Chapter 11
Plotting

Chapter 13
Images

Outline

11.1 Plotting in General

- Plotting is perhaps the most powerful aspect of MATLAB. Plots can be two-or three-dimensional with a wide variety of appearance to the plots
- All plots are hosted in a separate window, a **`figure`**
- A number of capabilities can be used with any plot:
  - Configuring the axes
  - Setting a color map
  - Turning on a grid
  - Title, axis labels and legends
  - Text annotations
  - Multiple plots on one figure

11.1 Plotting in General

- *axis*([*xl xu yl yu*]) overrides the automatic computation of the axis values.
- *colormap* <*specification*> establishes a sequence of colors. The legal specification values are listed in Appendix A. Examples of these are *autumn*, *bone*, *cool*, etc.
- *grid on* puts a grid on the plot.
- *hold on* hold the existing data on the figure to allow subsequent plotting call to be added to the current figure without erasing the existing plot; *hold off* redraws the current figure erasing the previous contents.
- *legend*(...) creates a legend box

11.1.2 Simple Functions for Enhancing Plots

- *text*(*x, y, {z}, <message>*) places the text provided at the specified location on a 2-D plot, or at the (x,y,z) location on a 3-D plot.

- *title*(…) places the text provided as the title of the current plot

- *view*(*az, el*) *sets the angle form which to view the plot.*

- *xlabel*(*…*) *sets the string provided as the label for the x-axis.*

- *ylabel*(*…*) *sets the string provided as the label for the y-axis.*

- *zlabel*(*…*) *sets the string provided as the label for the z-axis.*

11.1.3 Multiple Plots on One Figure

- Within the current figure, you can place multiple plots with the *subplot* command.

- The function subplot(r,c,n) divides the current figure into r rows and c column of equally spaced areas and then establishes the nth of these as the current figure:

```
...
subplot(3,2,1); % divides plotting are in 3x2 areas.
plot(x,sin(x));   % plots x vs. sin(x) in 1st. Window
...
```

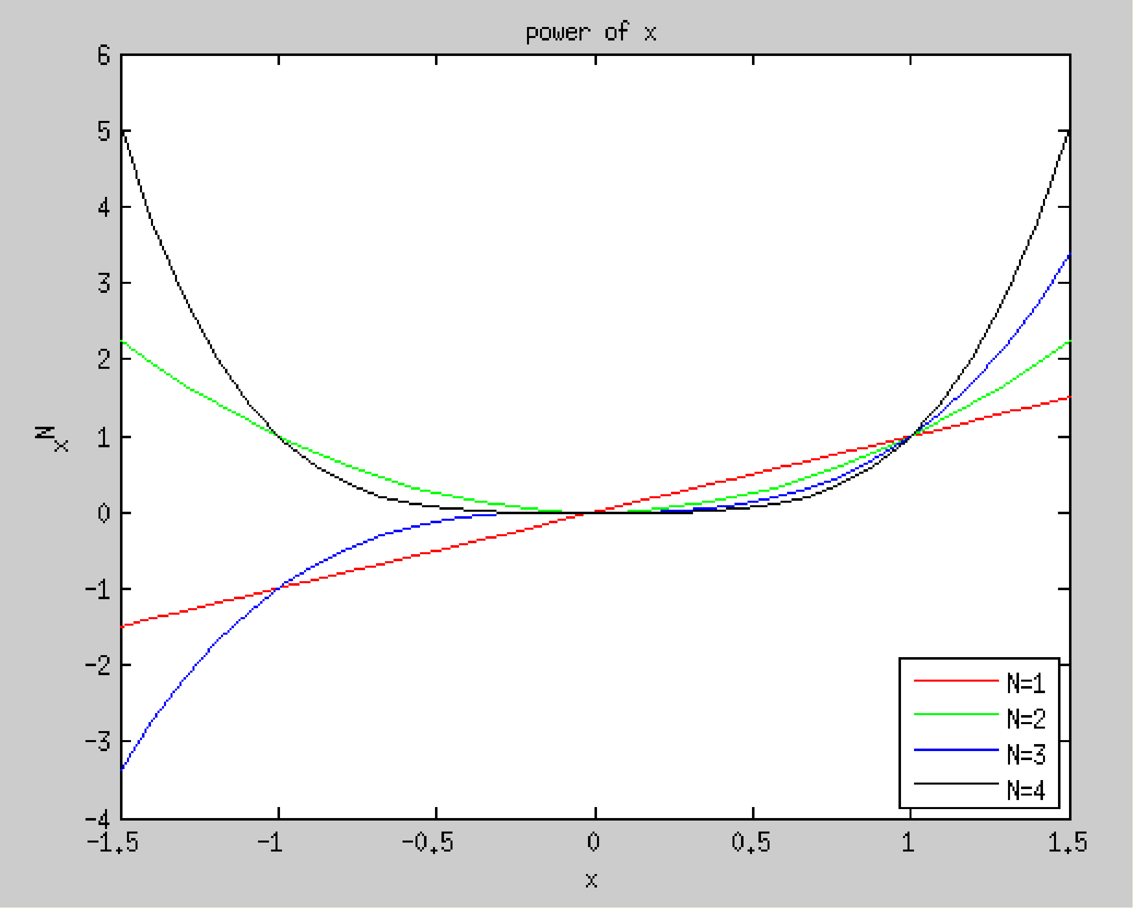
11.2 2-D Plotting

- The basic function to use for 2-D plots is *plot*(…). The normal use of this function is to give it three parameters, *plot*(*x,y,str*), where x and y are vectors of the same length, and str is a string containing one or more optional line color and style control characters.

11.2 2-D Plotting

```
x=linspace(-1.5,1.5,30);
clr='rgbk';
for pwr=1:4
    plot(x,x.^pwr,clr(pwr));
    hold on;
end
xlabel('x');
ylabel('x^N');
title('power of x');
legend({'N=1','N=2','N=3','N=4'},'Location','SouthEast')
```

11.2 2-D Plotting



11.2.4 Other 2-D Plot Capabilities

- You can also create some more exotic plots that are powerful methods for visualizing real data:
- *bar*(*x,y*) produces a bar graph with the values in y positioned at the horizontal locations in x.
- *fill*(*x,y,n*) produces a filled polygon defined by the coordinates in x and y.
- *hist*(*y,m*) produces a histogram plot with the values in y counted into bins defines by x.
- *pie*(*y*) makes a pie chart of the values in y.

