

of technique and (optimized) implementation; there is even a Wikipedia entry and a discussion website supporting it. Then there are the VXL libraries (the Vision-*something*-Libraries, groan). This is ‘a collection of C++ libraries designed for computer vision research and implementation’. Finally, there is Adobe’s Generic Image Library (GIL), which aims to ease difficulties with writing imaging-related code that is both generic and efficient. Note that these are open source, but there are licences and conditions on use and exploitation.

A set of web links is shown in Table 1.2 for established freeware and commercial software image processing systems. Perhaps the best selection can be found at the general site, from the computer vision homepage software site at Carnegie Mellon (repeated later in Table 1.5).

Table 1.2 Software websites

Packages (freeware or student version indicated by *)		
General Site	Carnegie Mellon	http://www.cs.cmu.edu/afs/cs/project/cil/ftp/html/v-source.html (large popular index including links to research code, image processing toolkits, and display tools)
Visiquet (Khoros)	Accusoft	http://www.accusoft.com/
	Hannover University	http://www.tnt.uni-hannover.de/soft/imgproc/khoros/
AdOculos* (+ Textbook)	The Imaging Source	http://www.theimagingsource.com/
CVIPtools*	Southern Illinois University	http://www.ee.siue.edu/CVIPtools/
LaboImage*	Geneva University	http://cuiwww.unige.ch/~vision/LaboImage/labo.html
TN-Image*	Thomas J. Nelson	http://brneurosci.org/tnimage.html (scientific image analysis)
OpenCV	Intel	http://www.intel.com/technology/computing/opencv/index.htm and http://sourceforge.net/projects/opencvlibrary/
VXL	Many international contributors	http://vxl.sourceforge.net/
GIL	Adobe	http://opensource.adobe.com/gil/

1.5 Mathematical systems

Several *mathematical systems* have been developed. These offer what is virtually a word-processing system for mathematicians. Many are screen based, using a Windows system. The advantage of these systems is that you can transpose mathematics pretty well directly from textbooks, and see how it works. Code functionality is not obscured by the use of data structures, although this can make the code appear cumbersome. A major advantage is that the system provides low-level functionality and data visualization schemes, allowing the user to concentrate on techniques alone. Accordingly, these systems afford an excellent route to understand, and appreciate, mathematical systems before the development of application code, and to check that the final code works correctly.

1.5.1 Mathematical tools

Mathcad, Mathematica, Maple and Matlab are among the most popular of current tools. There have been surveys that compare their efficacy, but it is difficult to ensure precise comparison owing to the impressive speed of development of techniques. Most systems have their protagonists and detractors, as in any commercial system. Many books use these packages for particular subjects, and there are often handbooks as addenda to the packages. We shall use both Matlab and Mathcad throughout this text as they two very popular mathematical systems. We shall describe Matlab later, as it is different from Mathcad, although the aim is the same. The website links for the main mathematical packages are given in Table 1.3.

Table 1.3 Mathematical package websites

General		
Guide to available mathematical software	NIST	http://gams.nist.gov/
Vendors		
Mathcad	MathSoft	http://www.mathcad.com/
Mathematica	Wolfram Research	http://www.wolfram.com/
Matlab	Mathworks	http://www.mathworks.com/
Maple	Maplesoft	http://www.maplesoft.com/

1.5.2 Hello Mathcad, hello images!

Mathcad uses *worksheets* to implement mathematical analysis. The flow of calculation is very similar to using a piece of paper: calculation starts at the top of a document, and flows left to right and downwards. Data is available to *later* calculation (and to calculation to the right), but is not available to prior calculation, much as is this case when calculation is written manually on paper. Mathcad uses the Maple mathematical library to extend its functionality. To ensure that equations can migrate easily from a textbook to application, Mathcad uses a WYSIWYG (what you see is what you get) notation [its equation editor is not dissimilar to the Microsoft Equation (Word) editor]. Mathcad offers a compromise between many performance factors, and is available at low cost. There used to be a free worksheet viewer called Mathcad Explorer which operated in read-only mode, which is an advantage lost. An image processing handbook is available with Mathcad, but it does not include many of the more sophisticated feature extraction techniques.

