EECE 4353 Image Processing

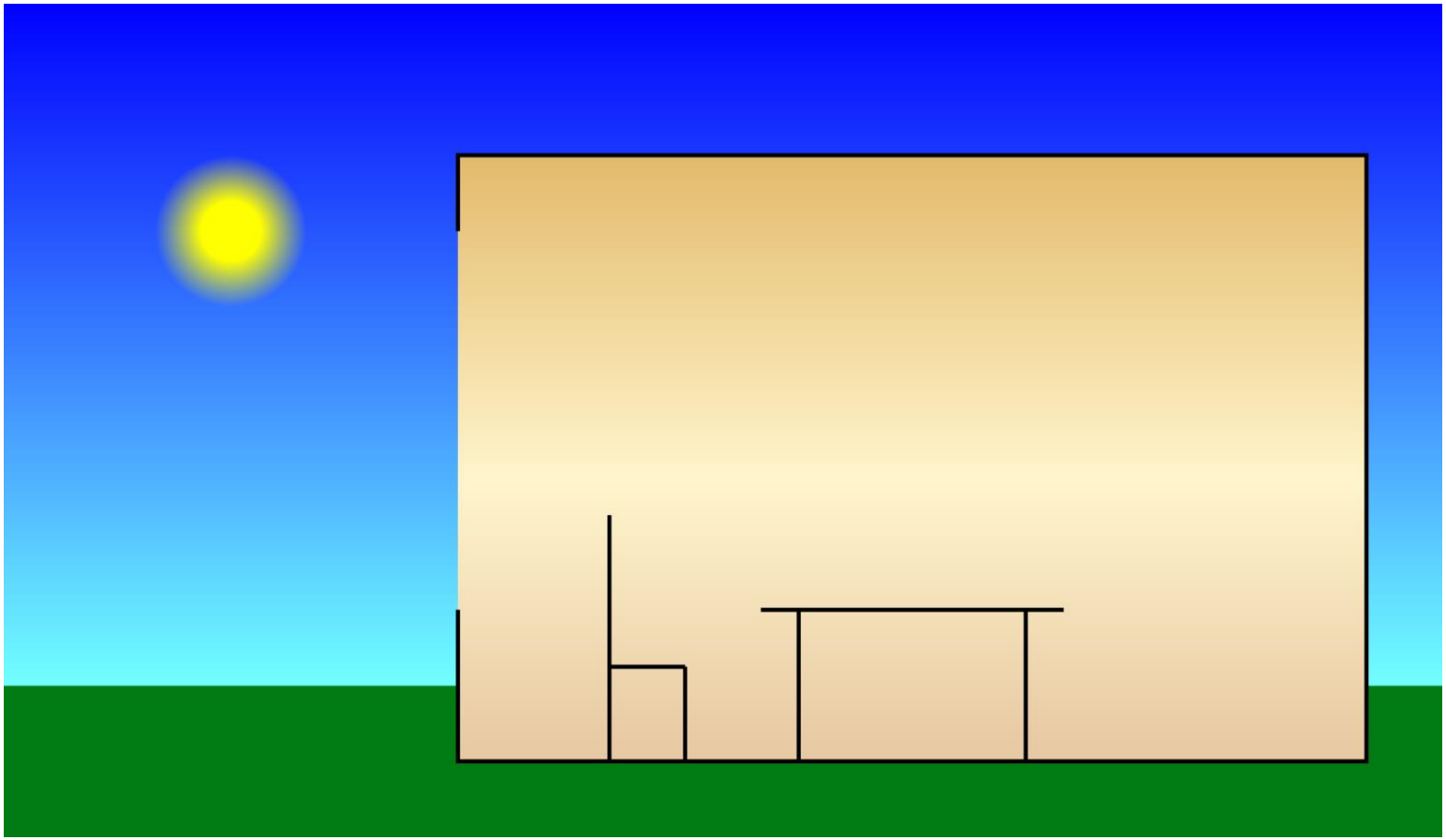
Lecture Notes: Light Scattering, Image