11.3 3-D Plotting

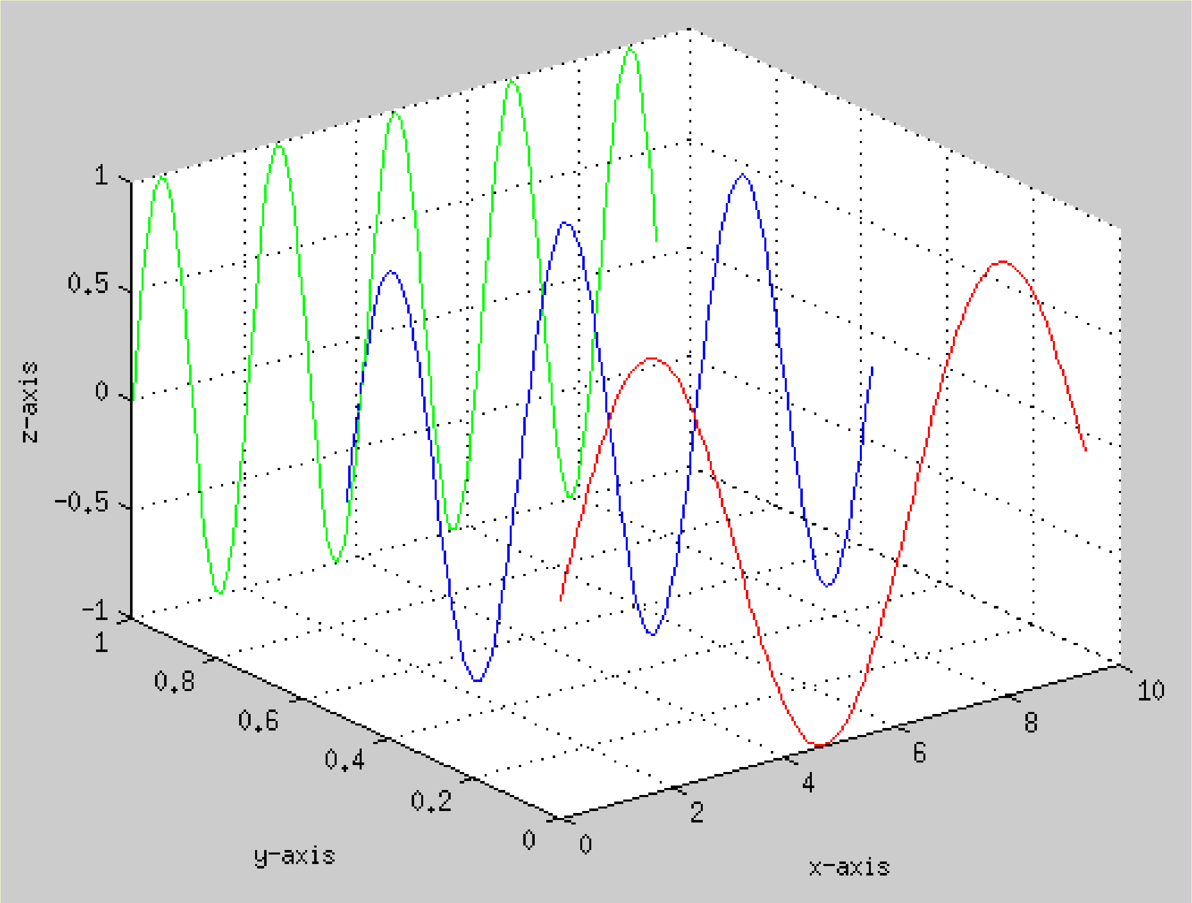
- The simplest method of 3-D plotting is to extend our 2-D plots by adding a set of z values.

- The function *plot3*(*x,y,z,str*) consumes three vectors of equal size and connect the points defined by those vectors in 3-D space. The optional str specifies color and/or line style.

11.3 3-D Plotting

```
x=0:0.1:3.*pi;
y1=zeros(size(x));
z1=sin(x);
z2=sin(2.*x);
z3=sin(3.*x);
y3=ones(size(x));
y2=y3./2;
plot3(x,y1,z1, 'r',x,y2,z2, 'b',x,y3,z3, 'g')
grid on
xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
```