Images are actually spatial data, data which is indexed by two spatial coordinates. The camera senses the *brightness* at a point with coordinates x,y . Usually, x and y refer to the horizontal and vertical axes, respectively. Throughout this text we shall work in *orthographic projection*, ignoring *perspective*, where real-world coordinates map directly to x and y coordinates in an image. The *homogeneous coordinate system* is a popular and proven method for handling three-dimensional coordinate systems (x , y and z , where z is depth). Since it is not used directly in the text, it is included as Appendix 2 (Section 10.3). The brightness sensed by the camera is transformed to a signal which is then fed to the A/D converter and stored as a value within the computer, referenced to the coordinates x,y in the image. Accordingly, a computer image is a

matrix of points. For a greyscale image, the value of each point is proportional to the brightness of the corresponding point in the scene viewed, and imaged, by the camera. These points are the picture elements, or *pixels*.

Consider, for example, the matrix of pixel values in Figure 1.13(a). This can be viewed as a surface (or function) in Figure 1.13(b), or as an image in Figure 1.13(c). In Figure 1.13(c) the brightness of each point is proportional to the value of its pixel. This gives the synthesized image of a bright square on a dark background. The square is bright where the pixels have a value around 40 brightness levels; the background is dark, and these pixels have a value near 0 brightness levels. This image is first given a label, `pic`, and then `pic` is allocated, `:=`, to the matrix defined by using the matrix dialogue box in Mathcad, specifying a matrix with eight rows and eight columns. The pixel values are then entered one by one until the matrix is complete (alternatively, the matrix can be specified by using a subroutine, but that comes later). Note that neither the background nor the square has a constant brightness. This is because noise has been added to the image. If we want to evaluate the performance of a computer vision technique on an image, but without the noise, we can simply remove it (one of the advantages of using synthetic images). The matrix becomes an image when it is viewed as a picture, in Figure 1.13(c). This is done either by presenting it as a surface plot, rotated by zero^{degrees} and viewed from above, or by using Mathcad's picture facility. As a surface plot, Mathcad allows the user to select a greyscale image, and the patch plot option allows an image to be presented as point values.

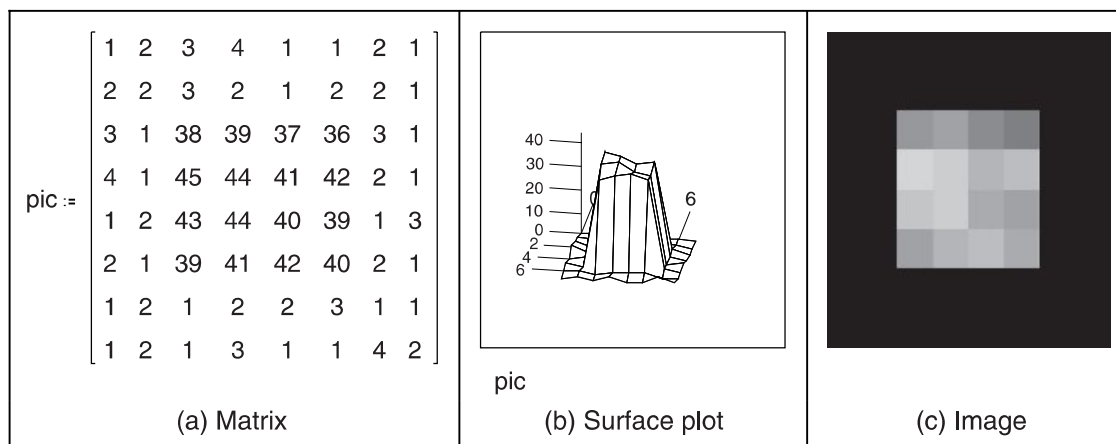


Figure 1.13 Synthesized image of a square

Mathcad stores matrices in row–column format. The coordinate system used throughout this text has x as the horizontal axis and y as the vertical axis (as conventional). Accordingly, x is the column count and y is the row count, so a point (in Mathcad) at coordinates x, y is actually accessed as `picy,x`. The origin is at coordinates $x = 0$ and $y = 0$, so `pic0,0` is the magnitude of the point at the origin and `pic2,2` is the point at the third row and third column and `pic3,2` is the point at the third column and fourth row, as shown in Code 1.1 (the points can be seen

```
pic2,2=38  pic3,2=45
rows(pic)=8  cols(pic)=8
```

Code 1.1 Accessing an image in Mathcad

in Figure 1.13a). Since the origin is at (0,0) the bottom right-hand point, at the last column and row, has coordinates (7,7). The number of rows and the number of columns in a matrix, the dimensions of an image, can be obtained by using the Mathcad rows and cols functions, respectively, and again in Code 1.1.