Formation, Digital Images, and MATLAB

Richard Alan Peters II

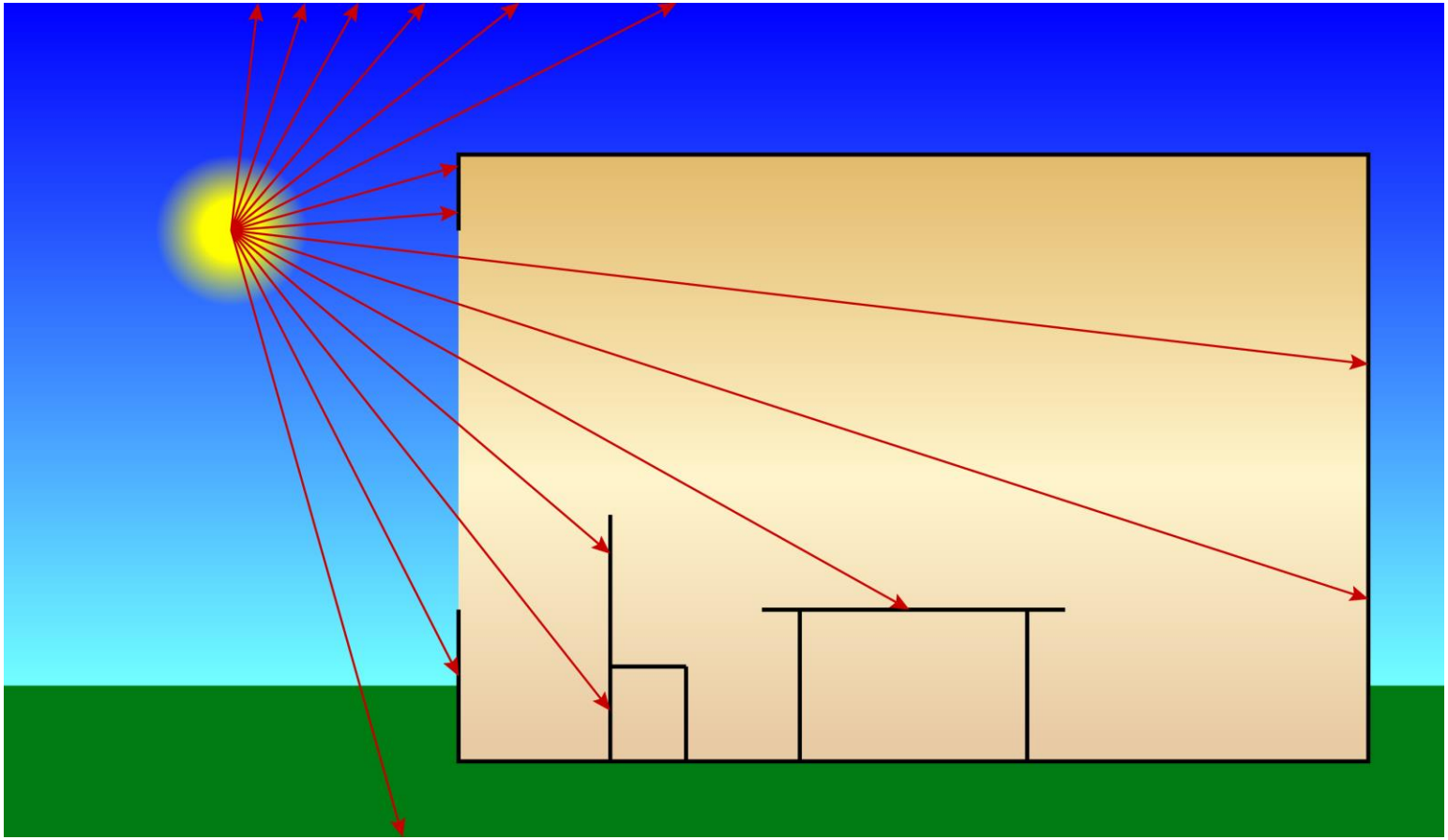