11.3 3-D Plotting



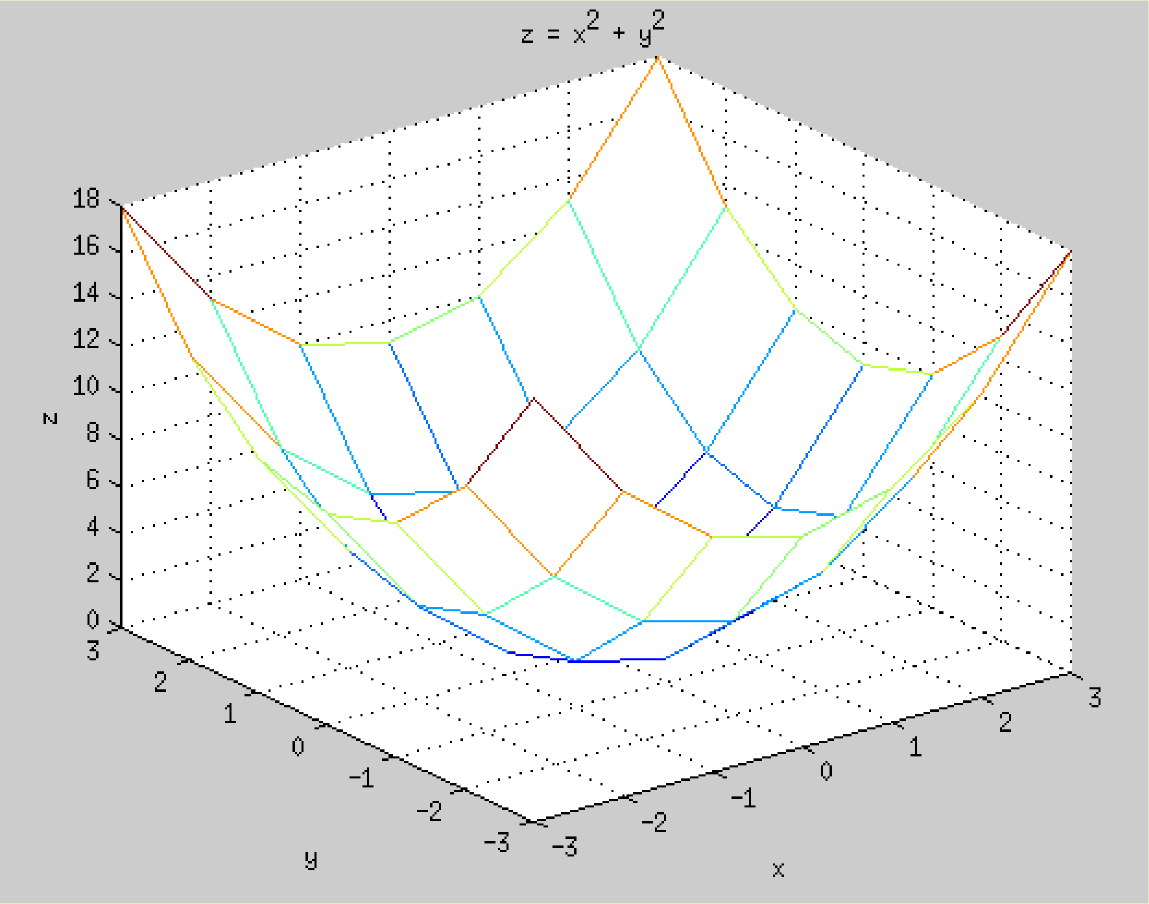
11.4 Surface Plots

- Three fundamental functions are used to create 3-D surface plots:
  - *meshgrid*(x,y) accepts the x and y vectors that bound the plaid and replicates the rows and columns appropriately to for 3-D plots.
  - *mesh*(xx,yy,zz) plots the surface as white facets outlined by colored lines.
  - *surf*(xx,yy,zz) plots the surface as colored facets outlines by black lines

11.4 Surface Plots

```
x=-3:3; y = x ;
[xx,yy]=meshgrid(x,y);
zz=xx.^2 + yy.^2;
mesh(xx,yy,zz)
axis tight
title('z = x^2 + y^2')
xlabel('x'),ylabel('y'),zlabel('z')
```

11.4 Surface Plots

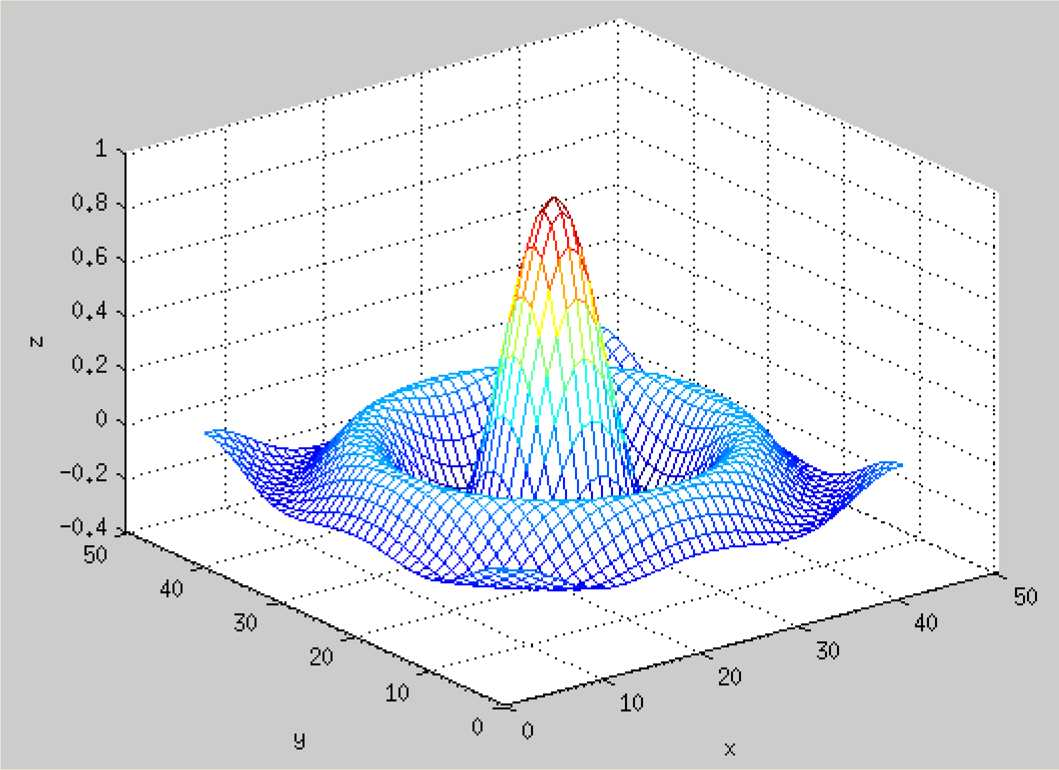


11.4 Surface Plots

- What the following code plot on the screen?

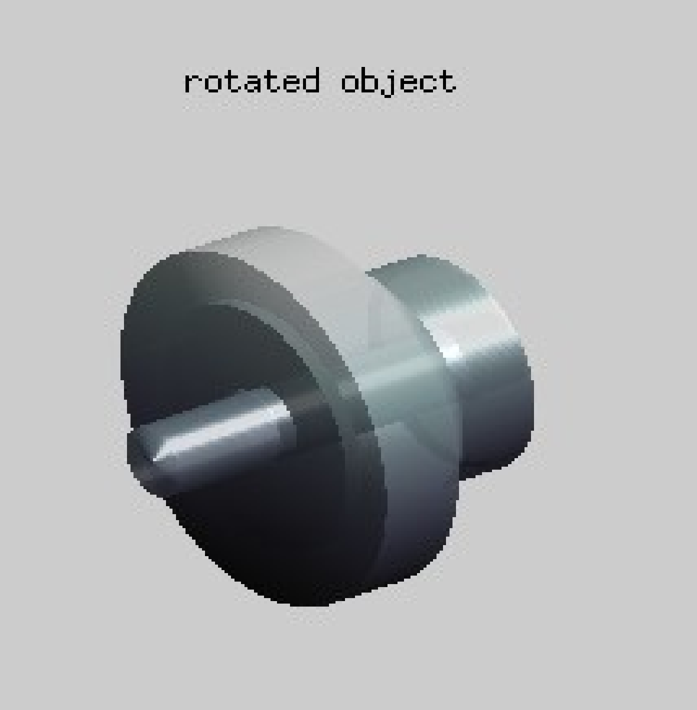
```
x=-10:.5:10;
y=x;
[X Y]=meshgrid(x,y);
R=sqrt(X.^2+Y.^2) + eps;
Z=sin(R)./R;
mesh(Z);
xlabel('x'),ylabel('y'),zlabel('z')
```

