International Journal on Information Theories & Applications, Vol 3, No 6, 1995 (18-25)

An Interactive Environment for Image Analysis and Manipulation

Peter L. Stanchev
Institute of Mathematics, Bulgarian Academy of Sciences
Acad. G. Bonchev St. 8, 1113 Sofia, Bulgaria, e-mail: stanchev@bgearn.bitnet

An interactive image analysis and manipulation environment is introduced in the paper. The environment contains image manipulation functions, voxel operations, neighbourhood and morphological filters. The most popular filters are implemented and an easy interactive way for creating new filters is provided. The environment can work with different types of 2D and 3D images. It can be used as a system for image processing, for developing applications in the image processing domain and as tools for image processing in an image database system. The environment can be used by users with different expertise and various levels of experience.

INTRODUCTION

An interactive environment has always played an important role in image processing, because of its visual nature. The immediate feedback interactive systems reduce the developing time. The functionality of existing environment ranges from a collection of library routines, via command and menu driven systems to specialised programming languages.

There are different image processing packages for IBM PC computers such as Photo Paint of Corel Corporation and Photo Stiller of Aldus. There are also specific image processing packages for the UNIX operating system such as SCIL-Image [4] and ANALYZE [3]. The most popular application in this field for MACKINTOSH is IMAGE. Applying the image environment leads to correcting the distortion in the image and enhancing the image features.

The proposed environment allows: 1) easy way of applying the most commonly used image operations; 2) easy way for making new image manipulation operations; 3) image processing functions in image database systems [7]; 4) using great variety of 2D and 3D image types. In the paper only functions realised in the first version of the environment are described.

METHODS

Let us have a 3D image - *I*. It includes several 2D image slices which can be binary - black and white (1 bit), grey scale (1 byte), grey scale (2 bytes), index 256 colour (1 byte), true colour (3 bytes, a byte for every RGB colour) images. The voxel value at a coordinate *(x,y,z)* can be treated as a value of an intensity function *f(x,y,z)* and can be denoted as *v(x,y,z).*

In this version, the following four classes of image operations are considered in the image environment.

The **first class** of image operations includes image manipulation functions. The **second class** includes different voxel operations. The **third class** contains a set of neighbourhood operations. And the last **fourth class** includes a set of morphological operations.

First Class - Image Manipulation Functions

Functions such as: open, close, save an image, convert from one image type to another, print an existing image, zoom, show the value in a clicked image position, change a value in a clicked image position, extract a rectangular part from an image in a new one, going from slice to slice in a 3D image are realised.

Second Class - Voxel Operations

A voxel operation by a function *g* is defined as:

a) on each single voxel of an image I_1 as:

$g: f_1(x,y,z) \Rightarrow g \circ f_1(x,y,z)$, for every *(x,y,z)* in I_1;

b) on each pair of corresponding voxels of two images I_1 and I_2 as:

$g: f_1(x,y,z), f_2(x,y,z) \Rightarrow g \circ f_1(x,y,z), f_2(x,y,z)$, for each *(x,y,z)* in I_1, and I_2.

An ***arithmetic operation*** is a voxel operation that combines two or more images voxels by voxels. The realised in the image environment arithmetic operations are:

Addition: $g(x,y,z) = f_1(x,y,z)+f_2(x,y,z);$

Subtraction: $g(x,y,z) = f_1(x,y,z)-f_2(x,y,z);$

Multiplication: $g(x,y,z) = f_1(x,y,z)*f_2(x,y,z);$

Division: $g(x,y,z) = f_1(x,y,z)/f_2(x,y,z)$, *for* $f_2(x,y,z) \neq 0;$

Boolean or: $g(x,y,z) = f_1(x,y,z) \vee f_2(x,y,z)$, for binary images;

Boolean and: $g(x,y,z) = f_1(x,y,z) \wedge f_2(x,y,z)$, for binary images.

Image histogram: Histogram(*w*) presents a survey of the intensity values in the image. $Histogram(w) = \frac{\#(f(x,y,z) = w)}{N^2}$, for every *(x,y,z)* in *I*, where # is the counting operator, *N* is the total number of voxels in the image and *w* is between 0 and M_I - the maximum intensity value in the image. For full colour images three histograms are generated for every colour.

Grey scale manipulation functions are also voxel operations. The following functions have been built:

Shift: $g(x,y,z) = f_1(x,y,z) + constant$;

Multiplicative correction: $g(x,y,z) = f_1(x,y,z) * constant$;

Invert: $g(x,y,z) = M_I - f_1(x,y,z)$;

Threshold: $f_1(x,y,z) < t_{hreshold} \Rightarrow g(x,y,z) = 0,$

$f_1(x,y,z) \geq t_{hreshold} \Rightarrow g(x,y,z) = 1,$

where $t_{hreshold}$ is a constant called a ***threshold value***.

