Point transforms - arithmetic & logical operations

1 Matrices

If our image is a matrix then can process (transform) an image by manipulating the corresponding matrix
We can process (transform) images by manipulating the corresponding matrices

2 Transformation aims

Four categories:

- **Remove image degradations** introduced during capture
- **Improve image appearance** for viewing or further processing (i.e. image enhancement)
- **Identify image features** for recognition of scene objects
- **Transform image to alternative representation** for efficient processing

3 Types of image transforms

An image transform processing I_{input} to I_{output} may be:

- A point transform involving only a single pixel at a time
- A local transform involving the local image neighbourhood (pixel+those immediately "next to it" - later in course)
- A global transform involving the whole image (later in course)

4 Basic point transforms

Point transforms map individual points in the input image to individual points in the output image

Performed as an operation, denoted o, between to images, I_A and I_B, or between an image and a constant value C:

$$I_o = I_A \circ I_B$$

$$I_o = I_A \circ C$$

Pixel location (i,j) in the output image is computed as follows

$$I_o(i, j) = I_A(i, j) \circ I_B(i, j)$$

$$I_o(i, j) = I_A(i, j) \circ C$$

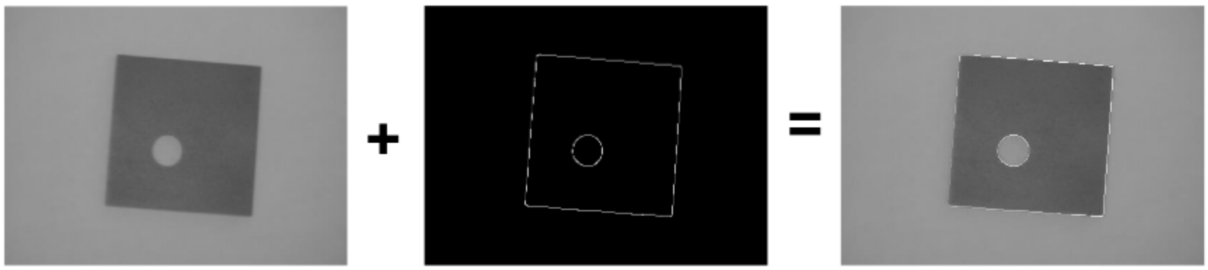
Apply transform iteratively over the image indices for $(i, j) = \{0...w - 1, 0..h - 1\}$
w= image width
h= image height

5 Arithmetic operations

5.1 Addition

Operation: adding a value to each image pixel
Application brightness adjustment: adding a positive constant value to each pixel increases brightness

Application blending: adding images together produces a composite image of both inputs



In both cases watch for integer overflow. The result of the addition can be out of range

5.2 Subtraction

Operation: subtracting a value to each image pixel
Application brightness adjustment: as per addition

Image subtraction can be used to see where things have moved in a scene. However it is subjects to high levels of noise.

5.3 Division

Operation: dividing each image pixel by a value
Application brightness adjustment: uniformly scale image intensities e.g. reduce to 25% of the original by dividing by 4
Application image differencing: dividing an image by another
This is less efficient than subtraction

5.4 Multiplication

Operation: multiplying each image pixel by a value (equivalent to division by the inverse of that value)
Application brightness adjustment: pixel value scaling as per division

6 Application: image blending

Image blending is an application of image arithmetic operations producing ghosting or overlay effects between different images

N images $I_1, I_2, ..., I_n$ can be blended in equal proportions

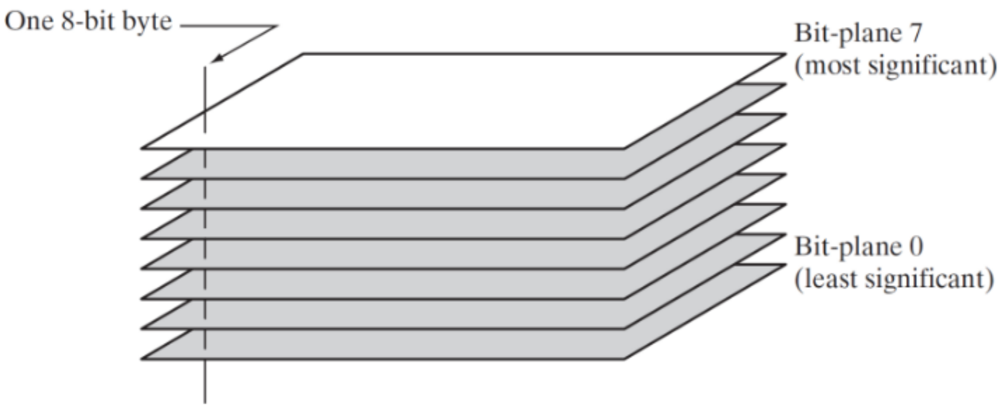
$$I_{\text{output}} = \sum_i \frac{1}{N} I_i$$

Alternatively, different weights can be used between images to enhance/suppress the features of different images in the final result

Blending images

7 Bit planes

Intuitively, logical (bit-wise) operations can be thought of as applied on bit planes



Bit plane representation of an 8-bit image.

- The most significant bit contains most of the information in the image, and so image processing could just be done on that
- Bit planes are used so that image processing can have a less complex image to deal with, and so is faster

8 Logical operations

8.1 NOT (inverse)

Operation: logical NOT to invert the image

For binary images, white pixels become black and vice versa

For 8-bit greyscale images (or 8-bit colour channels)

$$I_{output} = 255 - I_{input}$$

Image inversion is an invertible operation

On colour (and greyscale) images it produces the photographic negative effect

8.2 AND

Can be used for:

- Detecting differences or overlap between images - Not(A) and Not(B)
- Highlighting appropriate regions with a mask - All black with white round selection
- Slicing bit planes through an image - And with powers of 2
- Superimpose images

8.3 OR

An OR operation can be replaced by AND and NOT operations via:

A OR B = =NOT(NOT(A) AND NOT(B))

Can do basic image overlays but no control on weighing available

Quality of OR based overlay is contrast dependant
Here, controlled blending is better

8.4 XOR

XOR is a very useful tool in efficiently detecting image differences as it highlights only where change occur

9 Colour to Greyscale

This is a non invertible, lossy transform, information is destroyed.

A simple way to do the conversion is to take a weighted sub of the R,G,B values:

- Input RGB colour image, I_c
- Output Grayscale image, I_g

$$I_g = \alpha I_c(R) + \beta I_c(G) + \gamma I_c(B)$$

The coefficients (α, β, γ) are usually taken in proportion to the human vision sensitivity to the R,G,B colour channels

One commonly used weighting for the conversion is (NTSC television standard). $\alpha = 0.2989, \beta = 0.5870$ and $\gamma = 0.1140$

10 Computational Complexity

Square $n \times n$ images have n^2 pixels

Computational cost of a point transform applied to the whole image $O(n^2)$

Quadratic runtime means that the total number of CPU operations increases rapidly as the image size increases

Solutions:

- Parallel processing (traditional)
- Image pyramid approaches: down size image, do the processing, then up-size again