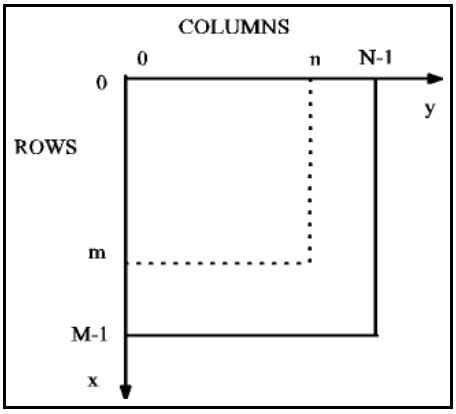
Unit 1. Introduction to Images

1.1 Images

Images and Pixels. An image is a rectangular array of dots called *pixels* (picture elements) where the number of rows M and number of columns N of dots in an image are specified. At each row-column intersection (m,n) there is a pixel, or picture element. The point (m,n) is the *location* of the pixel, while the *pixel value* at that location is designated by p(m,n), or sometimes by f(m,n) or f(x,y). Figure 1.1 shows the location (m,n), where $0 \leq m \leq M-1$ and $0 \leq n \leq N-1$. Note that in the figure the downward vertical direction is x and y is the the horizontal rightward direction. The origin is in upper left corner.

Fig. 1.1 The Image Coordinates.



The pixel values in an image may be *grayscale* or *color*. We first deal with grayscale because it is simpler and even when we process color images we often process the intensity part, which is grayscale, and then put the color back into the processed image. Grayscale usually has a range from 0 (no light intensity, or full black) to 255 (full light intensity, or full white), in integer steps. Thus the 256 grayscale values for pixels are 0, 1, 2, ..., 255. Each of these takes one byte of storage in a computer or in a file, so if an image has MxN = 256x256 = 65,536 pixel,s then that image takes 65,536 bytes of pixel storage in computer memory or on a storage device. An MxN = 1024x1280 image has 1,310,720 pixels. A pixel value p(x,y) or p(m,n), where $0 \leq m \leq M$ and $0 \leq n \leq N$, is a byte (0 to 255 in binary) in grayscale or 3 bytes in color, where the respective bytes are the red (R), green (G) and blue (B) values.

Color images most often take one of two different forms. The most common method is called *true color* and uses one byte for each of red, green and blue. Thus a single pixel value requires 3 bytes of memory or disk storage. From these values we can form (256)x(256)x(256) = 16,777,216 discrete colors, which is about the maximum number of different colors that humans can distinguish. An MxN = 256x256 color image of three bytes per pixel would then require 3(65,536) = 196,608 bytes. For an MxN = 1024x1280 color image the requirement is 3(1,310,720) = 3,932,160. It is clear that more colors and more pixels are more costly in computer storage and time to send on the Internet.

An older format for color is to allow only 256 colors at any one time on the screen. A byte indicates a color but it is actually the address from 0 to 255 of one of 256 color registers, each of which contains 18 bits for 6 bits each of R, G and B. The 256 color set, called a *pallette*, must be loaded into the registers before the image can be displayed, or else the pallette can be read from the image file. This is suitable for many types of images, but the trend is toward even more colors than true color, which may be a waste of resources due to the fact that such fine resolution of color is wasted on humans. Color images are covered in detail in a later unit.

Figure 1.2 shows the grayscale pixel values as a function f(m,n) of the pixel locations at rows m and columns n. Thus we can picture an image as a 3-D surface that has elevation (gray level) as range above the image plane that is the domain. The gray levels are discrete values, so the surface is made of discrete steps at the pixels.

Figure 1.2. Display of a pixel value.

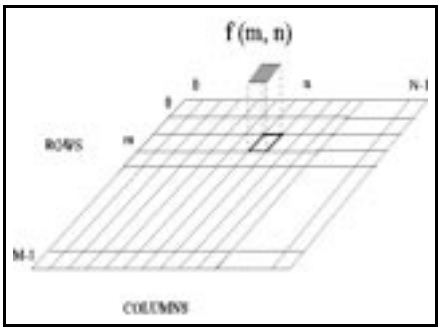


Image Files. An image is stored in a particular file format. The most popular formats nowadays are GIF (Graphics Interchange Format), JPEG (Joint Photographic Experts Group), PNG (Portable Network Graphics), TIFF (Tagged Image File Format), PGM (Portable Gray Map) and PPM (Portable Pixel Map). Examples of the file names for the image *lena256* are *lena256.gif, lena256.jpg* (or *lena256.jpeg*), *lena256.png*, *lena256.tif*, *lena256.pgm* and *lena256.ppm*. The PGM format is strictly grayscale and is the simplest file format for processing (there is no compression nor decomposition of the data into file sections. We examine this format later in this unit.

1.2 Displaying Images with Tools

The Tools. We assume here that the reader has a computer account and a user directory, which we will call *user*. We assume also that the user has the image *Lena*. If the computer has *Linux, Solaris* or *UNIX* as its operating system, then it has the image program *XView* installed (shareware). It should also have *Matlab 6* (or later) installed as well as the extra *Image Processing Toolbox*. If the machine has *MS Windows* installed as the operating system, then it should also have *Matlab 6* (or later) installed.

The program *LView Pro* is also a valuable program, so we should download it to the PC (personal computer). It can be found by bringing up *Netscape* (or *Internet Explorer*) and performing a search on *LView Pro* (or use the network address *www.lview.com*). It will be found immediately and can be downloaded and installed for evaluation. It can also be purchased for a reasonable price with complete documentation, but for this course the evaluation copy may be good enough if the other tools are available. Our main tool will be *Matlab* for either *MS Windows*, *Linux*, *Solaris* or *UNIX* because it is the most powerful and is the only tool that permits us to process images in the frequency domain. We will, however, demonstrate the use of the other tools where useful. In addition, we will include some simple fundamental programs in C for processing PGM image files. The images *lena256.tif, building.tif, shuttle.tif* and *peppers512.tif* are on the CD along with other images for processing.

Displaying the First Image with Matlab. The directions for displaying *Lena* with the *Matlab* tool are given below.

1. From an MS Windows PC: click on the *Matlab* icon. If there is no such icon, then click on the *Start* button (this is usually located on the left side of the bar across the bottom of the screen, but the bar could be on the right side of the screen as a vertical bar). Next, click *Programs* on the menu that pops up and then click *Matlab Release XX* on the new menu that appears. The basic Matlab window will come up on the screen as in the example shown in Figure 1.3. Click in the *Command Window* frame on the right hand side and the cursor will appear (the symbols ">>" indicate where your typed commands will show). Along the top of the Command Window are the words *Current Directory*, followed by a text entry field and an arrowhead pointing down. Click on this arrowhead and then select the directory where your images are stored. We assume hereafter that the directory is set to where the images are that we will load and process.

2. From a *Linux/Solaris/UNIX* Machine: from a terminal window (you must be in X-windows), type *Matlab* at the cursor.

```
> matlab
```

It may take several seconds to load if *Matlab* is being served up by a network server machine. But when it comes up, it will look the same as on a PC.

3. Display the Image: type the following at the command line cursor in the Command Window on the right hand side in the window.

```
>> I = imread('lena256.tif');
>> imshow(I);
```

Figure 1.3. Lena256 displayed by *Matlab*.