Department of Electrical Engineering and Computer Science

Fall Semester 2016

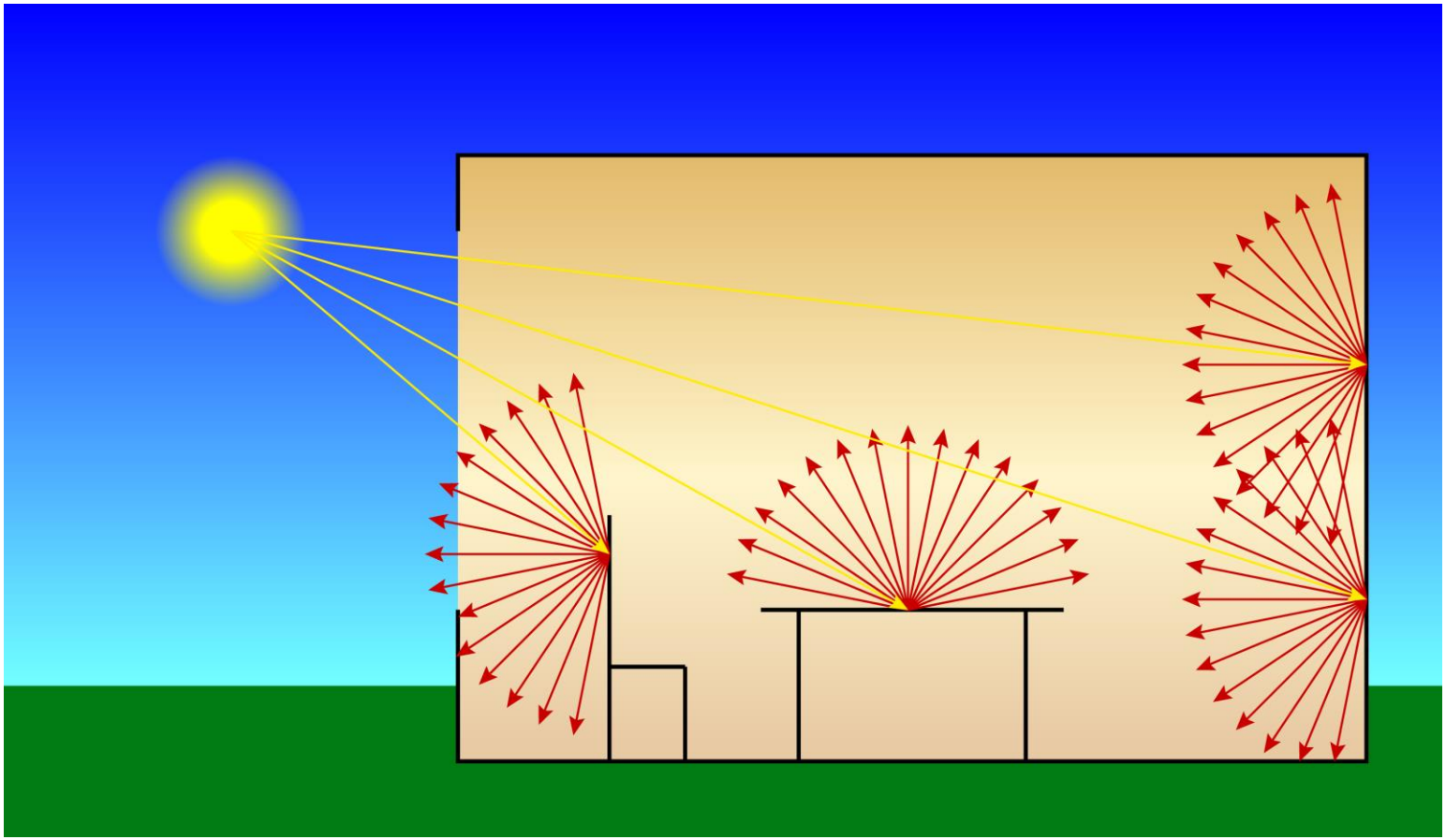