User defined pixel operations:

a) specified by the user function over every voxel in an image;

b) specified by the used function, between the voxels of two images.

Third Class - Neighbourhood Operations

The neighbourhood (mask), $N_{(x,y,z)}$, of a voxel (x,y,z), is a set of a voxels close to (x,y,z). The size and shape $N_{(x,y,z)}$ depend in general on the type of operation, the characteristics of the image and the characteristics of the noise [5].

Neighbourhood operation is defined as a function that returns for every voxel in the image a value, derived from the values of the voxels in the neighbourhood of that voxel.

In this environment, it is considered that the neighbourhood $N_{(x,y,z)}$ has parallelogram shape and is symmetrical along the voxel. The dimensions of the parallelogram (f_x, f_y, f_z) are called ***window dimension***. With d_x, d_y, d_z we denote the number of voxels from the central voxel to the border of the parallelogram. So $f_x = 1 + 2.d_x; f_y = 1 + 2.d_y; f_z = 1 + 2.d_z$.

Convolution sum is a special case of neighbourhood operation when the output voxels are obtained as a sum of the multiplication of the mask elements with the image voxels.

A filter is defined as a ***low pass filter*** if: a) the mask elements are positive and b) the sum of the mask coefficient equals 1. The low pass filter does not affect frequency component in the image data and attenuates the high frequency component. The low pass filter dampens the noise and smoothes the image data.

Let $V_t(x,y,z) = \sum_{k=-dz}^{dz} \sum_{j=-dy}^{dy} \sum_{i=-dx}^{dx} v(x+i, y+j, z+k)$ and $number = f_x \,.\, f_y .\, f_z$.

Local low pass average filter. The output image has at voxel (x,y,z) the value $v_{low\_pass} = \frac{v_t(x,y,z)}{number}$. If we smooth with a local average filter, the smooth depends on the size of neighbourhood. The larger the neighbourhood, the more smoothing will be obtained in the output image.

Unsharp Low Pass Mask filter. The output voxels are calculated as: $v_{unsharp}(x,y,z) = v(x,y,z) - v_{low\_pass}(x,y,z)$. This filter eliminates homogeneous regions and highest edges and noise.

Unsharp Low Pass Enhance Mask filter. The output voxels are calculated as: $v_{unsharp}(x,y,z) = v(x,y,z) - v_{low\_pass}(x,y,z)$. It has enhance behaviour than the Unsharp Mask filter.

A ***high-pass filter*** is defined if: a) the mask coefficient are positive or negative and b) the sum of the mask elements is 0. It does not change the high frequency components, attenuates the low frequency components, and eliminates any bias in the image. The high-pass filter modifies the edge voxels and is used as edge detection filters.

Sobel filter. It is a classic edge detection filter. It can be defined by separate kernels w_x, w_y, w_z of each of *x,y,z* directions:

$$w_x(i,j,k) = \frac{i}{(i^2+j^2+k^2)}, w_y(i,j,k) = \frac{j}{(i^2+j^2+k^2)}, w_z(i,j,k) = \frac{k}{(i^2+j^2+k^2)}$$

and the total filter weight is defined as:

$weight = \sum_{k=-dz}^{dz} \sum_{j=-dy}^{dy} \sum_{i=-dx}^{dx} |w_x(i,j,k)| + |w_y(i,j,k)| + |w_z(i,j,k)|$. The *x*-direction Sobel filter is thus defined as:

$S_x(x,y,z) = \sum_{k=-dz}^{dz} \sum_{j=-dy}^{dy} \sum_{i=-dx}^{dx} w_x(i,j,k) * v(x+i, y+j, z+k)$. The *y* and *z* weights are similarly computed. The final output of the Sobel filter is defined as:

$$v_{sobel}(x,y,z) = \sqrt{\frac{S_x(x,y,z)^2 + S_y(x,y,z)^2 + S_z(x,y,z)^2}{weight}}.$$

Sobel Enhance filter. It is defined as

$$v_{Sobel\_enhance}(x,y,z) = v(x,y,z) + v_{Sobel}(x,y,z).$$

The ***Maximum Orthogonal Gradient filter*** is defined as:

$v_{Gradient}(x,y,z) = \max(G_x(x,y,z), G_x(x,y,z), G_x(x,y,z))$, where

$$G_x(x,y,z) = |v(x+1,y,z) - v(x-1,y,z)|, G_y(x,y,z) = |v(x,y+1,z) - v(x,y-1,z)|,$$
$$G_z(x,y,z) = |v(x,y,z+1) - v(x,y,z-1)|.$$

It is an edge detection filter which output resembles Sobel filter.