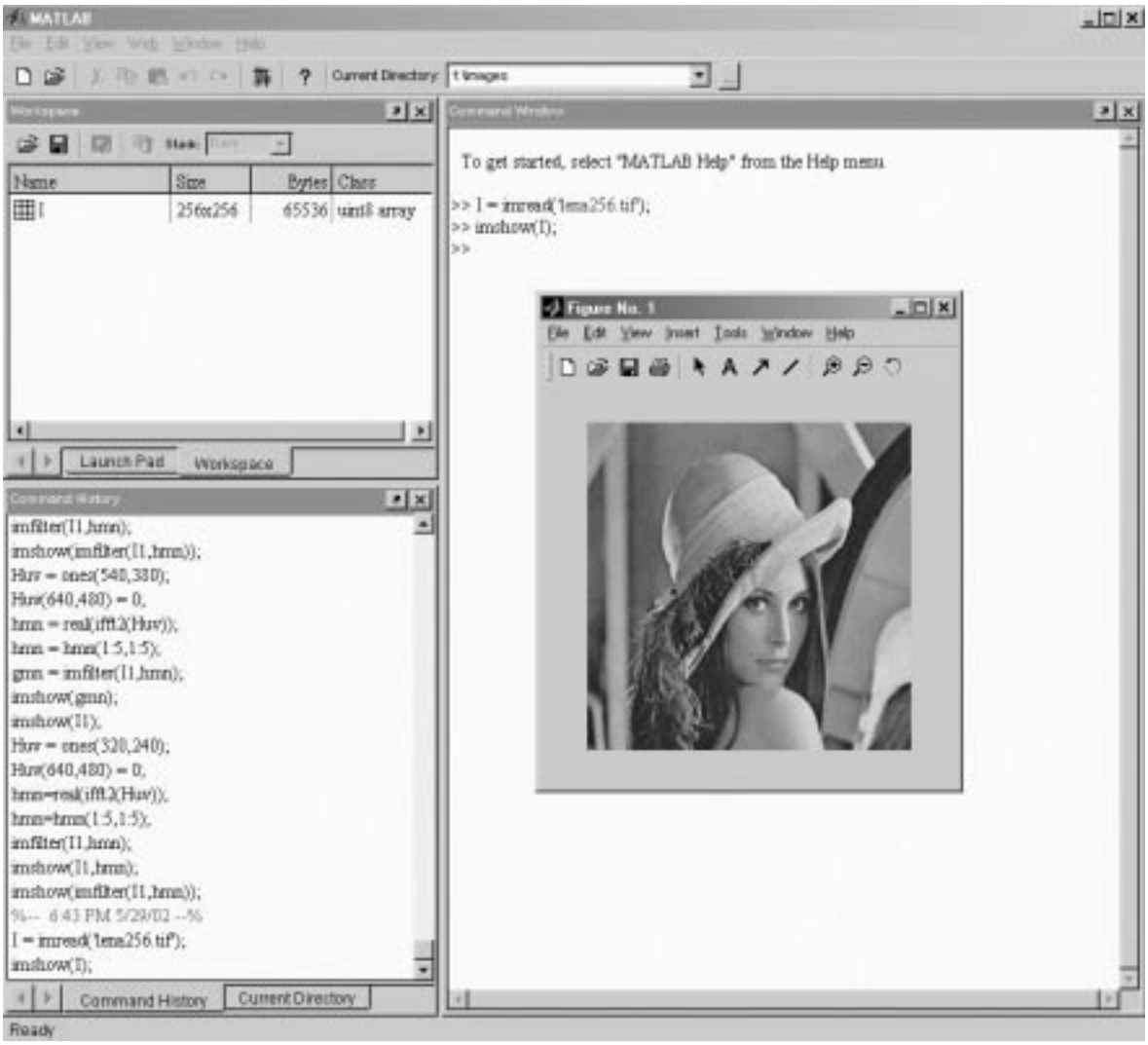


Figure 1.3 shows how to read in the file *lena256.tif*. The first command uses the function *imread()* to read in the file and put the raw image data in memory under the variable name *I*, which could be *Im1* or *myImage* or any other variable name that starts with an alphabetical character. The second command shows the image on the screen by writing it from computer memory into the graphics memory. Figure 1.3 shows the results. To move the image, click down on its top bar and drag to the location desired.

Displaying the First Image with Xview. From any *Linux*, *Solaris* or *UNIX* machine, bring up a terminal window (*X-windows* must be running, which will be the case usually), type the following at the command line.

```
> xv lena256.tif
```

The image comes up on the screen as shown below in Figure 1.4. By right clicking with the mouse pointer inside the image (on a right handed mouse), the *XView* window comes up as shown in Figure 1.5 below. To

end the display, click on *Quit* at the lower right corner of the control window (see Figure 1.5).

Fig. 1.4. *XView* display of *lena256.tif*.

Displaying the First Image with LView Pro. *LView Pro* runs only on PC's with *MS Windows*. It can be downloaded from www.lview.com free for evaluation, but for prolonged or commercial uses the user should pay for it and obtain the complete documentation.

Figure 1.5. The *XView* control window.

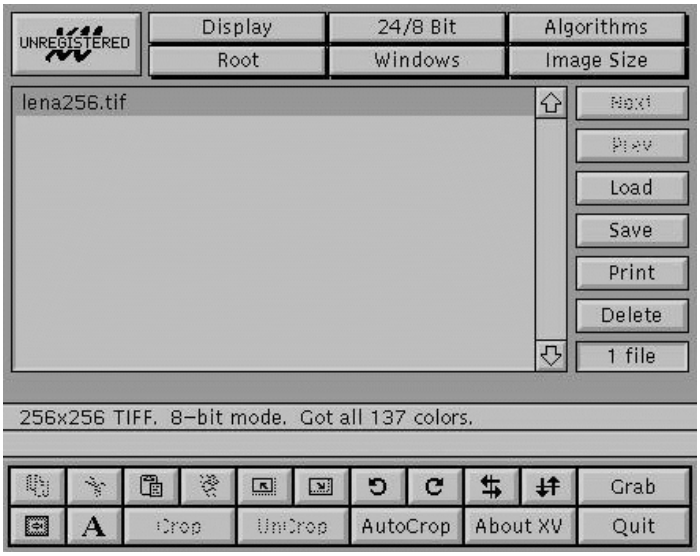
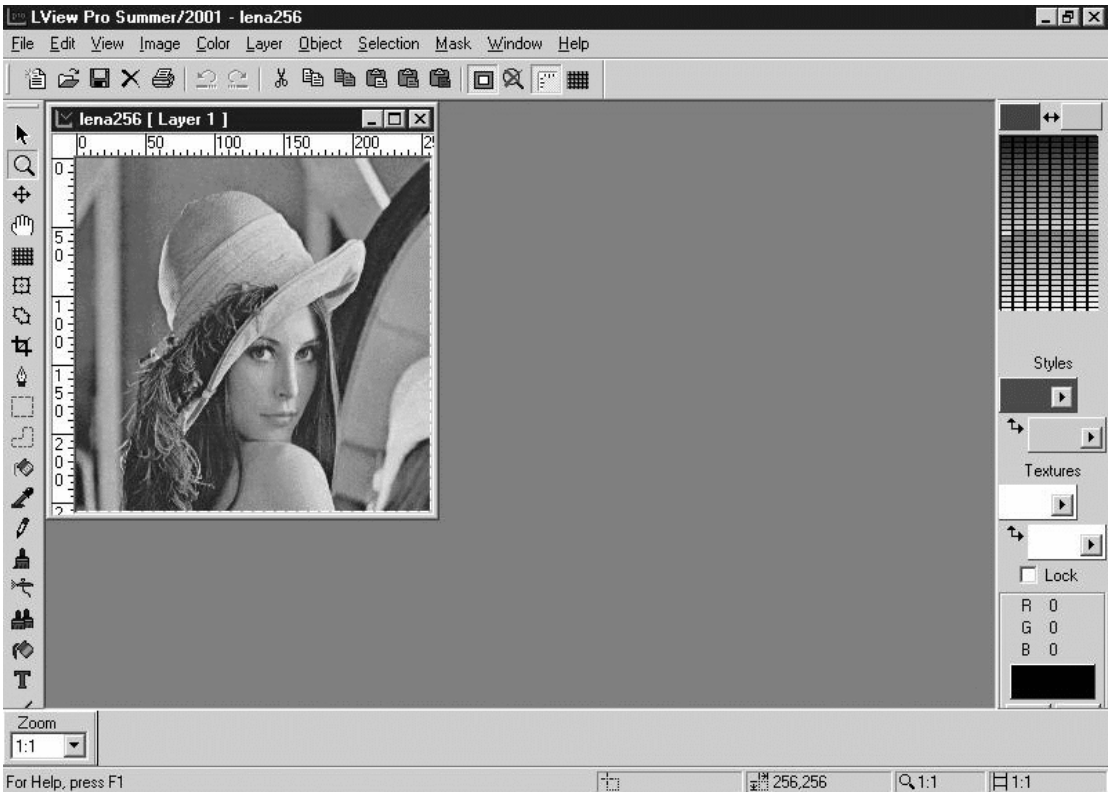