11.4 Surface Plots



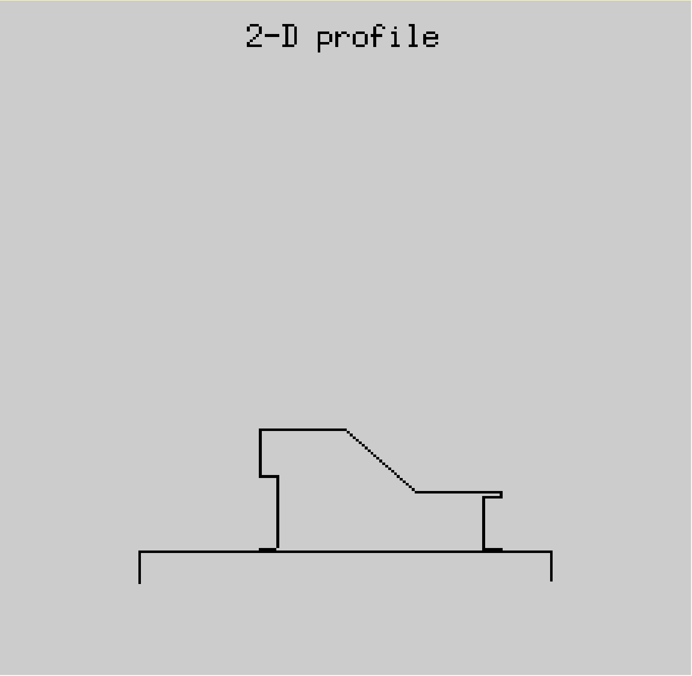
11.4 Rotating Discrete Functions

- Perform a rotation about the x-axis. After going through the meshgrid() to produce the a plaid, we run meshgrid().



11.4 Rotating Discrete Functions

- Complex surface plots can be drawn from simple 2-D profiles.
- Consider a 2-D profile of a fictitious machine part.



11.4 Rotating Discrete Functions

```
u = [0 0 3 3 1.75 1.75 2 2 1.75 1.75 3 4 ...
     5.25 5.25 5 5 5.25 5.25 3 3 6 6];
v = [0 .5 .5 .502 .502 .55 .55 1.75 1.75 ...
     2.5 2.5 1.5 1.5 1.4 1.4 ...
     .55 .55 .502 .502 .5 .5 0];
subplot(1, 2, 1)
plot(u, v, 'k')
axis ([-1 7 -1 3]), axis equal, axis off
title('2-D profile')
facets = 200;
subplot(1, 2, 2)
[xx tth] = meshgrid( u, linspace(0, 2*pi, facets) );
rr = meshgrid( v, 1:facets);
yy = rr .* cos(tth);
zz = rr .* sin(tth);
surf(xx, yy, zz);
shading interp
axis square, axis tight, axis off
colormap bone
lightangle(60, 45)
alpha(0.8)
title('rotated object')
```

11.5 Engineering Example – Visualizing Geographic Data

Problem:

- We are given two files of data: atlanta.txt, which represents the streets of Atlanta in graphical form, and ttimes.txt, which give the travel times between Atlanta suburbs and the city center.

- We are asked to present these data in a manner that will help to visualize and validate the data.

Analyze the Data:

1. Determine the file format. Since there are no strings in the file, it should be suitable to be read using the built-in dlmread(...) function.

11.5 Engineering Example – Visualizing Geographic Data

Analyze the Data:

2. Discern the street map file content. The atlanta.txt file contains columns with the following information: columns 3-6 are pairs of latitude, longitude coordinates (x1,000,000) for ends of streets, column 7 contains number in the range 1-6 which indicates the type of street:

53423.00	53343.00	-84546100.00	33988160.00	-84556050.00	33993620.00	1.00	3025.00
54528.00	53351.00	-84546080.00	33988480.00	-84558400.00	33995480.00	1.00	3025.00
130081.00	128176.00	-84243880.00	33780010.00	-84249980.00	33800840.00	1.00	3025.00
130105.00	128192.00	-84243590.00	33780060.00	-84249740.00	33800840.00	1.00	3025.00
58150.00	71086.00	-84509920.00	33944340.00	-84517200.00	33958190.00	1.00	3025.00

...

11.5 Engineering Example – Visualizing Geographic Data

Analyze the Data:

3. Discern the travel time content. The ttimes.txt contains columns with the following information: columns 1and 2 are used to build a plaid (much like the result of meshgrid()), columns 4,5 represent latitude/longitude (x1,000,000), and column 6 represents the z values of the plaid (it would be reasonable to assume that it represents time in minutes).

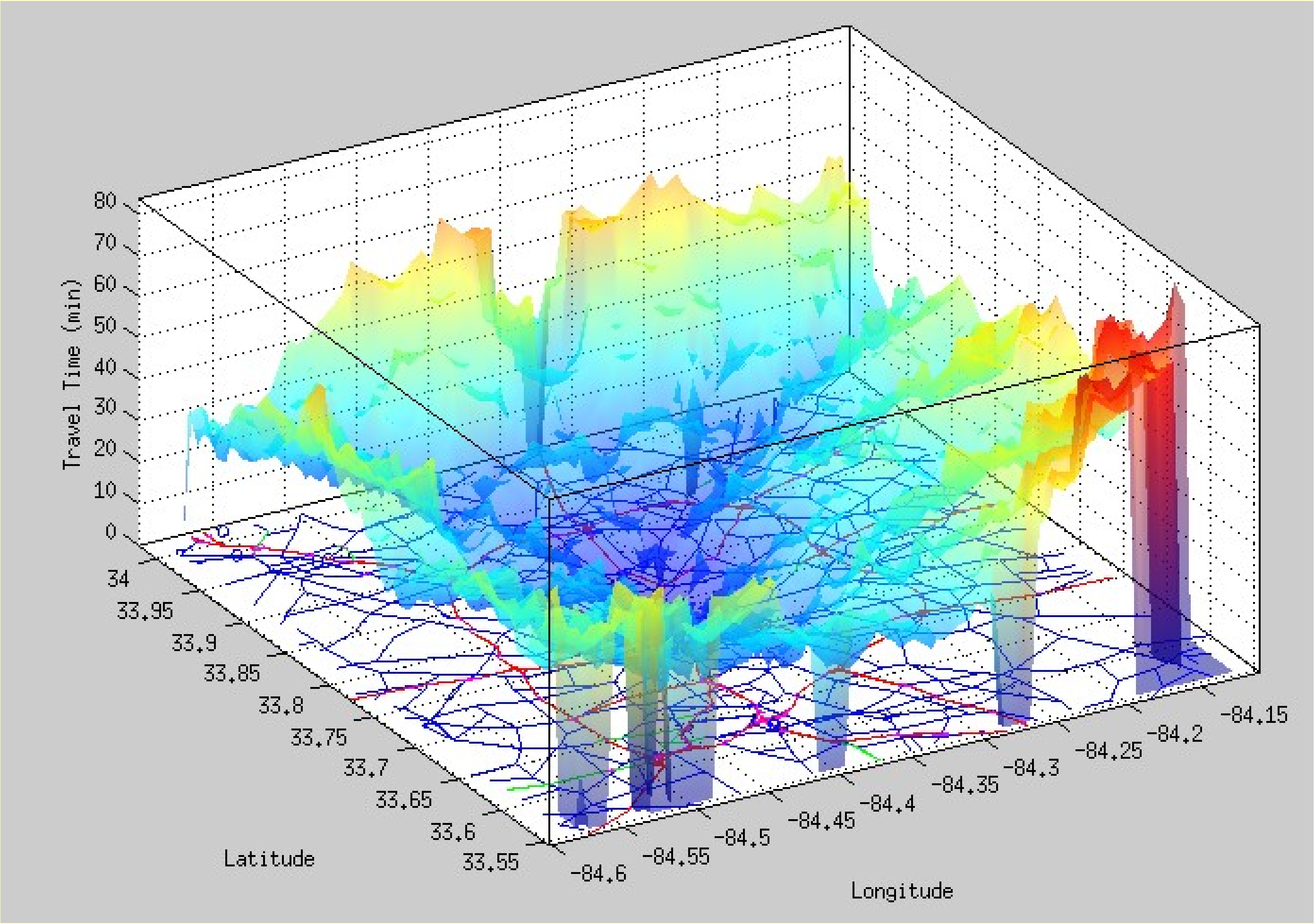
1	1	76	-84575725	33554573	14.34
1	2	77	-84569612	33554573	0
1	3	78	-84563499	33554573	0
1	4	79	-84557387	33554573	0
1	5	80	-84551274	33554573	51.66