Light from a source …



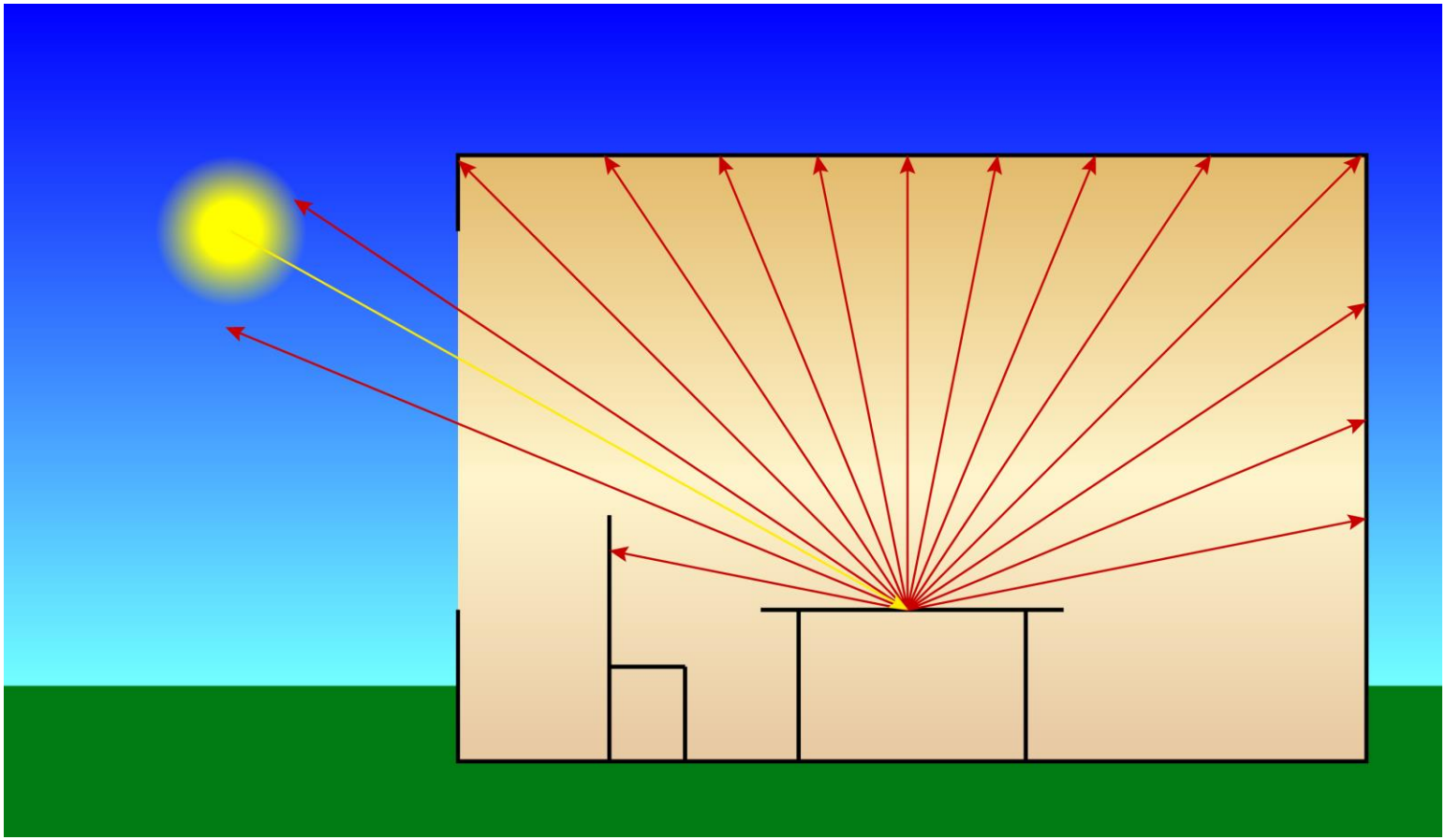
… illuminates a scene.