Figure 1.6. The LView Pro main window.



Once it is installed on a PC, it can be run by clicking on the *LView Pro* icon. Otherwise, click *Start* (lower right corner of the screen), then *Programs*, and then click on *LView Pro*. Figure 1.6 shows its main window that appears when it is run. By clicking on *File*, we can then open a directory and select an image file such as *lena256.tif, lena256.jpg* or *lena256.gif*. After this selection by clicking the mouse, the image will be loaded in the window as shown in Figure 1.6.

1.3 Changing Contrast: An Example of Processing and Saving Images

We will use the three tools demonstrated above to process images. Here we use the image *lena256.tif* to show the processing of an image and the saving of it to a file. The processing changes the contrast of the image, that is, changes the range of image gray levels.

Changing Contrast with Matlab. First we run *Matlab* and then read in an image. Then we use the function *imadjust()* to provide the input interval and output interval to which the input interval is to be mapped. The intervals are standardized so that [0, 255] is [0, 1] in the function. The commands are given below where Im1 is the variable into which lena256.tif is read and Im2 is the variable that contains the adjusted image.

```
>> Im1 = imread('lena256.tif');
>> Im2 = imadjust(Im1, [0.2, 0.8], [0, 1]);
```

Here, all standardized shades of gray from 0 to 0.2 are mapped into 0 and all shades from 0.8 to 1 are mapped into 1. The interval from 0.2 to 0.8 is stretched linearly to 0.2 to 1.0. Figures 1.7 and 1.8 show the respective *before* and *after* images.

The general format is

```
>> image2 = imadjust(image1, [low-in high-in], [low-out high-out]);
```

Images can have the contrast stretched or compressed in this manner.

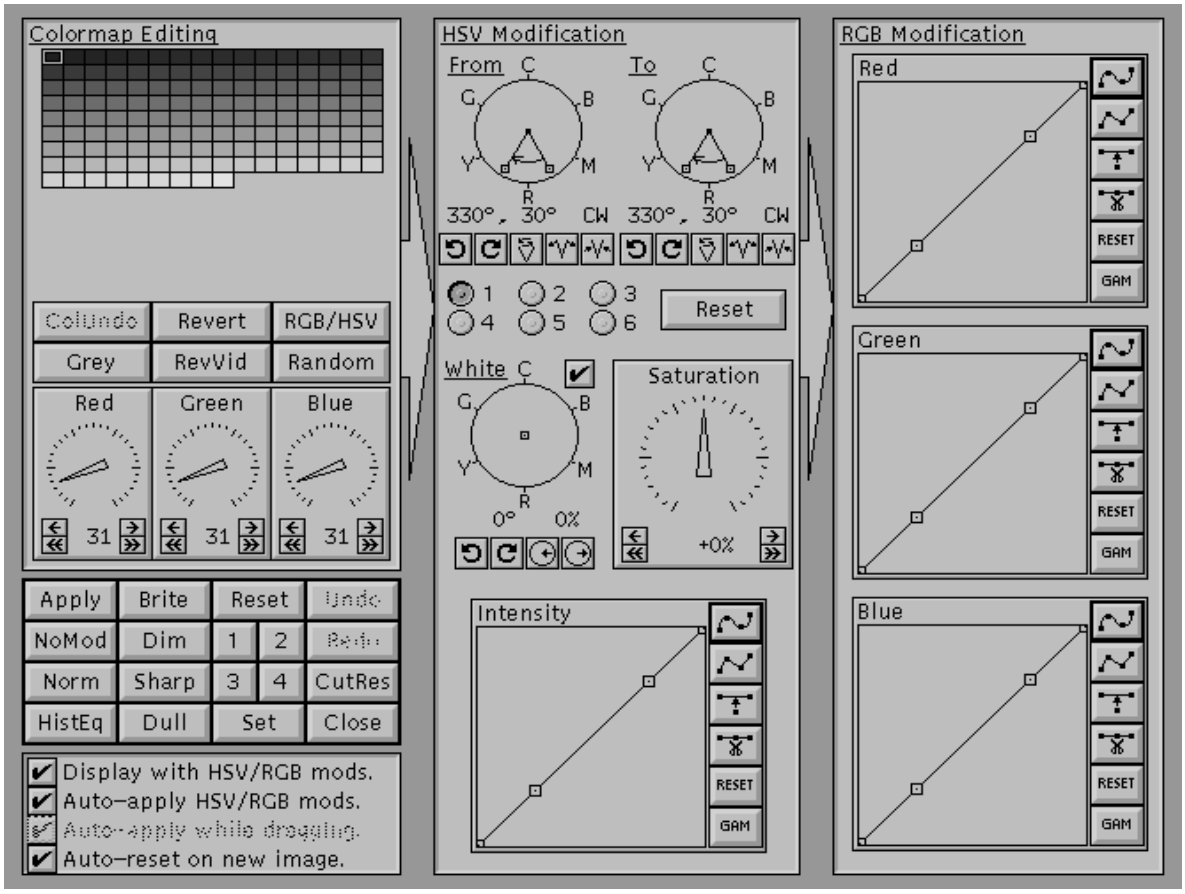
Figure 1.7. The original *lena256.tif*.

Figure 1.8. Matlab contrasted *lena256.gif*.

To save the processed image, click on the processed image to bring it into focus and then click on *File* in the upper left corner (see Figure 1.9). In the menu box that pops up, click on *Save As* to save in the same format or on *Export* to transform to another image file format. Click the down arrow at the bottom to bring up a list of options (EPS is encapsulated postscript, but there are JPEG, TIF, PNG, PGM, PPM and others). When everything is selected as desired, then click *OK*. To print the processed image, click on *File* and then on *Print* (select the *Print Preview* instead of *Print* to see what the printed image will look like).