…

11.5 Engineering Example – Visualizing Geographic Data

```
raw = dlmread('atlanta.txt');
streets = raw(:,3:7);
[rows,cols] = size(streets)
colors = 'rgbkcmo';
for in = 1:rows
    x = streets(in,[1 3])/1000000;
    y = streets(in,[2 4])/1000000;
    col = streets(in,5);
    col(col < 1) = 7;
    col(col > 6) = 7;
    plot(x,y,colors(col)); hold on
end
% plot the travel times
tt = dlmread('ttimes.txt');
[rows,cols] = size(tt)
for in = 1:rows
    r = tt(in, 1); c = tt(in, 2);
    xc(r,c) = tt(in, 4)/1000000;
    yc(r,c) = tt(in, 5)/1000000;
    zc(r,c) = tt(in, 6);
end
surf(xc, yc, zc)
shading interp
alpha(.5)
grid on; axis tight;
xlabel('Longitude'); ylabel('Latitude');
zlabel('Travel Time (min)'); view(-30, 45);
```

11.5 Engineering Example – Visualizing Geographic Data



Outline

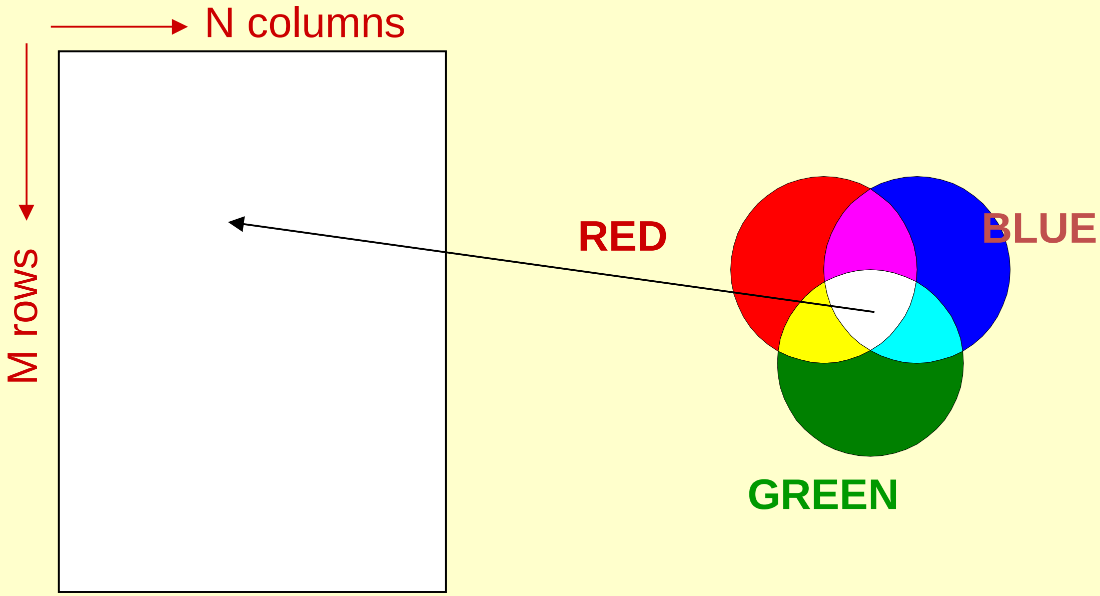
Introduction

- The graphical techniques we have seen so far have been 2-D and 3-D plots. These presentations are easily generated when we have a mathematical model of the data.
- However, many sensors observing the world do not have that underlying model of the data (which we cal images), leaving the interpretation of the images to the human observer.

13.1 Nature of an Image

- An image is a 2-D sheet on which the color at any point can have essentially infinite variability.
- We can represent any image as a 2-D, MxN array of points usually referred to as picture elements, or pixels.
- Each pixel is “painted” by blending variable amounts of the three primary colors: Red (R), Green (G), and Blue (B).
- The color resolution is measured by the number of bits in the words containing the red, green, and blue (RGB) components.