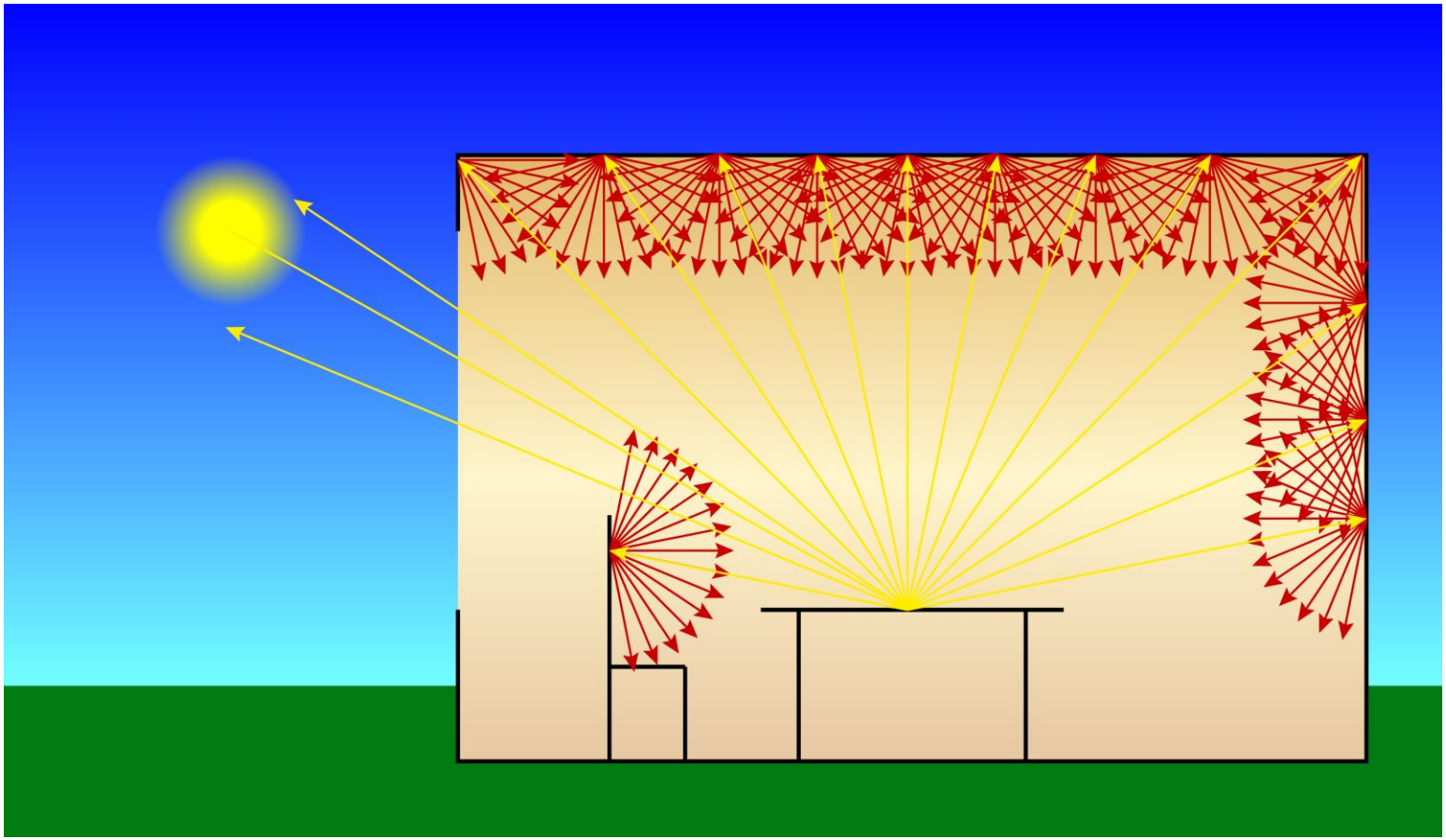
Each surface scatters light …



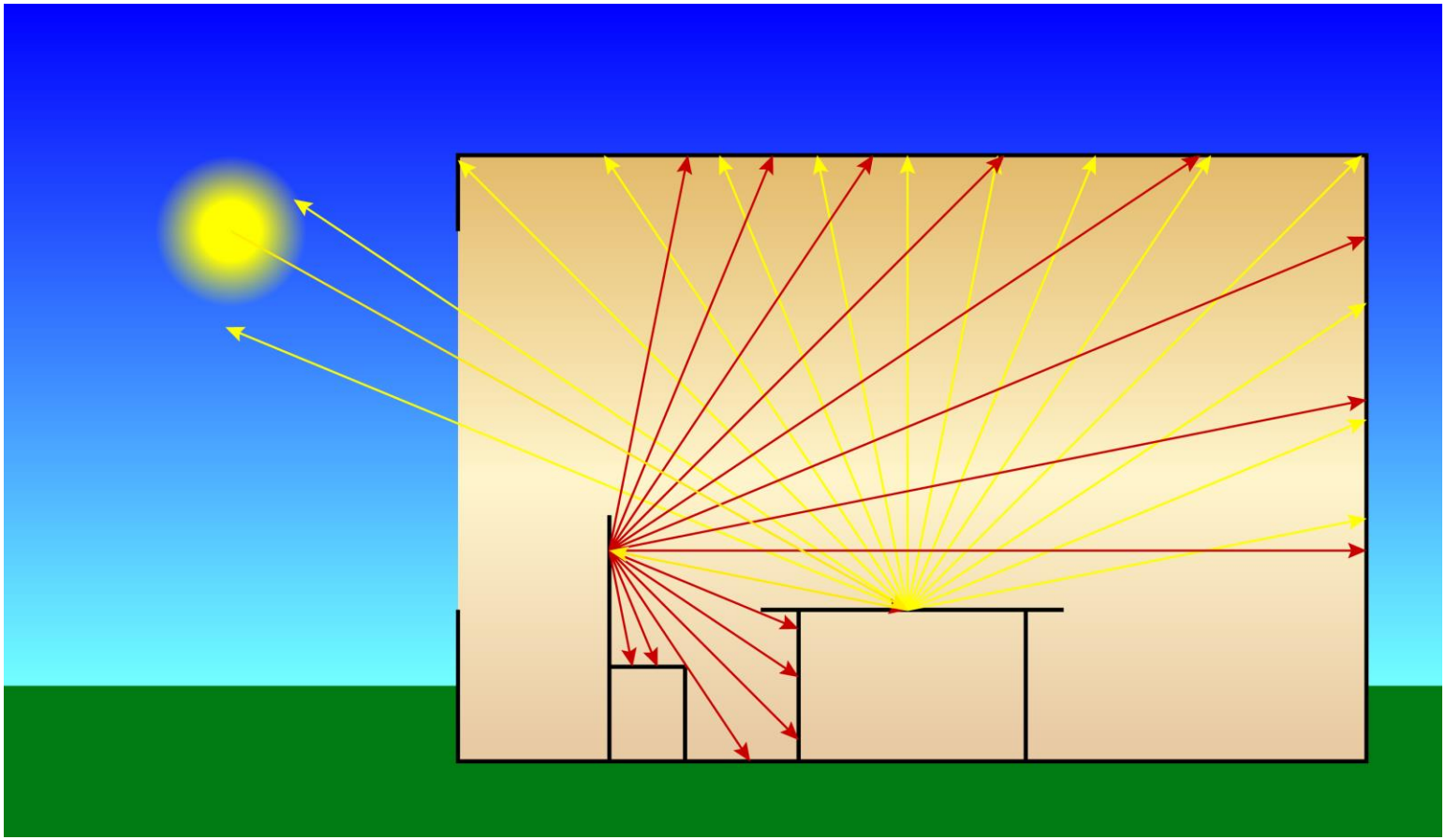