Changing contrast with XView. We bring up *XView* to display *lena.tif* on the screen and then right click inside the image to display the *XView* control window (shown in Figure 1.5). We next click on the top center button *Windows* and then select *Color Editor* on the pop-up menu. The color editor window of Figure 1.9 comes up on the screen. Now we go to the *Intensity* box at the bottom center and adjust the intensity curve, which is currently a straight line of the form y = x, where x denotes the input gray levels and y denotes the mapped (changed) gray levels for the output image.

Figure 1.9 XView Color Edit window.



The line y = x in the *Intensity* box has two small square nodes on it. Click on the bottom node and move it downward and then click on the top node and move it upward. The result is an "S" shaped curve known as a *sigmoid* function. The new intensity box is displayed in Figure 1.10. The result is that the very dark and very light areas are compressed into fewer gray levels while the mid-range gray levels are stretched out. The results are shown in Figures 1.11 (original) and 1.12 (the contrast enhanced image).

Figure 1.10. The changed XView intensity box.

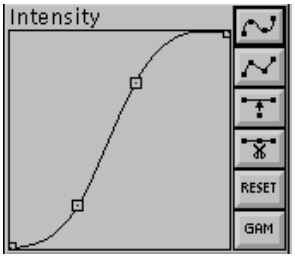
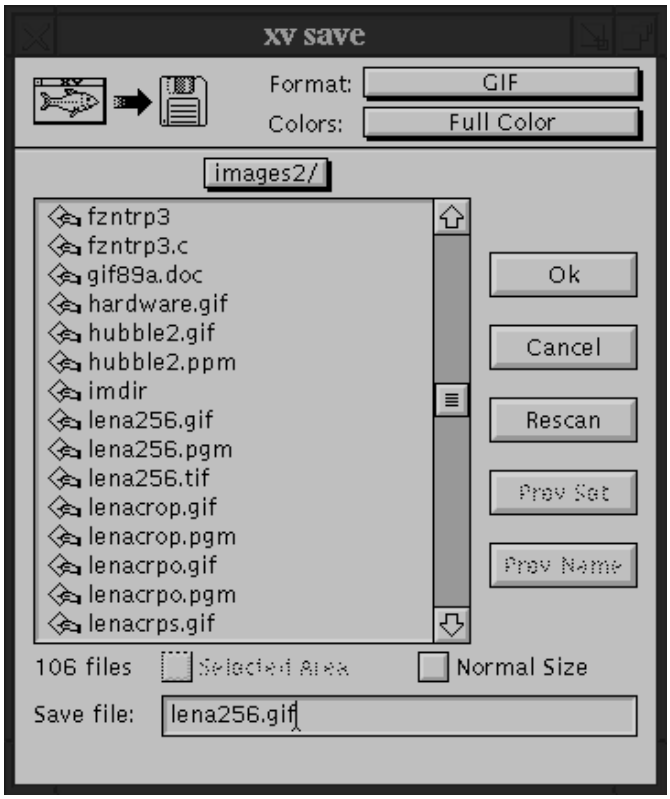


Figure 1.11 The original Lena.

Figure 1.12 The contrast enhanced Lena.

Figure 1.13. The *Save* window in *XView*.



To save the new contrast enhanced image, click the *Save* button on the right side of the control window (see Figure 1.5). The *XV save* window comes up as shown in Figure 1.13. Type in the new file name in the text entry field along the bottom of this new window. Before saving, it is important to click on the *Format* button at the top of this window to select the desired image file format (such as GIF, TIF, JPEG, PNG, PPM, PGM). We suggest PGM for ease of processing, PNG or GIF for use on the Web, or TIF if other tools are to be used on the resulting image (e.g., *Matlab*).

Before we save, we must set a couple of items. The *Colors* button below the *Format* button (in the control window) must be clicked and grayscale selected. When we are processing color images, we will select *Full Color* or *Reduced Color* here as desired (full color uses a large number of colors whereas reduced color is constrained to 256 different colors). We are almost ready to save. Below the *Colors* button (see Figure 1.13), toward the center, there is a button with the name of the current directory (in the figure the directory is *images2/*). Click this button to get the desired directory where the

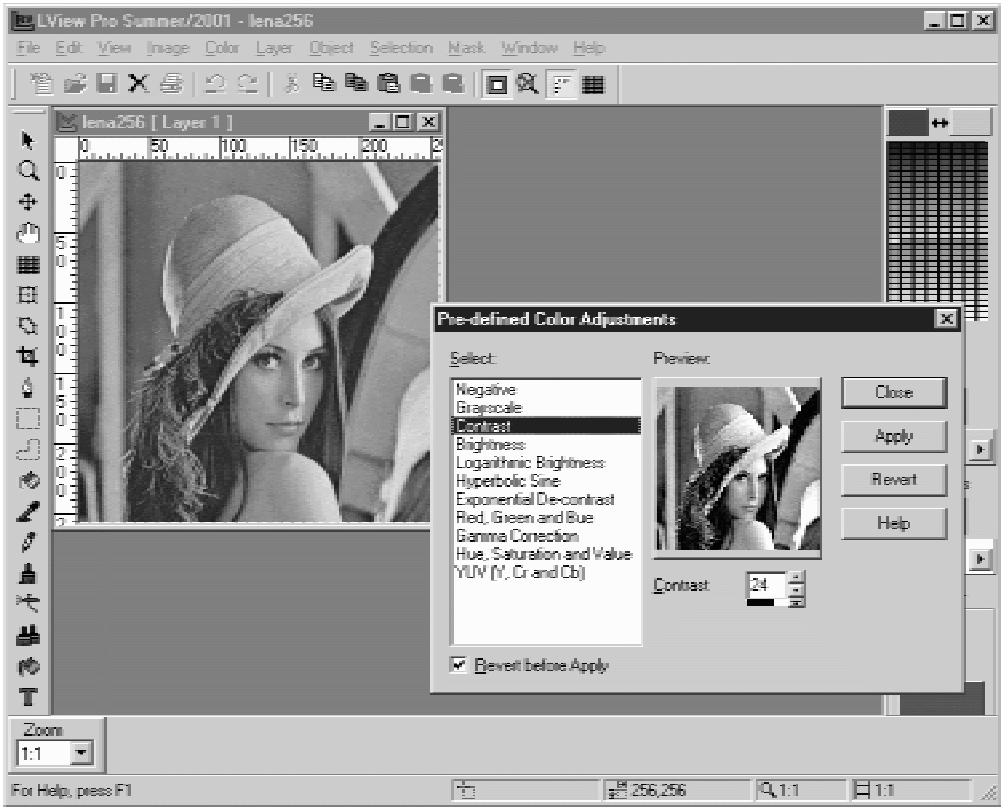
image file is to be saved. Finally, click the *OK* button on the right side (see Figure 1.12) and the image file will be saved in the selected image file format with the selected name in the selected directory.

To print a displayed image from *XView* we must first save an image as a *postscript* file (rather than a TIF or PGM, etc.) and then select *Print* from the control window (Figure 1.5). After selecting the page format to be printed, which will be requested, click on *OK*.

