*

Image Processing

*Material based on Chapter 11, Image Processing, of B. Wilkinson et al., "PARALLEL PROGRAMMING. Techniques and Applications Using Networked Workstations and Parallel Computers"

©2002-2004 R. Leduc

Low Level Image Processing

Low level image processing is used to improve or enhance an image to aid in human and computer recognition of picture.

Some applications are medical diagnosis, police fingerprints, film industry etc.

Operates directly on the image stored digitally as a two dimensional array of pixels.

Images stored as monochrome images or as color images.

Pixels in monochrome images are given a *gray level* value or *intensities.*

Gray levels have range of value from a minimum value and maximum. Typically zero for Black and 255 for white.

In a color image, each pixel typically has an attached red, green, and blue value.

©2002-2004 R. Leduc

Low Level Image Processing Cont.

We will assume that the image uses a coordinate system where the origin is the top left corner. See Figure 11.1.

In low level processing, we use individual pixel values to alter the image in the desired way.

Typical operations can be classified as:

Point Operations: operation requires the value of a single pixel.

Local Operations: operation requires a group of neighbouring pixels.

Global Operations: operation requires every pixel in image to calculate result.

©2002-2004 R. Leduc

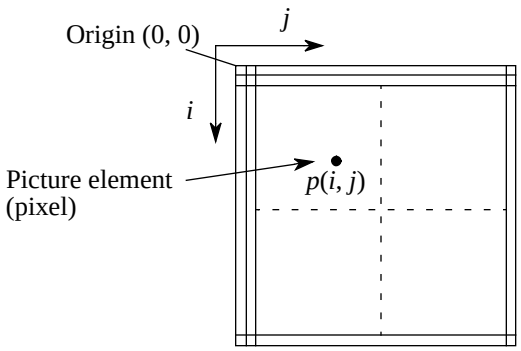

Figure 11.1 Pixmap.

Why Parrallelize?

If we have a 1024×1024 greyscale image (8 bits per pixel), we would have 2^{20} pixels, requiring 1Mbyte of storage.

If each pixel required just one operation, you are looking at 2^{20} operations. For a 10ns/operation computer, that would take about 10ms.

If doing real time operations for video, must do this once per frame with rate typically $60 - 85$ frames/second (ie. every 12-16ms).

Normally, several complex operations need to be applied to the image every frame.

©2002-2004 R. Leduc

Smoothing and Sharpening

Smoothing and *sharpening* are common local operations performed on images. Will assume using monochrome values.

Smoothing removes large fluction in grey-level values over image.

Accomplished by removing high frequency content of the image.

Sharpening emphasizes transitions which enhances details.

Accomplished by removing low frequency content of the image.

We will look at the *mean* and *weighted mask* operators to perform these two operations.

©2002-2004 R. Leduc

Mean Operator

Simple operator that takes the mean or average value of a group of neighbouring pixels.

Performs a smoothing operation.

Requires access to a group of pixels around the pixel to be modified. Often uses a 3×3 group such as in Figure 11.3.

To calculate x'_4, the new value for x_4, we compute:

$$x'_4 = \frac{x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8}{9}$$

Sequential Code: Above calculation requires 9 operations and must be applied to all n pixels.

$$t_s = 9n \text{ and is thus } O(n)$$

©2002-2004 R. Leduc

x_0	x_1	x_2
x_3	x_4	x_5
x_6	x_7	x_8

Figure 11.3 Pixel values for a 3×3 group.

Mean Operator: Parallel Code

To parallelize the code, the easiest way is to divide the image into blocks of rows, and give them to multiple processors.

Assume we have $\sqrt{n} \times \sqrt{n}$ pixels, p processors, and that $\sqrt{n}/p$ is an integer.

Each processor has $\sqrt{n}/p$ rows to process, thus n/p pixels.

$$t_p = 9n/p \text{ and is thus } \quad O(n/p)$$

In both cases, would need two arrays. One to store original image, one to store updated image.

©2002-2004 R. Leduc

Weighted Masks

Similar to the mean operator, but have a weight value for each pixel in the mask.

For a 3×3 mask, would have values: $w_0, w_1, \ldots, w_8$.

If pixels of interest are $x_0, x_1, \ldots, x_8$, the new value for center pixel, x_4, is:

$$x'_4 = \frac{w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + w_8x_8}{k}$$

Typically, $k = w_0 + w_1 + \ldots + w_8$.

The mean operator is equivalent to a mask with all weights set to 1.

To perform a sharpening operation, use:

$$x'_4 = \frac{8x_0 - x_1 - x_2 - x_3 - x_4 - x_5 - x_6 - x_7 - x_8}{9}$$

©2002-2004 R. Leduc

Boundary Cases

Pixels at the edges of the image will not have a full set of pixels to perform mean or weighted mask operation.

Easiest way to handle this is to not process border pixels.

If have $n \times n$ image, then process only pixels:

$p(i, j)$ where $(1 \leq i \leq \sqrt{n} - 1)(1 \leq j \leq \sqrt{n} - 1)$

©2002-2004 R. Leduc