… that illuminates other surfaces …



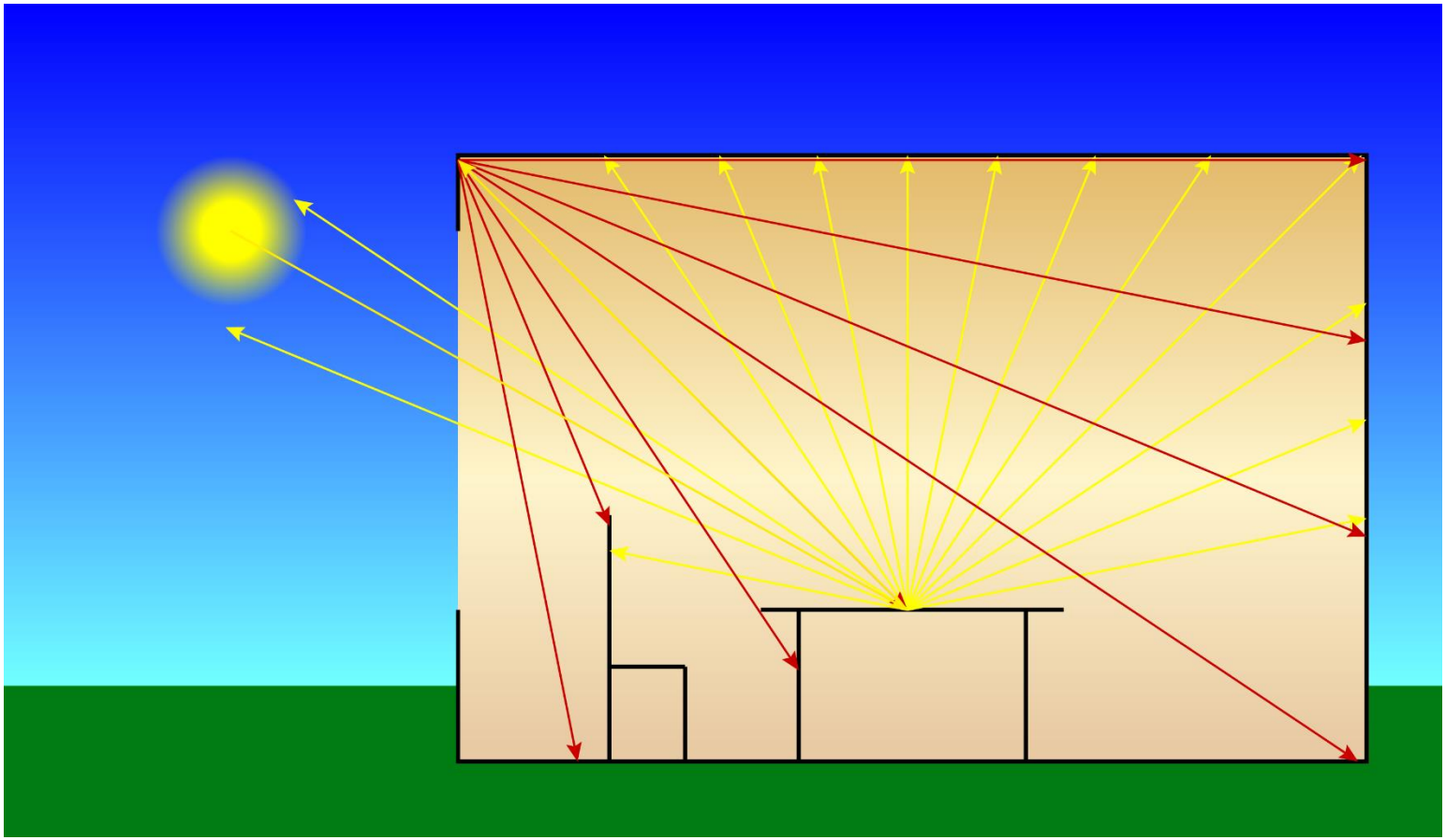
.. that scatter light.