This synthetic image can be processed using the Mathcad programming language, which can be invoked by selecting the appropriate dialogue box. This allows for conventional for, while and if statements, and the earlier assignment operator which is := in non-code sections, is replaced by [back-arrow] in sections of code. A subroutine that inverts the brightness level at each point, by subtracting it from the maximum brightness level in the original image, is illustrated in Code 1.2. This uses for loops to index the rows and the columns, and then calculates a new pixel value by subtracting the value at that point from the maximum obtained by Mathcad's max function. When the whole image has been processed, the new picture is returned to be assigned to the label newpic. The resulting matrix is shown in Figure 1.14(a). When this is viewed as a surface (Figure 1.14b), the inverted brightness levels mean that the square appears dark and its surroundings appear white, as in Figure 1.14(c).

<pre>new_pic:=</pre>	<pre>for x∈0..cols(pic)-1 for y∈0..rows(pic)-1 newpicture_{y,x}←max(pic)-pic_{y,x} newpicture</pre>
----------------------	---

Code 1.2 Processing image points in Mathcad

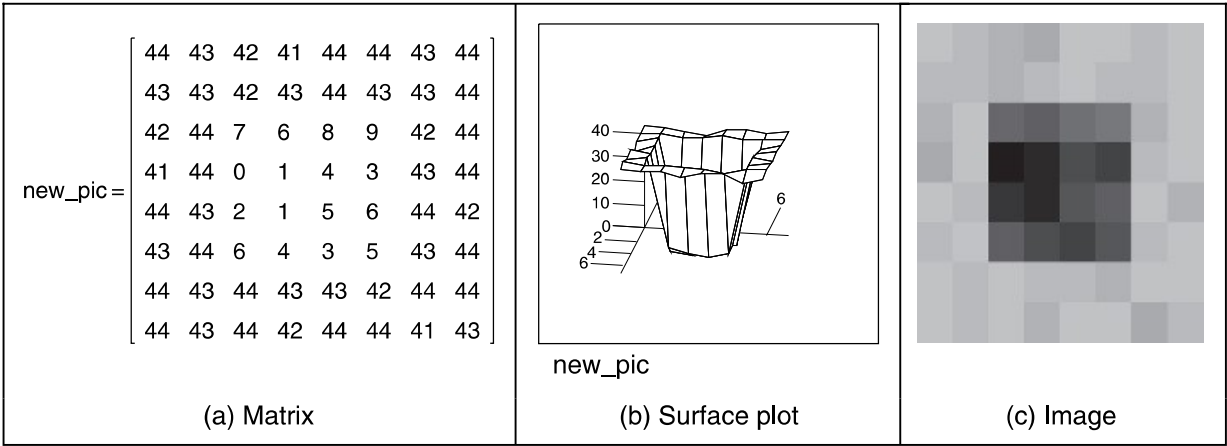


Figure 1.14 Image of a square after division

Routines can be formulated as *functions*, so they can be invoked to process a chosen picture, rather than restricted to a specific image. Mathcad functions are conventional; we simply add two arguments (one is the image to be processed, the other is the brightness to be added) and use the arguments as local variables, to give the add function illustrated in Code 1.3. To add a value, we simply call the function and supply an image and the chosen brightness level as the arguments.