Changing Contrast with LView Pro. Run *LView Pro* and display *lena256.tif*. In the *LView Pro* main window of Figure 1.14, click *Color* in the top menu bar and then click *Adjustments* in the resulting menu to get a pop-up window named *Pre-defined Color Adjustments* as shown in Figure 1.14. Several pre-defined adjustments are now selectable. Click on *Contrast* in the list of options. Now we adjust the value below the thumbnail sketch (image preview) area to change the contrast. A positive number increases the contrast while a negative number decreases it, both from the current image. The thumbnail image will change immediately, but the regular displayed image will remain the same until we click the *Apply* button on the right of the *Pre-defined Color Adjustment* window.

To save the processed image, click on *File* at the left of the bar at the top of the main window. In the resulting pop-up menu select *Save As* and then choose the desired directory and format before clicking the *Save* button. To print a displayed image, select *File* (upper left corner) and then click on *Print*.

Figure1.14. LView Pro window and menu to change image contrast.



1.4 An Overview of Image Processing

Public Domain Images. The image *Lena* of Figure 3 is 256x256 and has been in the public domain for almost half a century (it supposedly appeared in *Playboy* magazine originally). It is often used by image processing researchers around the world for comparison of their methods and algorithms with other ones and appears in most journals on image processing. We will perform certain image processing algorithms on it as well as on other images in the public domain. Another image that is often used is *Peppers512*, which is 512x512 pixels. It is shown in Figure 1.15 below. Other images that we will use often are the public domain images *Shuttle* and *Building*, which are both 320x240 in size. These images are shown respectively in Figures 1.16 and 1.17. These images are quite different and are useful for demonstrating different types of image processing.

Figure 1.15. The original *Peppers*.

Fig. 1.16. The image *Shuttle*.

Fig. 1.17. The image *Building*.

In what follows, we describe two ways of considering the field of image processing. The first is by the needs, or purposes, of the processing. The second is by type and method, of processing. These are listed below.

The purposes of image processing. The list below gives important processes, but not necessarily all.

1. Image Enhancement
- improve image or prepare for some other purpose
- smoothing, sharpening, contrast reduction/increase, color improvement, edge enhancement, edge detection
- adjust histograms
- enlarging/reducing image size, interpolation for values of new pixels
- combining multiple images to show details better
- preprocessing before applying some other methods
- construction of complete image from multiple partial ones

2. Image Restoration
- removal of speckle (dot) noise, short line noises or scratches
- filtering out dark and light bands, elimination of artifacts
- removal of motion blur
- unwarping of spatial distortions
- elimination of glint, glare and haze

3. Image Analysis

- obtain numerical or graphical parameters from image properties
- segment image into similar parts and extract features
- obtain regional features
- measure object parameters
- detect cracks in material or foreign materials in packaged products, etc.
- "see" to detect and track objects in scenes (computer vision)

4. Image Compression

- reduce image file size in number of bytes of images for transmission/storage
- lossless or lossy give respectively low or high ratio of original to compressed size in bytes (e.g., 2-to-1 or 20-to-1)

5. Computer Vision

- detect objects and extract features from images
- use features to make decisions to control or respond to scene activity
- enhance, analyze, measure and track objects in images

The Methods/Types of Image Processing. These are the kinds of processes done on image files.

1. Images

- images, grayscale and color pixels
- grayscale image data structures
- image file formats, PGM, PPM, TIFF, GIF, PNG, JPEG

2. Point Processes

- threshold gray levels
- contrast stretching and contraction
- pixel transformations, histogram adjustment
- map pixel values to hide or expose certain objects
- histogram equalization, dual histogram equalization

3. Area Processes

- transform pixel values according to the values of it and its neighbors
- smooth, sharpen, detect edges, median filters
- filter out noise, scratches (despeckling)
- trimmed median and convolution filtering

4. Frame Processes

- registration of two or more images
- combining images via pixelwise sums, subtraction, multiplication, division
- combining images via pixelwise boolean logic (or fuzzy logic) functions
- combining by minimum or maximum operations at each pixel
- fuse images by other mathematical methods
- multispectral methods

5. Geometrical Processes

- expand/shrink lighter areas relative to darker areas to smooth boundaries, fill holes and remove noise
- affine/linear transformations to rotate, translate and scale images
- interpolation, transformed pixel locations, downsampling and upsampling, zooming in and out
- nonlinear transformations to remove distortion, mirror or flip images
- segmentation, clustering of pixels, labeling

6. Frequency Domain Analysis

- Discrete cosine transforms (DCT's)
- Fast Fourier transforms (FFT's)
- lowpass, bandpass, highpass and bandstop filters in the frequency domain
- Gaussian filters in the frequency domain
- Convolution and frequency filters in the spatial domain, spatial Gaussian filters
- Deconvolution, blind deconvolution
- registration using frequency features

7. Color Image Processing

- capturing color images, color cameras, color scanners
- human color perception
- the color models RGB, CMY, HSI, CMYK and color model transformations
- intensity levels in color images, processing color intensity
- pseudo-color for image enhancement
- color image file formats: PPM, GIF, TIFF, JPEG

8. Image Compression

- lossless and lossy encoding
- run length encoding
- LZW compression, GIF and PNG
- discrete cosine transforms (DCT's) and JPEG
- the Carlson semi-lossy 4-to-1 fast algorithm

9. Special Topics: Stereoscopy, Synthesis and Data Visualization

- stereo vision
- image synthesis
- data visualization

1.5 The Field of Image Processing

The Origin of Image Processing. Imaging began in the 19th Century with photography and continued with x-rays, television and electronic scanning in the 20th Century. Image processing as a field of study began in the 1950s with pictures of the earth from high flying "spy" airplanes and then with pictures of the earth's surface taken from orbiting satellites. Electronic sensors were sent into space to probe the surfaces of the planets and their moons in the 1970s and 1980s. The newer infrared and optic sensors, and additionally synthetic array and high range resolution radars create images that require intensive processing to reveal details for detection and classification of man-made objects, crops and other foliage and of minerals. These are captured from ground stations, unmanned aerial vehicles, airplanes and satellites.

Applications of Image Processing. Nowadays, image processing is used in

medical diagnostics, forensics, biological microscopy,
inspection of parts and materials, crop yield estimates,
foliage types and area estimates, minerals, defense intelligence,
topographic maps (a type of stereo vision), ocean temperatures,
meteorology and other areas

An important developing area that is based mainly on image processing is computer vision. It includes enhancing images, selecting objects, identification and recognition of objects, monitoring the behavior of the objects, tracking objects, and related areas.

1.6 PGM Grayscale Image Files

P2 and P5 Types of PGM Image Files. The simplest file format for grayscale images is the *PGM* format. It has two subformats: i) one known as *P2* that stores pixel values as the ASCII characters for the digits, delimited by spaces between consecutive pixel values; and ii) another known as *P5* that writes the bytes without delimiters binary numbers from 0 to 255.

Each PGM file has a filename, such as *lena256.pgm*, and a header of three lines. The first line contains either the characters *P2* or *P5*. The second line starts with the character # and is a comment that provides some information about the image. The third line contains three integers that are respectively the number of columns, number of rows and the maximum gray level (usually 255).

Viewing PGM Files with an Editor. Table 1.1 shows a P2 PGM file for *lena256.pgm*. The datum 158, for example, is three ASCII characters for the respective digits *1*, *5* and *8*, followed by a space. Table 1.2 shows the P5 PGM version where each pixel is stored as a byte that is a binary number from 0 to 255 (text editors show 0 through 127 as ASCII characters and 128 through 255 at whatever code the editor uses for those values).