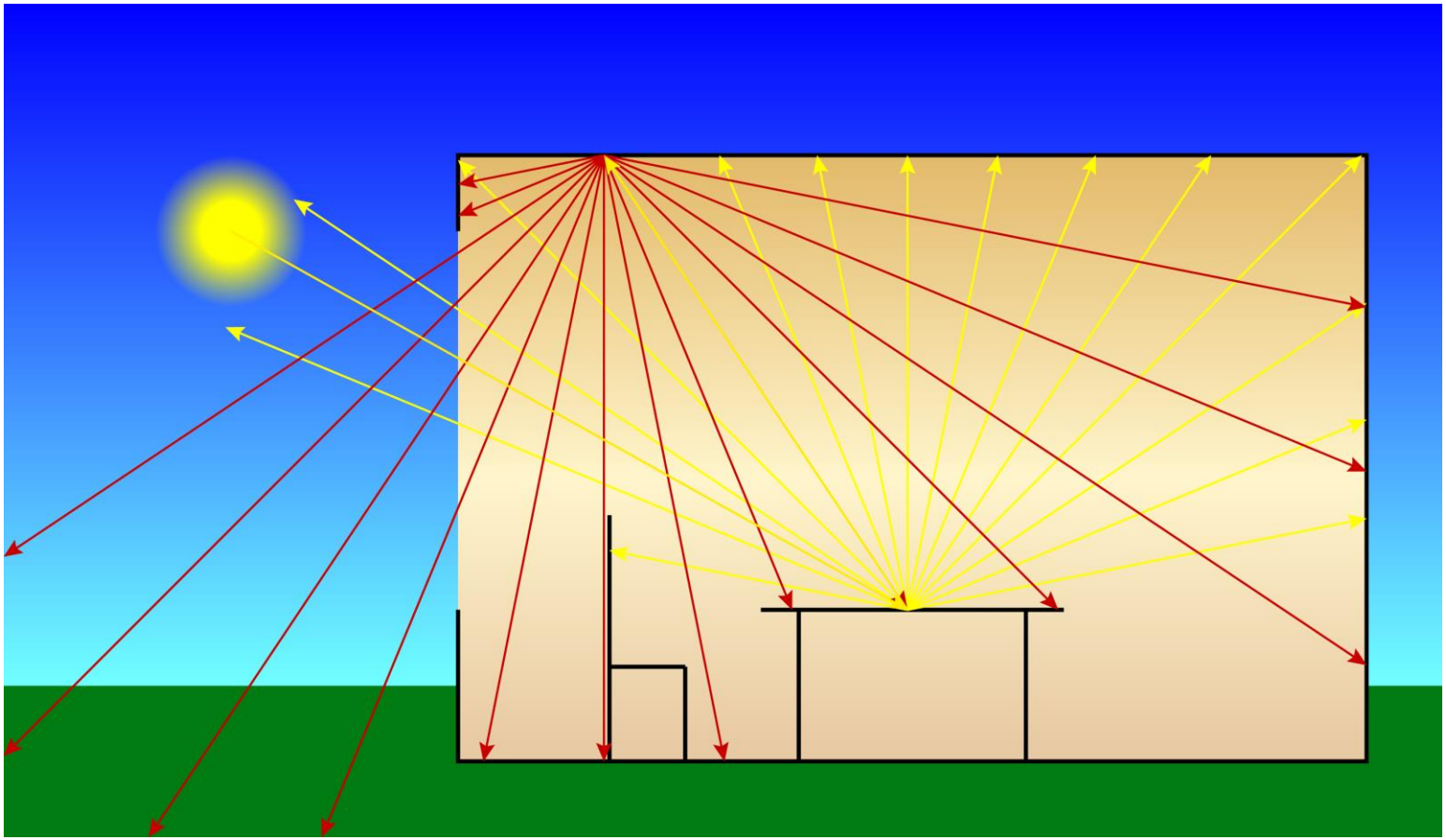
The degree of interreflection …