<code>add_value(inplic,value):=</code>	<pre> for x∈0..cols(inplic)-1 for y∈0..rows(inplic)-1 newpicture_{y,x}←inplic_{y,x}+value newpicture </pre>
--	---

Code 1.3 Function to add a value to an image in Mathcad

Mathematically, for an image which is a matrix of $N \times N$ points, the brightness of the pixels in a new picture (matrix), $\mathbf{N}$, are the result of adding b brightness values to the pixels in the old picture, $\mathbf{O}$, given by:

$$\mathbf{N}_{x,y} = \mathbf{O}_{x,y} + b \qquad \forall x,y \in 1,N$$

(1.1)

Real images have many points. Unfortunately, the Mathcad matrix dialogue box only allows matrices that are 10 rows and 10 columns at most, i.e. a 10×10 matrix. Real images can be 512×512 , but are often 256×256 or 128×128 ; this implies a storage requirement for 262×144 , 65×536 and 16×384 pixels, respectively. Since Mathcad stores all points as high-precision, complex floating point numbers, 512×512 images require too much storage, but 256×256 and 128×128 images can be handled with ease. Since this cannot be achieved by the dialogue box, Mathcad has to be ‘tricked’ into accepting an image of this size. Figure 1.15 shows an image

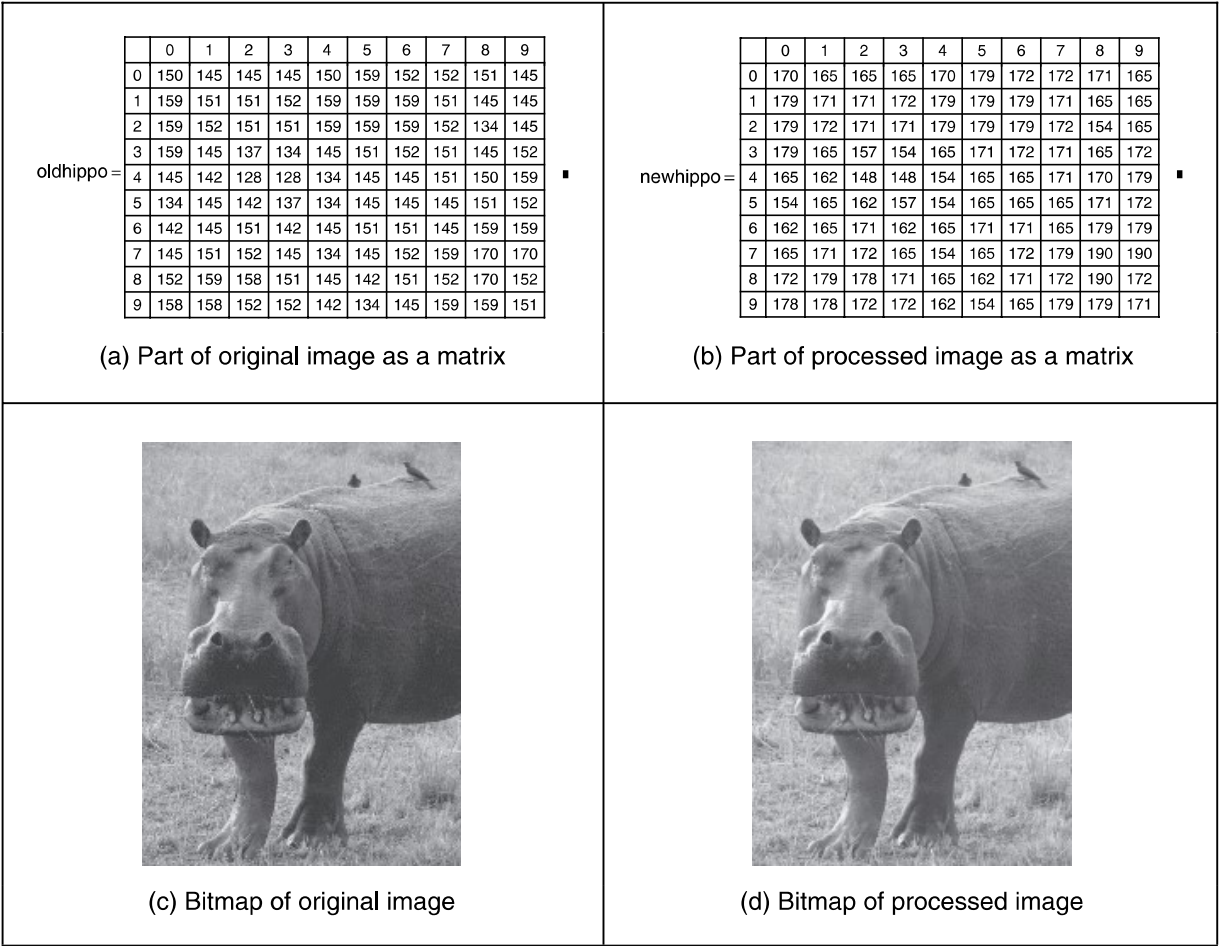


Figure 1.15 Processing an image

captured by a camera. This image has been stored in Windows bitmap (.BMP) format. This can be read into a Mathcad worksheet using the READBMP command (in capitals – Mathcad cannot handle readbmp), and is assigned to a variable. It is inadvisable to attempt to display this using the Mathcad surface plot facility, as it can be slow for images and requires a lot of memory. It is best to view an image using Mathcad’s picture facility or to store it using the WRITEBMP command, and then look at it using a bitmap viewer.

So, we can make an image brighter, by addition, by the routine in Code 1.3, via the code in Code 1.4. The result is shown in Figure 1.15. The matrix listings in Figure 1.15(a) and (b) show that 20 has been added to each point (these only show the top left-hand section of the image where the bright points relate to the grass; the darker points on, say, the ear cannot be seen). The effect will be to make each point appear brighter, as seen by comparison of the (darker) original image (Figure 1.15c) with the (brighter) result of addition (Figure 1.15d). In Chapter 3 we will investigate techniques that can be used to manipulate the image brightness to show the face in a much better way. For the moment, we are just seeing how Mathcad can be used, in a simple way, to process pictures.

```
oldhippo:=READBMP("hippo_orig")
newhippo:=add_value(hippo,20)
WRITEBMP("hippo_brighter.bmp"):=newhippo
```

Code 1.4 Processing an image

Mathcad was used to generate the image used to demonstrate the *Mach band* effect; the code is given in Code 1.5. First, an image is defined by copying the hippo image (from Code 1.4) to an image labelled mach. Then, the floor function (which returns the nearest integer less than its argument) is used to create the bands, scaled by an amount appropriate to introduce sufficient contrast (the division by 21.5 gives six bands in the image of Figure 1.4a). The cross-section and the perceived cross-section of the image were both generated by Mathcad’s X–Y plot facility, using appropriate code for the perceived cross-section.