Table 1.1 The P2 PGM file *lena256.pgm*.

```
P2
# Created with LvVew Pro
256 256 255
158 165 158 158 158 158 155 158 155 161 155 150 155 155 158 151 150 .....
```

Table 1.2. The P5 PGM file l*ena256.pgm*.

```
P5
# Created with LView Pro
256 256 255
ž¥žžžž›ž›¡›–››ž——››–¡¡®®«§«Ÿ——†u`^Zfflle‖`llllllglIxnunuu{}□| †„}†ˆ□.....
```

We see that the P5 type of PGM is *packed* (without delimiters) so there is a single byte for each pixel. This is also called the *raw data* format. The size of this file is 65,576 bytes. The P2 type of PGM uses a byte for each numerical symbol (digit) and therefore requires three bytes for each number greater than 99 and also uses a space character after each pixel value. Thus the file is nearly four times as large. This P2 file is 245,724 bytes. However, humans can read the P2 type of PGM file, whereas they can not read the ASCII characters of the packed bytes, which can appear different in different editors (characters 128 to 255). Appendix 1.A gives the ASCII code.

To access the data in either type of PGM file, we read the first three lines, set the the number of columns and rows, and then read the rows into a data structure. Then we can process this data and write the new pixel values out to a new PGM file.

1.7 Converting to and from PGM File Formats

The quickest and easiest way to convert an image such as *lena256.tif, lena256.gif, lena256.png*, or *lena256.jpg* to *lena256.pgm* is to use *Matlab, XView*, or *LView Pro*. The first can be used with either the *MS Windows* or *Linux* (including *Solaris* and *UNIX*) operating system, while the second one does not work on *MS Windows* but the third one does (and not on any *UNIX* type operating system).

***Converting with* Matlab**. Run *Matlab* first and then load the image into memory and show it with the command lines given below. Then click on the image to bring it into focus. Next, select *File* in the upper left corner of the image frame, then click on *Export* in the menu that pops up and select the directory where the

file is to be saved. Find the textbox with the label *Save as type:* to its left and click on the option arrow at the right of the textbox to see the options. Select the type of file to convert to from these options (TIF, PNG, JPEG, PGM, etc.). Finally, click on *Save*. The loaded file will be saved in the selected directory in the file format choosen.

To load and display the image *building.tif*, for example use the following commands.

```
>> Image1 = imread('building.tif');
>> imshow(Image1);
```

***Converting with* XView**. Bring up *XView* to display *lena256.tif* with the command line

```
> xv lena256.tif
```

Then left click inside the image to bring up the Control Window. Next, click on Save on the right side. Then select Format at the top right and choose PGM binary. Note that the name of the file is displayed in the text field at the bottom of the menu. Click OK to save in the current directory (see Figure 1.12 to find the button on which to click to change directories).

To view the file we can use an editor such as EMACS, vi or pico to load the file. For example

```
> emacs lena256.pgm
```

This will bring up the file to expose the data shown in Table 1 above. After examining the file data, we can exit without saving to preserve the integrity of this file. Sometimes *XView* puts in an extra comment line (when operating on a PGM file and writing to another PGM file) and we should delete the extra line to avoid possible trouble later).

***Converting with* Lview Pro**. Run *Lview Pro*, then select *File* on the upper left corner, then click on *Open*, select the directory and click on the image file *building.tif*. The image file will be loaded into memory and displayed on the screen automatically. Next select *File*, *Save As*, select the options arrow to the right of the textbox with label *Save as type* and select, for example *Portable (ppm, pgm, pbm)*. Choose the *Grayscale* radio button. Now click on the *File Type Options* button on the right and select the *PGM* tab and then *Binary Format* (this packs the bytes as raw data and not in the ASCII characters to form an integer). Click the *OK* button and then the *Save button*. The image will now be written in the new file format.

Warning: *Lview Pro* saves all PGM files with the suffix *ppm* rather than *pgm*, but the file header will use the P2, P3, P4, P5, etc. designations accurately.

1.8 Exercises

1.1. Suppose a color image file has 1024 rows and 1024 columns, uses *true color* of 3 bytes per pixel (one byte for each of R, G and B). The display is to use 60 frames per second. How many bits per second (bps) is required for the video signal from the graphics card to the monitor?

1.2. Write an algorithm in pseudo-code that reads the lena256.pgm data and computes the average pixel gray level over the image pixels. Now do the same for an algorithm that computes the mean-square error (variance) of the gray level over the image.

1.3. Convert *lena256.tif* to the file *lena256.pgm* by use of *Xview*. Now convert the original file to lena256.jpg using *Matlab*.

1.4. Use an editor and look at the data in the *lena256.tif* file. What can be surmised from this display?

1.5. Use *Matlab* to map the gray levels 0 to 50 to black, shades 200 to 255 into white and 51 to 200 in the interval [51, 199] on the image *lena256.tif*.

1.6. Use *LView Pro* to convert lena256.tif to lena256.pgm in the binary format (P2). Now open the output file in an editor program and examine the data. Is the P2 written on the first line? Is each digit in the number for a pixel an ASCII character or is the entire number (pixel value) a single binary byte?

1.7. Bring up *lena256.tif* with *Matlab*. Then click on *File* (upper left corner), go down to *Export* and follow the options to save it in the current directory as a PGM file.

1.8. Use *XView* to load *lena256.tif*. Then bring up the control window, select *Color Editor* and change the *Intensity* box line y = x so that all output pixels are either black or white. Examine the resulting image. Move the point at which the black changes to white and notice the change in the image. This reduces an image to black and white only (two segments).

1.9. Write a message on a piece of paper with black ink and scan this as an image. Use a tool to lower the contrast until the message can not be read. This file can be displayed but no one can read it. Now use a tool to stretch the contrast to include black and white and all shades of gray in between. What is the result?

1.10 Segment the image *lena256.tif* into an image with only four shades of gray by working with the *Intensity* box of *XView*. By clicking on either of the top two buttons on the right of this box, we can add more *tics* (small square objects) on the line to allow finer adjustments to be made to the intensity function. When you are satisfied with the segmentation, then print out the image (postscript must be used).

Appendix 1.A - ASCII Code

Decimal	Hexadecimal	Symbol
0	0	NULL
1	1	SOH (Start of Heading)
2	2	STX (Start of Text)
3	3	ETX (End of Text)
4	4	EOT (End of Transmit)
5	5	ENQ (Enquiry)
6	6	ACK (Acknowledge)
7	7	BEL (Bell)
8	8	BS (Backspace)
9	9	HT (Horizontal Tab)
10	A	LF (Linefeed)
11	B	VT (Vertical Tab)
12	C	FF (Formfeed)
13	D	CR (Carriage Return)
14	E	SO (Shift Out)
15	F	SI (Shift In)
16	10	DLE (Data Line Escape)
17	11	DC1 (Device Control 1)
18	12	DC2 (Device Control 2)
19	13	DC3 (Device Control 3)
20	14	DC4 (Device Control 4)
21	15	NAK (Negative Acknowledge)
22	16	SYN (Synchronous Idle)
23	17	ETB (End of Transmit Block)
24	18	CAN (Cancel)
25	19	EM (End of Medium)