The ***Sigma filter*** uses a local smoothing scheme. For each voxel in the input volume the filter calculates the mean value of a set of voxels within 2*sigma value of the voxel of interest. Only voxels with a preliminary specified neighbourhood are consider in the calculation. If too few points within the local neighbourhood lie within the 2*sigma value, then the voxel of interest is left unchanged, otherwise, the calculated mean values is assigned to the output voxel. It smoothes noise, preserves edges, and can leave lines untouched.

The masks in the case of 2D images with 3 by 3 pixel masks [2] for the built in the system filters are given in Appendix 1. In the environment, the shape of the mask can be changed by the user.

User defined filters.

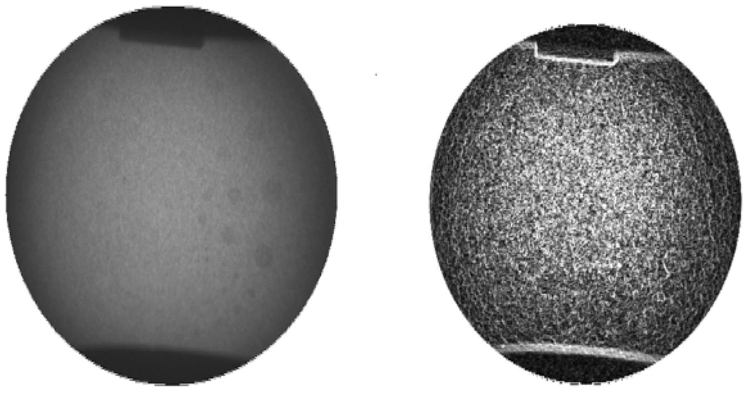
The user can define new filters giving: 1) the shape of the neighbourhood: 2) the function defining the mask elements; 3) the mask elements values.

Four Class - Morphological Filters

Serra and Matheron have found the theory of mathematical morphology [1]. The operations deletion, erosion, opening, closing, Hit-or-miss are defined in the image environment. They are described in details in [8].

AN EXAMPLE

Sobel filters applied over the CT phantom image in Fig. 1a produces the image in Fig. 1b.



(a) (b)

Fig. 1. Original image (a) and the image after applying a Sobel edge detection filter (b).

CONCLUSIONS AND FUTURE WORK

The proposed environment allows:

- working with 2D and 3D images of different types;
- studying the image filter behaviour;

- easy way of creating new image operations;
- applying a large set of image processing functions;
- the support of image processing functions in an image database system;
- visual feedback by applying the environment functions.

The environment has been implemented on IBM PC using Borland C++ under Windows 3.1. The design of the environment uses an experience gained during the development of the AMSTERDAM image data base system [6], working with electronic schema images. It will be part of the image database system [7].

ACKNOWLEDGEMENT

This project is partially supported by a project of the National Foundation for Science Research of Bulgaria.

REFERENCES

1. Haralick R., Sternberg S., Zhuang X., “Image Analysis Using Mathematical Morphology”, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **Vol. PAMI-9, No. 4**, 532-50, July 1987.
2. Louis J., Gabliati J., “Machine Vision and Digital Image Processing Fundamentals”, Prentice-Hall, 1990.
3. Robb R., Hanson D., “A Software System for Interactive and Quantitative Visualisation of Multidimensional Biomedical Images*”, Australian Physical and Engineering Sciences in Medicine*, **14(1):**9-30, 1991.
4. SCIL-Image Manual, University of Amsterdam, 1991.
5. Smoulders A., “An Introduction to Image Processing and Computer Vision”, University of Amsterdam, 1991.
6. Stanchev P., Smoulders A., Groan F., “An Approach to Image Indexing of Documents”, Visual Database Systems”, E. Kenneth and L. Whiner, eds., North Holland, 63-77, 1992.
7. Stanchev P., “Morphological Filters in Image Processing Environment”, to be published, 1995.
8. Stanchev P., “An Image Database with Power Image Retrieval by Image Content”, to be published, 1995.

APPENDIX 1

Let us have 2D images and the neighbourhood is given by a 3 by 3 matrix. In this case some of the common used filters can be defined by the following matrices:

1. Low pass filter: $\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$.
2. Laplacian edge enhancement high pass filters: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$.
3. Gradient-directional filter: $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix}$. The first is in north, the second in north east, and the third in east direction.
4. Shift filter $\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. The first shifts the vertical edges, the second the horizontal and the third the horizontal and vertical edges.
5. Blurs filter: $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The first is for horizontal, the second for vertical and the third for diagonal blurring.
6. Difference filters: $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$. The first is for vertical and the second for horizontal differences.
7. Horizontal difference and vertical smoothing: $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$.
8. Bright region expansion. Maximum of $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.
9. Medium filter: the fifth largest of $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.
10. Enhanced line segment: $\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$. The first is for vertical, the second for horizontal and the third for left-right diagonal line.