mach:=	<pre>for x∈0..cols(mach)-1 for y∈0..rows(mach)-1 mach_{y,x}←brightness·floor($\frac{x}{\text{bar_width}}$) mach</pre>
--------	---

Code 1.5 Creating the image of Figure 4(a)

The translation of the Mathcad code into application can be rather prolix when compared with the Mathcad version by the necessity to include low-level functions. Since these can obscure the basic image processing functionality, Mathcad is used throughout this book to show how the techniques work. The translation to application code is perhaps easier via Matlab, as it offers direct compilation of the code. There is also an electronic version of this book, which is a collection of worksheets to help you to learn the subject; and an example Mathcad worksheet

is given in Appendix 1 (Section 9.1 for Mathcad; 9.2 for Matlab). You can download these worksheets from this book's website (<http://www.ecs.soton.ac.uk/~msn/book/>) and there is a link to the old Mathcad Explorer there too. You can then use the algorithms as a basis for developing your own application code. This provides a good way to verify that your code actually works: you can compare the results of your final application code with those of the original mathematical description. If your final application code and the Mathcad implementation are both correct, the results should be the same. Your application code will be much faster than in Mathcad, and will benefit from the graphical user interface (GUI) that you have developed.

1.5.3 Hello Matlab!

Matlab is rather different from Mathcad. It is not a WYSIWYG system, but instead it is more screen based. It was originally developed for matrix functions, hence the 'Mat' in the name. Like Mathcad, it offers a set of mathematical tools and visualization capabilities in a manner arranged to be very similar to conventional computer programs. In some users' views, a WYSIWYG system like Mathcad is easier to start with, but there are a number of advantages to Matlab, not least the potential speed advantage in computation and the facility for debugging, together with a considerable amount of established support. Again, there is an image processing toolkit supporting Matlab, but it is rather limited compared with the range of techniques exposed in this text. Its popularity is reflected in a book dedicated to use of Matlab for image processing (Gonzalez et al., 2003), by perhaps one of the subject's most popular authors.

Essentially, Matlab is the set of instructions that process the data stored in a workspace, which can be extended by user-written commands. The workspace stores the different lists of data and these data can be stored in a MAT file; the user-written commands are functions that are stored in M-files (files with extension .M). The procedure operates by instructions at the command line to process the workspace data using either one of Matlab's commands or your own commands. The results can be visualized as graphs, surfaces or images, as in Mathcad.