26	1A	SUB (Substitute)	
27	1B	ESC (Escape)	
28	1C	FS (File Separator)	
29	1D	GS (Group Separator)	
30	1E	RS (Record Separator)	
31	1F	US (Unit Separator)	
32	20	(Space)	
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	,	
45	2D	-	(Dash)
46	2E	.	(Period)
47	2F	/	
48	30	0	
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	
64	40	@	
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	

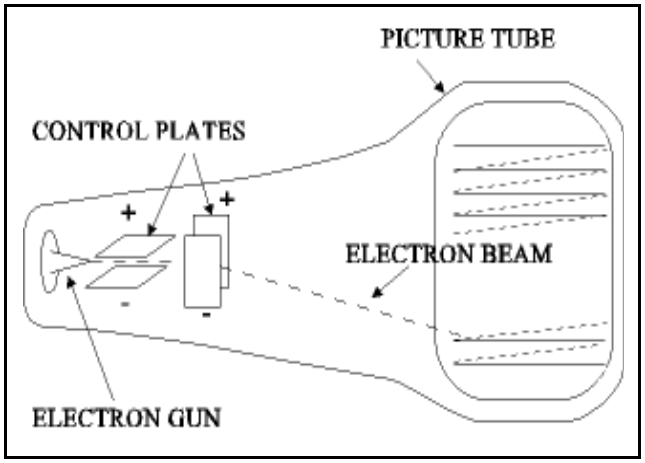
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	
92	5C	\	
93	5D	]	
94	5E	^	(Caret)
95	5F	_	(Underline)
96	60	`	
97	61	a	
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6A	j	
107	6B	k	
108	6C	l	
109	6D	m	
110	6E	n	
111	6F	o	
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	77	w	
120	78	x	
121	79	y	
122	7A	z	
123	7B	{	
124	7C	|	
125	7D	}	
126	7E	~	
127	7F	DEL (Delete)	

Appendix 1.B - The Display and Capture of Images

Cathode Ray Tubes for Display. The inside surface of the screen of a monochrome monitor is coated with a phosphoric material that converts the energy of colliding electrons into light emission on the outer side. This material is uniformly coated so that the light emitted is of a single color such as white, amber, or green. Any spot where there has been an absence of colliding electrons for a short time appears dark to the viewer. The screen is the flattened end of a large vacuum tube that contains an electron gun at the other end. The gun's cathode is heated electrically so that it emits a stream, or beam, of electrons toward the inside of the screen. Such a device is called a *cathode ray tube* (CRT).

Figure 1.B.1 shows the scheme. Electrons pass through two different pairs of parallel metallic plates, of which one pair is horizontal and the other is vertical. A voltage difference across each pair pulls the electrons up or down, left or right. The *control signal* voltages on the plate pairs are designed to force the electron beam to move across the screen in rows that are imperceptibly slanted downward and then return across to the left at a position one row down. This continues until all rows on the screen have been traced, at which time the beam tracing system is reinitialized to repeat the process. Such reinitialization causes a very short time delay. This fashion of painting the screen is call *raster scanning*.

Figure 1.B.1. The Picture Tube Scheme.



The denser the electron beam on any dot of phosphor, the brighter is the light emitted from that dot on the screen. If the beam density is sufficiently low, the dot appears black, while if it is at maximum level, the dot emits the maximum intensity of light and is white. The *intensity signal* f(t) is determined by the image data and f(t) controls the electron beam density at each instant that corresponds to a position in the raster scan. The timing of the raster scan control signal is such that a small dot, or area, say, of 0.26 mm diameter, is excited by electrons during a very small time interval Δt. Such a dot is called a *pixel* (for "picture element"). When the entire screen has been painted, that is, all pixels have been excited in order during the raster scan, we say one *frame* has been executed.

A graphics interface card connects into the computer bus (data bus, control bus, and power bus, where *bus* denotes a set of lines that carry signals). It converts binary values that represent the intensity level of the pixels into a voltage signal that controls the intensity of the electron gun at the specific times that particular pixels are being painted via the raster scan. In this manner, a temporary image is painted as a frame. The *persistence* is a property of the phosphor in that it keeps emitting light for a short time after excitation stops. The first pixel in a frame is still emitting light when the last pixel in the frame is being excited. Such light emission must decrease and be significantly less perceptible in a fraction of a second so that it does not garble symbols in the next frame.

A flat digital screen is a matrix of tiny units, each of which is excited by voltages across the respective column and row lines at the matric point (m,n) of intensity f(m,n). This is a more natural way to display an image on a screen.

Image Data and Data Rates. Rates greater than 44 frames per second are necessary to avoid the human perception of flickering. To achieve this rate, previous display systems traced alternate rows to the screen and then on the next scan wrote the rows in between. Thus 44 frames of half the number of rows were traced each second to avoid the flickering, although the actual rate was 22 full frames per second (a trick borrowed from the cinema industry). Such a method of scanning is called *interlaced*. Nowadays graphics systems are mostly

noninterlaced and can display more than 44 frames per second, usually 60Hz (*Herz*, or cycles per second, which here means frames per second), 75 Hz, 90 Hz, 120 Hz or higher.

A 1280x1024 screen with has 1,310,720 pixels. Let each grayscale pixel have an intensity value from 0 to 255 (one byte, or 8 bits). Then a file of 1,310,720 bytes is needed to store an image. A stream of bytes is read from a file and written to the graphics memory on the graphics interface card and the values are used on the next scan. The image may not appear on the screen instantaneously. Some UNIX systems wait until all data is in graphics memory and then put it all on the screen in a single frame scan, so the image appears on the screen instantaneously. At 60 frames per second of 1,310,720 values of 8 bits each, this requires a graphics system that has a rate of 629,145,600 bits per second (629.145 *Megabits per second* or *Mbps*).

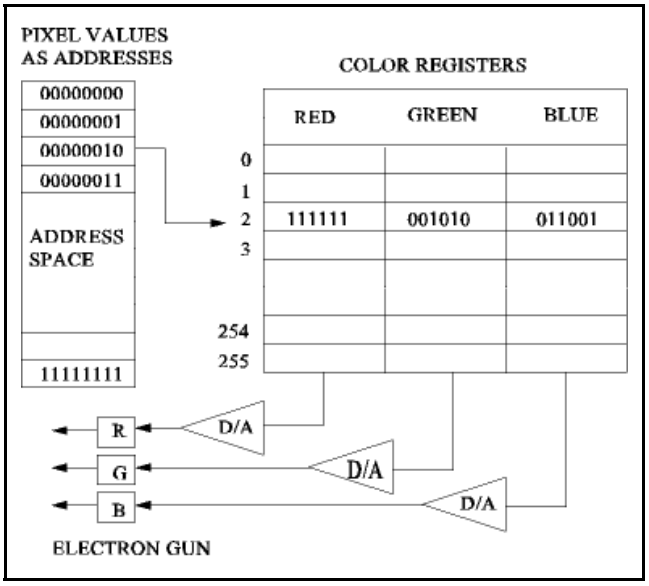
Display of Color Images. The display of a color image requires three different phosphors that respectively emit three independent colors of light when excited. It is known that any color can be obtained by adding the correct proportions of red (R), green (G) and blue (B). Color video display systems use this property. A color video monitor contains closely packed dots of three types of phosphor on the inside of the screen. Each pixel has an area composed of three dots: one each of R, G and B light emitting phosphor. Three electron guns are used of which one excites each of the red, green and blue phosphor dots. The R gun, for instance, is set to scan along the R phosphor dots, while the others each trace along their particular set of dots. The pixel color is determined by the combination of intensities of each of R, G and B. Equal intensities of R, G and B make a gray shade, from black to white.