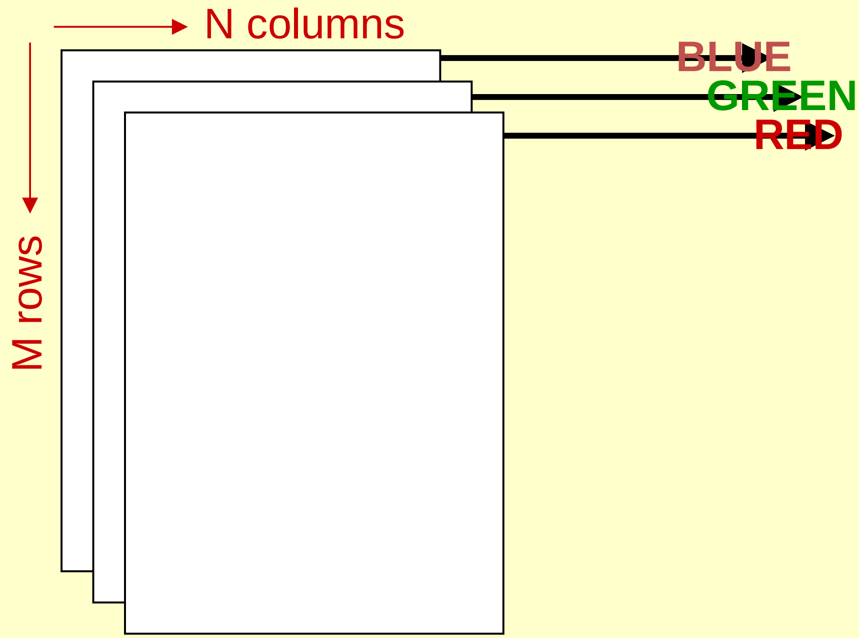
13.1 Nature of an Image



13.2 Image Types

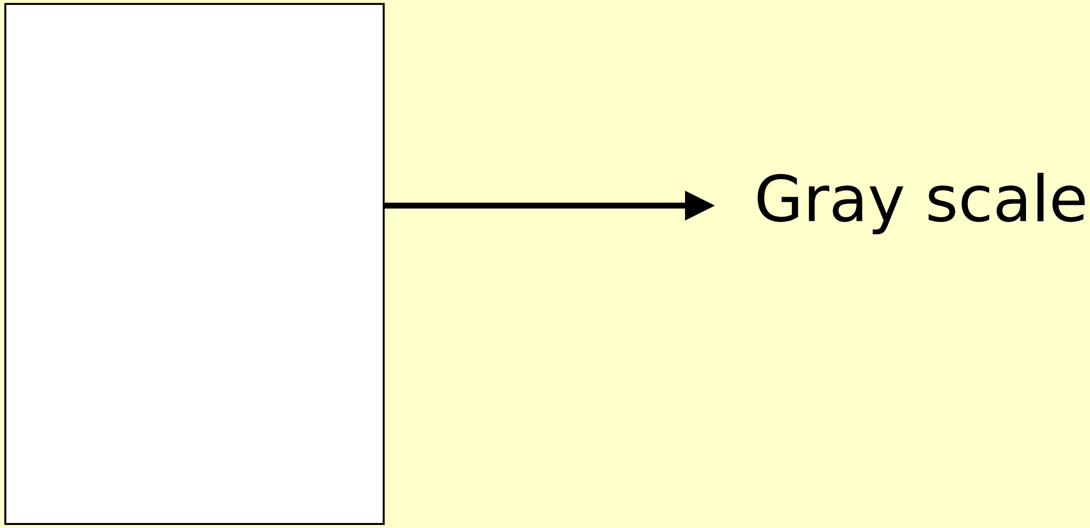
- Images are provided in a wide variety of formats.
- According to MATLAB documentation, it recognizes files in:
  TIFF, PNG, HDF, BMP, JPEG, GIG, and others.
- True color images are stored in a MxNx3 array where every pixel is directly stored as uint8 values in three layers of the 3-D array:
  The first layer contains the red values.
  Second layer contains the green values.
  Third layer contains the red values.
- Gray scale images only save the black-to-white intensity value for each pixel as a single uint8 values rather than three values.

13.2 Image Types



True Color

13.2 Image Types

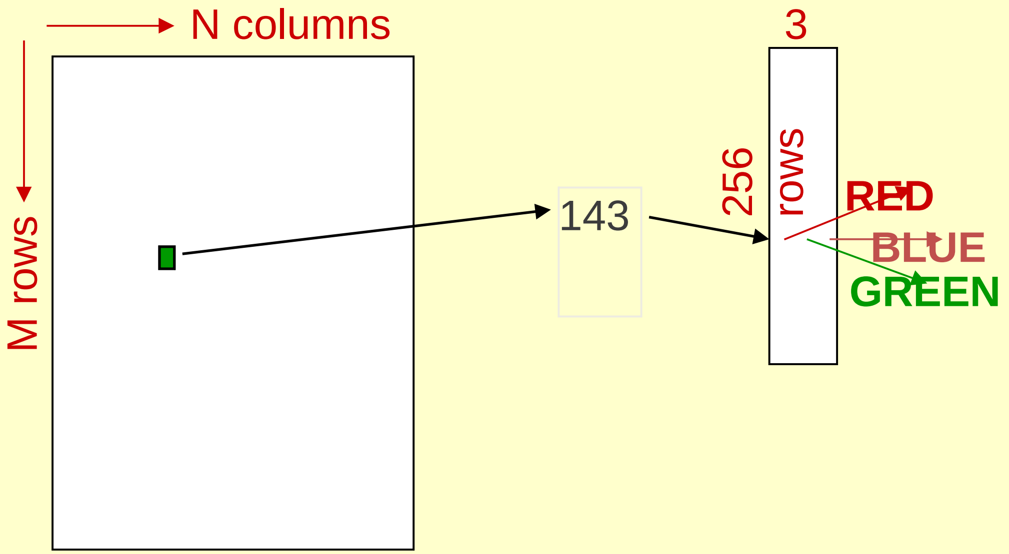


Black-and-White Color

13.2.3 Color Mapped Images

- Color mapped, or indexed,images keep a separate map either 256 items or up to 32,768 items long.
- This is done for maximum economy of memory. Therefore, each item in the color map contains the red, blue, and green values of a color, respectively.
- As illustrated in the following figure, a certain pixel index might contain the value 143. The color to be shown at that pixel location would be the 143rd color set (RGB) on the color map.

13.2 Image Types



Bit Mapped

13.3 Reading, Displaying, and Writing Images

- Image files are stored in many different formats
- We will concern ourselves only with .jpg files.
- Note, however, that .jpg files use a mathematical compression technique that cannot guarantee that the uncompressed image matches the original.

13.3 Reading, Displaying, and Writing Images

- MATLAB uses one image reading function, imread(...) for all image file types:

img = *imread*(file) reads a file

imshow(img) or *image*(img) displays the image

imwrite(img, file, '.jpg') writes a modified image to a file in JPEG format.

13.4 Operating on Images

- Since images are stored as arrays, we can employ the normal operations of creating, manipulation, slicing, and concatenation.
- We can uniformly shrink or stretch an array (image) to match an exact size.
- Assume that the horizontal size is good, but we want to stretch or shrink the image vertically.

13.4.1 Stretching or Shrinking Images

- We can use the following commands to shrink the image:

  rowv=linespace(1,rows,nrows) generates new row indices

  rowv=round(rowv) rounds row numbers

  newpicture=picture(rowv,cols,:) generate a re-sampled image

13.4.2 Color Masking

- Consider an image that is 2400x1600 JPEG image that can be taken with any good digital camera.

- The appearance of the Vienna garden is somewhat marred by the fact that the sky is gray, not blue. Fortunately, we have a picture of a cottage with nice, clear blue sky.