Matlab provides powerful matrix manipulations to develop and test complex implementations. In this book, we avoid matrix implementations in favour of a more C++ algorithmic form. Thus, matrix expressions are transformed into loop sequences. This helps students without experience in matrix algebra to understand and implement the techniques without dependency on matrix manipulation software libraries. Implementations in this book only serve to gain understanding of the techniques' performance and correctness, and favour clarity rather than speed.

The system runs on Unix/Linux or Windows and on Macintosh systems. A student version is available at low cost. There is no viewer available for Matlab; you have to have access to a system for which it is installed. As the system is not based around worksheets, we shall use a script which is the simplest type of M-file, as illustrated in Code 1.6. To start the Matlab system, type `MATLAB` at the command line. At the Matlab prompt (`>>`) type `chapter1` to load and run the script (given that the file `chapter1.m` is saved in the directory you are working in). Here, we can see that there are no text boxes and so comments are preceded by `%`. The first command is one that allocates data to our variable `pic`. There is a more sophisticated way to input this in the Matlab system, but that is not available here. The points are addressed in row-column format and the origin is at coordinates $y = 1$ and $x = 1$. So we access these points `pic3,3` as the third column of the third row and `pic4,3` as the point in the third column of the fourth row. Having set the display facility to black and white, we can view the array `pic` as a surface. When the surface (Figure 1.16a), is plotted, then Matlab has been made to pause until you press Return before moving on. Here, when you press Return, you will next see the image of the array (Figure 1.16b).

```

%Chapter 1 Introduction (Hello Matlab) CHAPTER1.M
%Written by: Mark S. Nixon

disp('Welcome to the Chapter1 script')
disp('This worksheet is the companion to Chapter 1 and is an
introduction.')
disp('It is the source of Section 1.4.3 Hello Matlab.')
disp('The worksheet follows the text directly and allows you to
process basic images.')

disp('Let us define a matrix, a synthetic computer image called
pic.')

pic =[1   2   3   4   1   1   2   1;
      2   2   3   2   1   2   2   1;
      3   1  38  39  37  36   3   1;
      4   1  45  44  41  42   2   1;
      1   2  43  44  40  39   1   3;
      2   1  39  41  42  40   2   1;
      1   2   1   2   2   3   1   1;
      1   2   1   3   1   1   4   2]

%Pixels are addressed in row-column format.
%Using x for the horizontal axis (a column count), and y for the
%vertical axis (a row count) then picture points are addressed as
%pic(y,x). The origin is at coordinates (1,1), so the point
%pic(3,3) is on the third row and third column; the point pic(4,3)
%is on the fourth row, at the third column. Let's print them:
disp('The element pic(3,3) is')
pic(3,3)
disp('The element pic(4,3)is')
pic(4,3)

%We'll set the output display to black and white
colormap(gray);
%We can view the matrix as a surface plot
disp('We shall now view it as a surface plot (play with the
controls to see it in relief)')
disp('When you are ready to move on, press RETURN')
surface(pic);
%Let's hold awhile so we can view it
pause;
%Or view it as an image
disp('We shall now view the array as an image')
disp('When you are ready to move on, press RETURN')
imagesc(pic);
%Let's hold awhile so we can view it
pause;

```

```

%Let's look at the array's dimensions
disp('The dimensions of the array are')
size(pic)

%now let's invoke a routine that inverts the image
inverted_pic=invert(pic);
%Let's print it out to check it
disp('When we invert it by subtracting each point from the
maximum, we get')
inverted_pic
%And view it
disp('And when viewed as an image, we see')
disp('When you are ready to move on, press RETURN')
imagesc(inverted_pic);
%Let's hold awhile so we can view it
pause;
disp('We shall now read in a bitmap image, and view it')
disp('When you are ready to move on, press RETURN')
face=imread('rhdark.bmp','bmp');
imagesc(face);
pause;
%Change from unsigned integer(uint8) to double precision so we can
process it
face=double(face);
disp('Now we shall invert it, and view the inverted image')
inverted_face=invert(face);
imagesc(inverted_face);
disp('So we now know how to process images in Matlab. We shall be
using this later!')