A color image file consists of binary values for the pixels as does a monochrome image file. The difference is in the interpretation of these values. In one method, a binary value for a color image is broken into three parts to produce a binary value for each of the R, G and B electron guns. For example, in *true color* a pixel value is 3 bytes long. The first byte represents R, the second is for G and the third is for B. These files are quite large. As an example, an 800x600 image would take 480,000x3 = 1.44 MB (Megabytes). However, each pixel can take one of 2^{24} = 16,777,216 colors. Each of R, G and B can take 2^8 = 256 intensities. For example 11111111 00000000 000000 is a pure red of the highest intensity, while 00001111 00001111 00001111 has equal intensities of 15 for each of R, G and B and so appears dark gray (equal values for R, G and B is always a shade of gray). A value of 24 bits of all 1's gives the highest intensity of gray, which is white.

Another common method is to use a single byte for each pixel value so that there are 256 values. In this scheme each pixel value is actually the address of one of 256 registers of 18 bits (6 for each of R, G and B). Functions can be used to put a set of 256 RBG colors in the 256 registers, so that 256 different color images can be displayed in a single image. While a single image can use only one set of 256 colors, the color set (pallette) can be changed for another image (but not part of the way through a raster scan to put a single image on the screen with colors from another pallette). Because each 256 colors set can be selected from 2^{18} = 262,144 different colors, the number of such pallettes is (262,144)!/[256!(262,144 - 256)!], which is the number of ways 256 things can be selected from 262,144 unique things (a very large number). Thus one image can show 256 shades of various yellows for flowers and some greens for the stems and leaves, while another image may show various shades of blues and purples. When the R, G and B 6-bit values are all equal in the 18-bit color registers, then there are 256 shades of gray.

Figure 1.B.2 shows the registers. If one image is displayed in 256 colors and another is to be displayed after that in a different set of 256 colors, then the color registers must be given new color values (rewritten by a computer program function). Each color register contains the three R, G and B parts of 6 pixels each that control the three respective color gun intensities for exciting the respective set of R, G and B phosphor dots in the pixels. While the color values of 6 bits each are digital values, they are transformed by a *digital-to-analog* (D/A) converter into analog (continuous) signals to control the intensity of the color guns.

Figure 1.B.2. VGA 256 Color Scheme.



Different file formats exist for image color information. Before we process an image we will convert it into a PPM color file format (3 bytes per pixel in the raw packed data file) or a PGM grayscale format. After processing, it can be translated into another file formats if compression is needed to reduce the size.

Capturing and Storing Images. An image is often captured by passing a photograph through a *scanner*, which may have color capability. It may also be captured directly from a video camera that digitizes the light signal into groups of bits. In the first case, a set of small optical sensors, or photovoltaic diodes, one of which is shown in Figure 1.B.3, is arranged to form a row as shown in Figure 1.B.4. Light rays from the tubular light source reflect from a horizontal line along the picture into the small optical sensors. Each single device receives the light reflected from a small dot area on the picture. Its average dot intensity is converted into current by the device and then captured into memory.

The timing signals of the scanner cause the currents associated with the dot intensities along a horizontal row to be *captured* into digital values by shifting and latching (writing) to registers, which are sequences of bit-memory devices (*flip-flops* that can be set to 0 or 1). The picture then moves a slight amount in a direction perpendicular to the row of sensors and then another row of dots is captured similarly, and so forth until the entire page has been captured as digital data.

The size of the detectors and their closeness together determine the sampling interval along the horizontal (row) spatial dimension. The resolution is measured in *dots per inch* (dpi). Many inexpensive scanners capture 600 dpi horizontally, but because they can move the picture a slight amount, they can obtain 1200 dpi (or more) vertically. The horizontal dots are often interpolated to a resolution of 1200 dpi (or more) by inserting a dot between each pair in a row. More expensive scanners interpolate to thousands of dpi. However, high resolution color images are slow in printing, displaying and transmitting on the Internet and take up large amounts of storage space.

A video camera signal is fed into an interface card (a printed circuit card with logic chips installed) that is connected to the computer busses (data, control, address and power busses), which then captures the sampled values as a stream of binary values by storing them in memory on the card. As the memory on the interface card fills with data, an associated computer program transfers the data into a buffer in computer memory for writing to a file on disk. The data goes from the hard disk to the graphics card for display.

Figure 1.B.3. A photovoltaic diode.

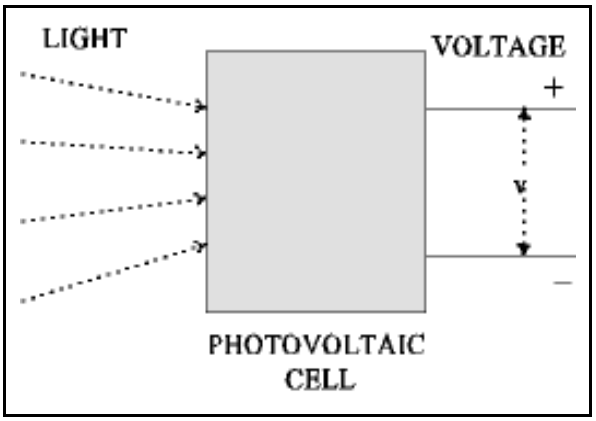


Figure 1.B.4. An Image Scanner.

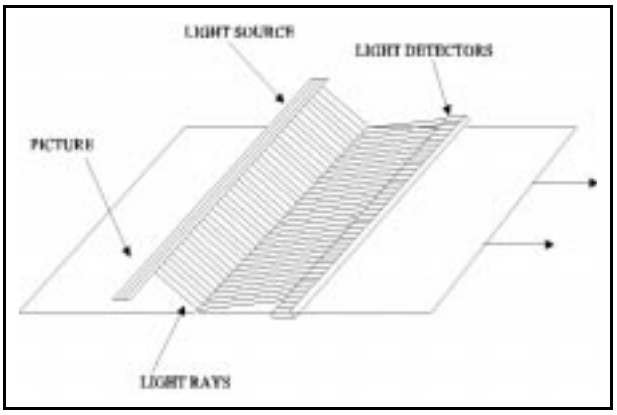


Figure 1.B.5 gives a higher level overview of image capture and display. Figure 1.B.6 shows a head writing electromagnetic pulses on a moving medium that is coated with ferrous material (disk or tape).

Figure 1.B.5. Image capture/display.

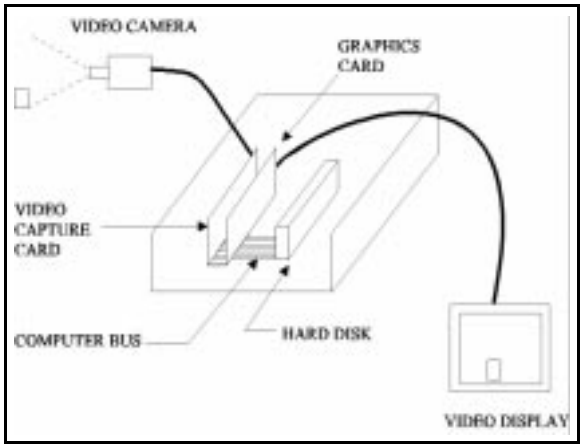


Figure 1.B.6. Reading/writing pulses.