13.4.2 Color Masking

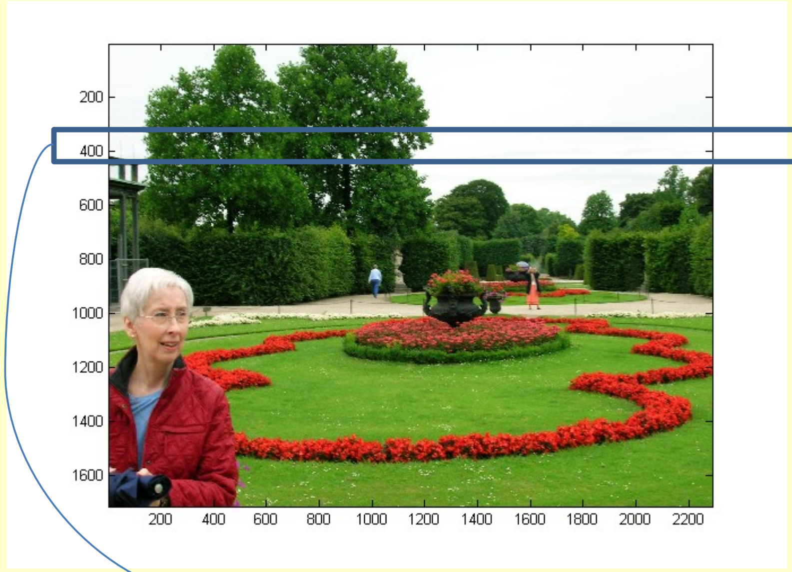
13.4.2 Color Masking

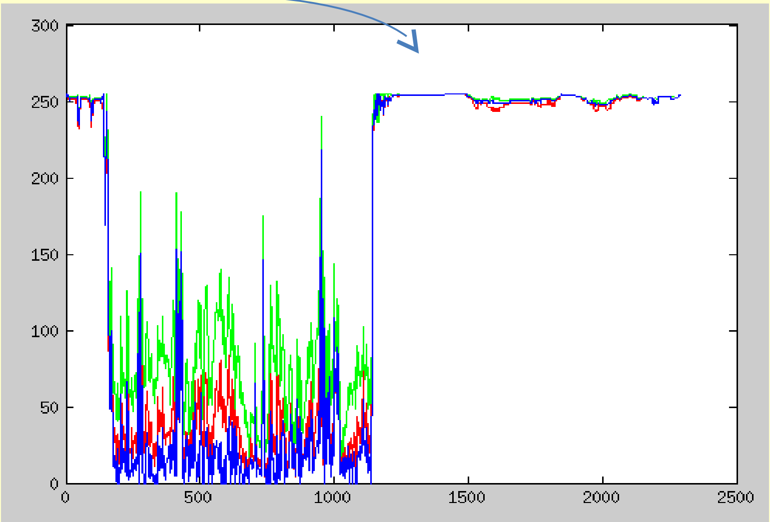
- So the solution for this problem is to replace the gray sky in the Vienna garden with blue sky from the cottage picture.
- To do this we need to explore the Vienna picture to determine how to distinguish the gray sky from the rest of the picture.
- The solution is to choose a representative row in the image that includes some sky and look at the red, blue, and green values for sky pixels.

13.4.2 Color Masking

- So the solution for this problem is to replace the gray sky in the Vienna garden with blue sky from the cottage picture.
- To do this we need to explore the Vienna picture to determine how to distinguish the gray sky from the rest of the picture.
- The solution is to choose a representative row in the image that includes some sky and look at the red, blue, and green values for sky pixels.

13.4 Operating on Images





13.4.2 Color Masking

- As we examine the plots we see that the red, green, and blue values for the open sky are all around 250 because the sky is almost white.
- We could decide for example to define the sky as all those pixels where the red, blue, and green values are all above a chosen threshold, and could safely set that threshold at 160.

13.4.2 Color Masking

- However, it would be unfortunate to turn the hair of the lady blue, and there are fountains and walkways that might also logically appear to be the sky.

- We can prevent this embarrassment to limiting the color replacement to the upper portion of the picture above row 700.

13.4.2 Color Masking

- So we are ready to create the code that will replace the gray sky with blue:

```
v=imread('Vienna.jpg'); w=imread('Witney.jpg');
image(w); figure;
thres=160;
layer=(v(:,:,1)>thres) & (v(:,:,2)>thres) & (v(:,:,3)>thres);
mask(:,:,1)=layer; mask(:,:,2)=layer; mask(:,:,3)=layer;
mask(700:end,:,:)=false;
nv=v; nv(mask)=w(mask);
image(nv);
imwhite(nv,'newVienna.jpg','jpg');
```

