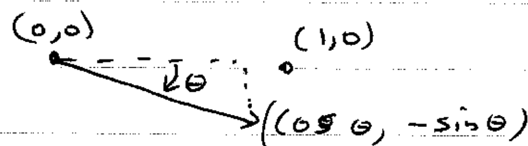
$$x' = ax + \Delta x$$

$$y' = by + \Delta y$$

Rotation

Another simple geometric transformation we use all the time is rotation, that is "spinning" the image a predetermined amount about a coordinate origin. We can derive the necessary transformation by examining how each basis vector changes during rotation. Start with a unit vector in the x-direction:

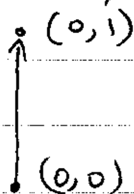


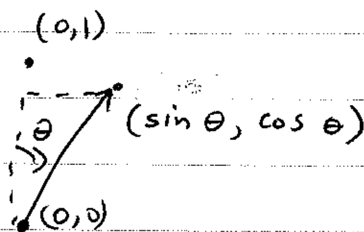


Initial location

Rotated vector

Now the y-direction





Initial location

rotated location

Hence the mapping of an arbitrary vector (x,y) through a clockwise rotation by θ can be expressed as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

or

$$x' = x\cos\theta + y\sin\theta$$
$$y' = -x\sin\theta + y\cos\theta$$

Combining all three transformations yields the matrix equation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

Image addition and subtraction

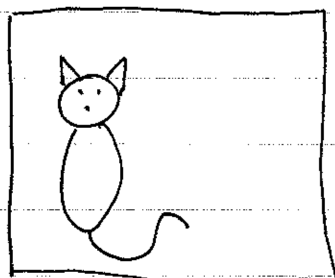
All of the above manipulations involved operations relating the brightness of one point in an output image to a single location in the input image, accounting for a set of geometric distortions. But quite often we want to operate on the intensity of the image as well. Two simple image operations are addition and subtraction of images, easily represented by

$$g(x,y) = f_1(x,y) - f_2(x,y) \quad \text{(subtraction)}$$
$$g(x,y) = f_1(x,y) + f_2(x,y) \quad \text{(addition)}$$

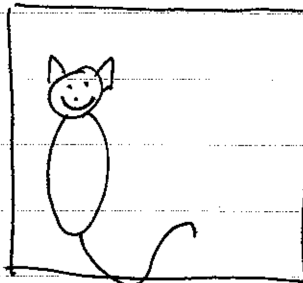
where f_1 and f_2 represent two input images and $g(x,y)$ is their difference or sum. Here we have not altered the geometry, that is the location of corresponding points, in the input and output images, but we have implemented a

mathematical operation on the image intensity.

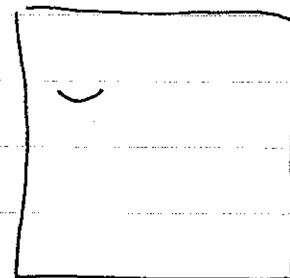
Image subtraction may be used to identify changes in a scene with time:



Time 1



Time 2



Difference

Here the only thing visible in the difference is the cat's smile.

Image addition will be helpful in many processing tasks and in the creation of a number of visual effects later in the quarter.

Note that multiplication or division can be similarly defined. We will find scaling in particular, that is multiplication by a constant, of ~~particular use~~ common use.

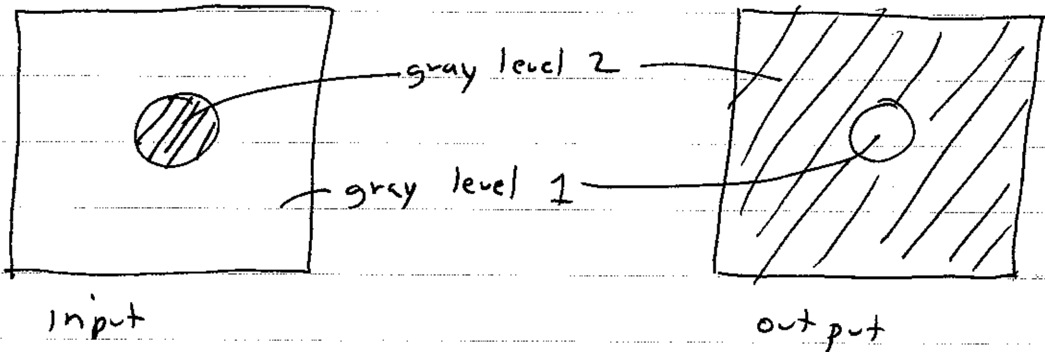
Inversion

One operation we use in image processing that is related to the above, but not the same as any, is that of image inversion. Here we map black into white, white into black, and each gray into a reversed intensity. If we are using 8-bit images, we can represent this as

$$g(x,y) = 255 - f(x,y)$$

This is quite similar to subtraction, but we would use a

matrix of all "255"'s as the first image. This would produce an image similar to a photographic negative.



Here we assume that the relation

$$\text{gray level 2} = 255 - \text{gray level 1}$$

holds.

Note that we can implement the exact same operation using a bitwise NOT operation:

$$g(x,y) = \text{NOT}(f(x,y)) \quad \text{(sometimes called XOR)}$$

where NOT is assumed to flip each bit in a digital number. For example, let $f(x,y) = 93$. In base 2,

$$f(x,y) = 93 = 01011101$$

negate each bit: ↓↓↓↓↓↓↓↓

$$10100010 = 128 + 32 + 2 = 162$$

and $93 = 255 - 162$.

Other image effects result from other bitwise manipulations.