```

Code 1.6 Matlab script for Chapter 1

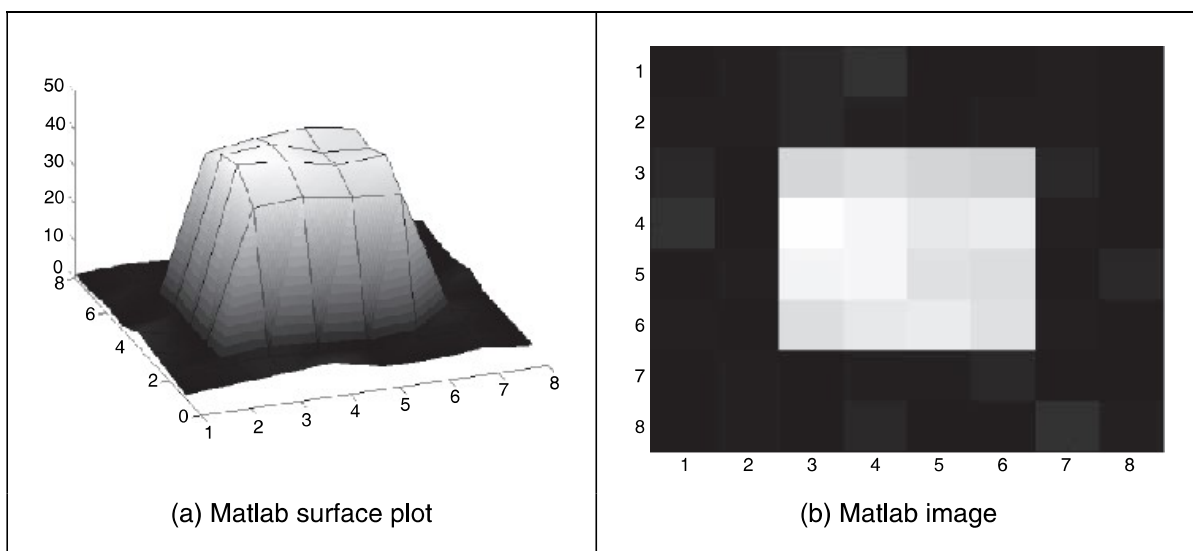


Figure 1.16 Matlab image visualization

We can use Matlab's own command to interrogate the data; these commands find use in the M-files that store subroutines. An example routine is called after this. This subroutine is stored in a file called `invert.m` and is a function that inverts brightness by subtracting the value of each point from the array's maximum value. The code is illustrated in Code 1.7. Note that this code uses `for` loops, which are best avoided to improve speed, using Matlab's vectorized operations (as in Mathcad). The whole procedure can be implemented by the command `inverted=max(max(pic))-pic`. One of Matlab's assets is a 'profiler' which allows you to determine exactly how much time is spent on different parts of your programs. There is facility for importing graphics files, which is actually rather more extensive (i.e. it accepts a wider range of file formats) than available in Mathcad. When images are used, this reveals that unlike Mathcad, which stores all variables as full precision real numbers, Matlab has a range of datatypes. We must move from the unsigned integer datatype, used for images, to the double precision datatype to allow processing as a set of real numbers. In these ways Matlab can and will be used to process images throughout this book. As with the Mathcad worksheets, there are Matlab scripts available at the website for online tutorial support of the material in this book; an abbreviated example worksheet is given in Appendix 1 (Section 9.2).

```
function inverted=invert(image)
%Subtract image point brightness from maximum
%
%Usage:[new image]=invert(image)
%
%Parameters: image-array of points
%
%Author: Mark S.Nixon
%get dimensions
[rows,cols]=size(image);

%find the maximum
maxi=max(max(image));

%subtract image points from maximum
for x=1:cols %address all columns
    for y=1:rows %address all rows
        inverted(y,x)=maxi-image(y,x);
    end
end
```

Code 1.7 Matlab function (`invert.m`) to invert an image

1.6 Associated literature

1.6.1 Journals and magazines

As in any academic subject, there are many sources of literature. The *professional magazines* include those that are more systems orientated, like *Vision Systems Design* and *Advanced Imaging*. These provide more general articles, and are often a good source of information about