13.4.2 Color Masking

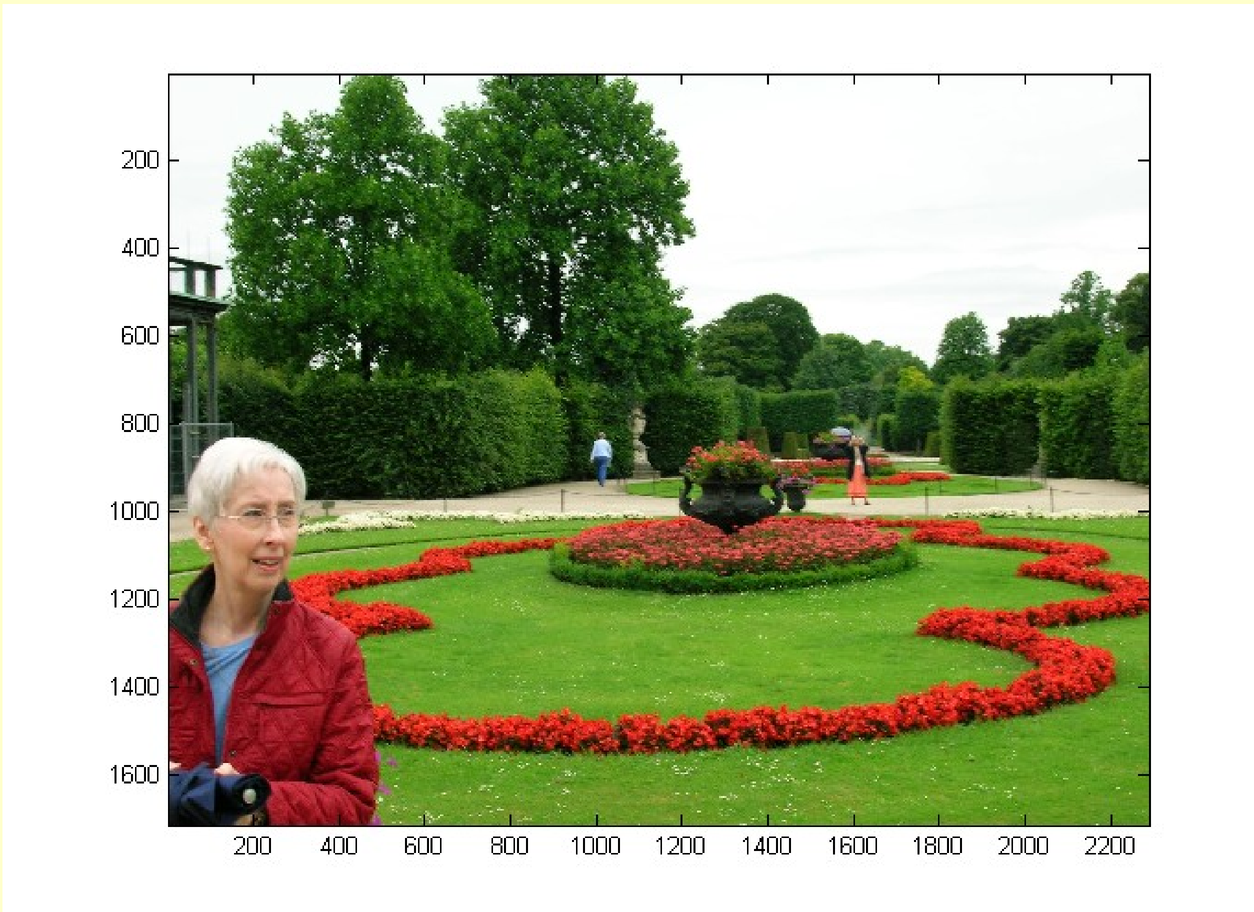
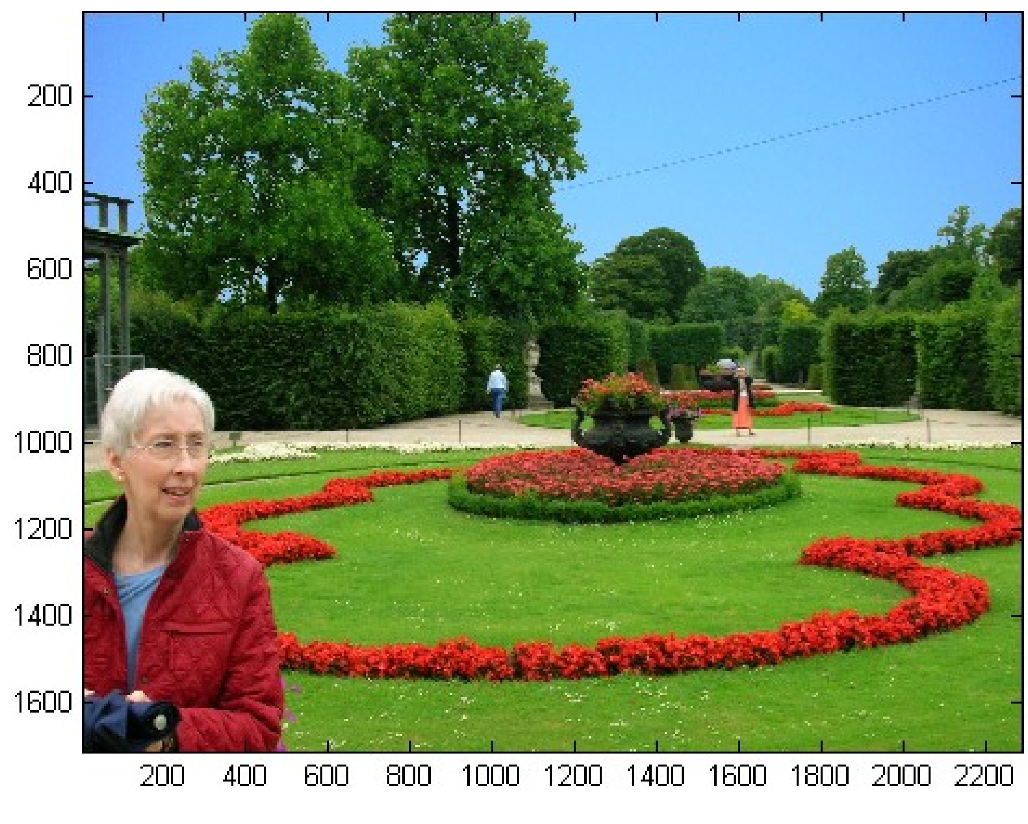


Figure 13.7


200
400
600
800
1000
1200
1400
1600
200
400
600
800
1000
1200
1400
1600
1800
2000
2200

Let’s write some Code …

13.5 Engineering Example – Detecting Edges

- While images are powerful methods for delivering information to the human eye, they have limitations when being used by computer programs.
- Our eyes and brain have astonishing ability to interpret the content of an image, while computer programs need a lot of help.

13.5 Engineering Example – Detecting Edges

- One operation commonly performed to reduce the complexity of an image is *edge detection*.
- The image is replaced by a very small number of points that mark the edges of “interesting artifacts”.
- The key element of the edge detection algorithm is the ability to determine unambiguously whether a pixel is part of the object of interest or not.

13.5 Engineering Example – Detecting Edges

- Edge detection using the Sobel method

The magnitude of the vector Δf is denoted as,

$$\Delta f = mag(\Delta f) = [G_x^2 + G_y^2]^{1/2}$$

where Gx is for x direction and Gy for y direction.

The sobel masks (3x3):

For x-Direction:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

For Y-direction:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

13.5 Engineering Example – Detecting Edges

13.5 Engineering Example – Detecting Edges

```
A=imread('Lena.JPG');
figure; imagesc(A); title('Originalimage'); colormap(gray);
B=A(:,:,1);
C=double(B);
[r,c]=size(C);

for i=1:r-2
  for j=1:c-2
    %Sobel mask for x-direction:
    Gx=((2*C(i+2,j+1)+C(i+2,j)+C(i+2,j+2))-(2*C(i,j+1)+C(i,j)+C(i,j+2)));

    %Sobel mask for y-direction:
    Gy=((2*C(i+1,j+2)+C(i,j+2)+C(i+2,j+2))-(2*C(i+1,j)+C(i,j)+C(i+2,j)));

    %The gradient of the image
    B(i,j)=sqrt(Gx.^2+Gy.^2);

  end
end
figure; imagesc(B); title('Sobel gradient'); colormap(gray);
```

13.5 Engineering Example – Detecting Edges