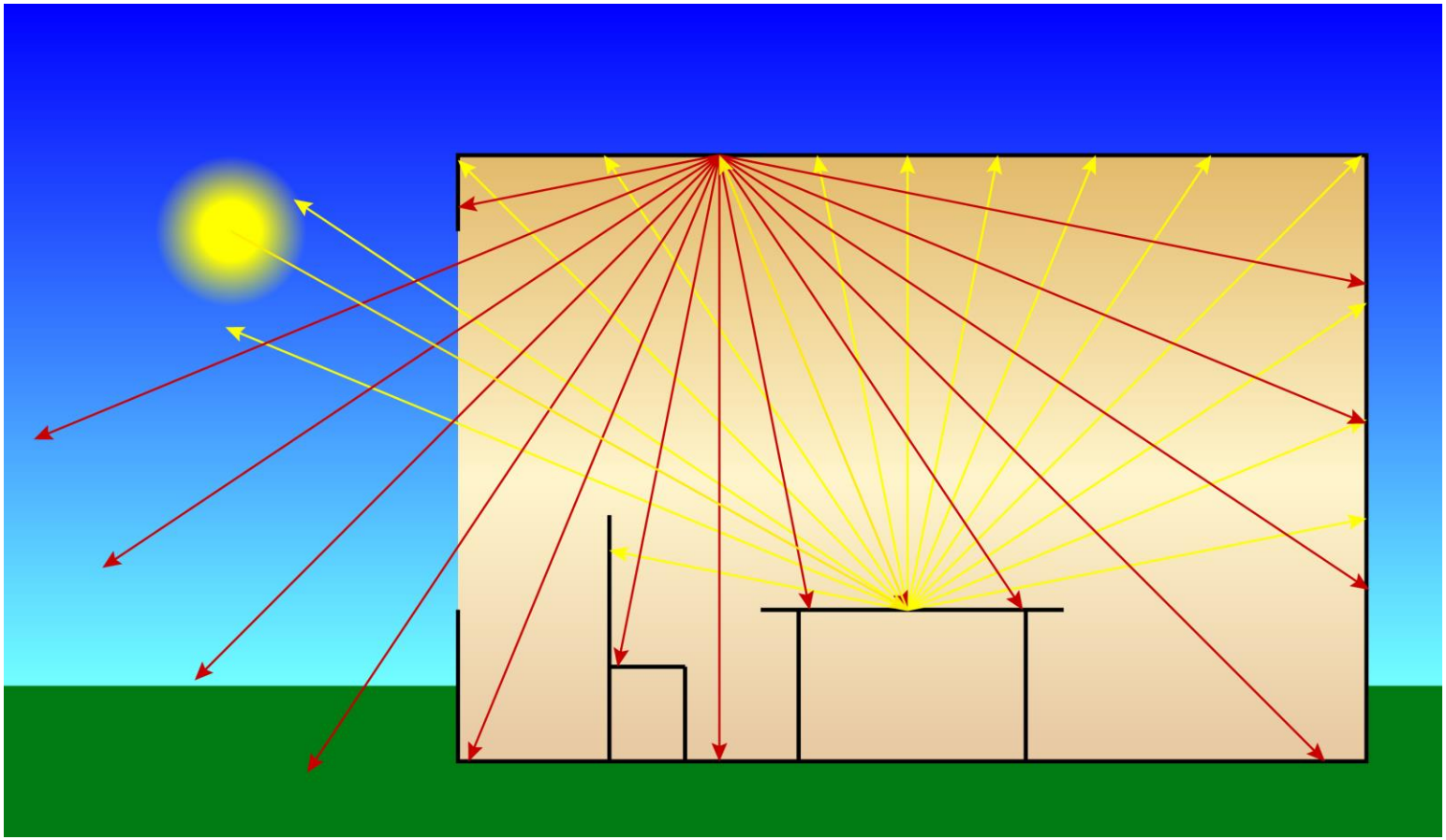
… depends on the material properties …



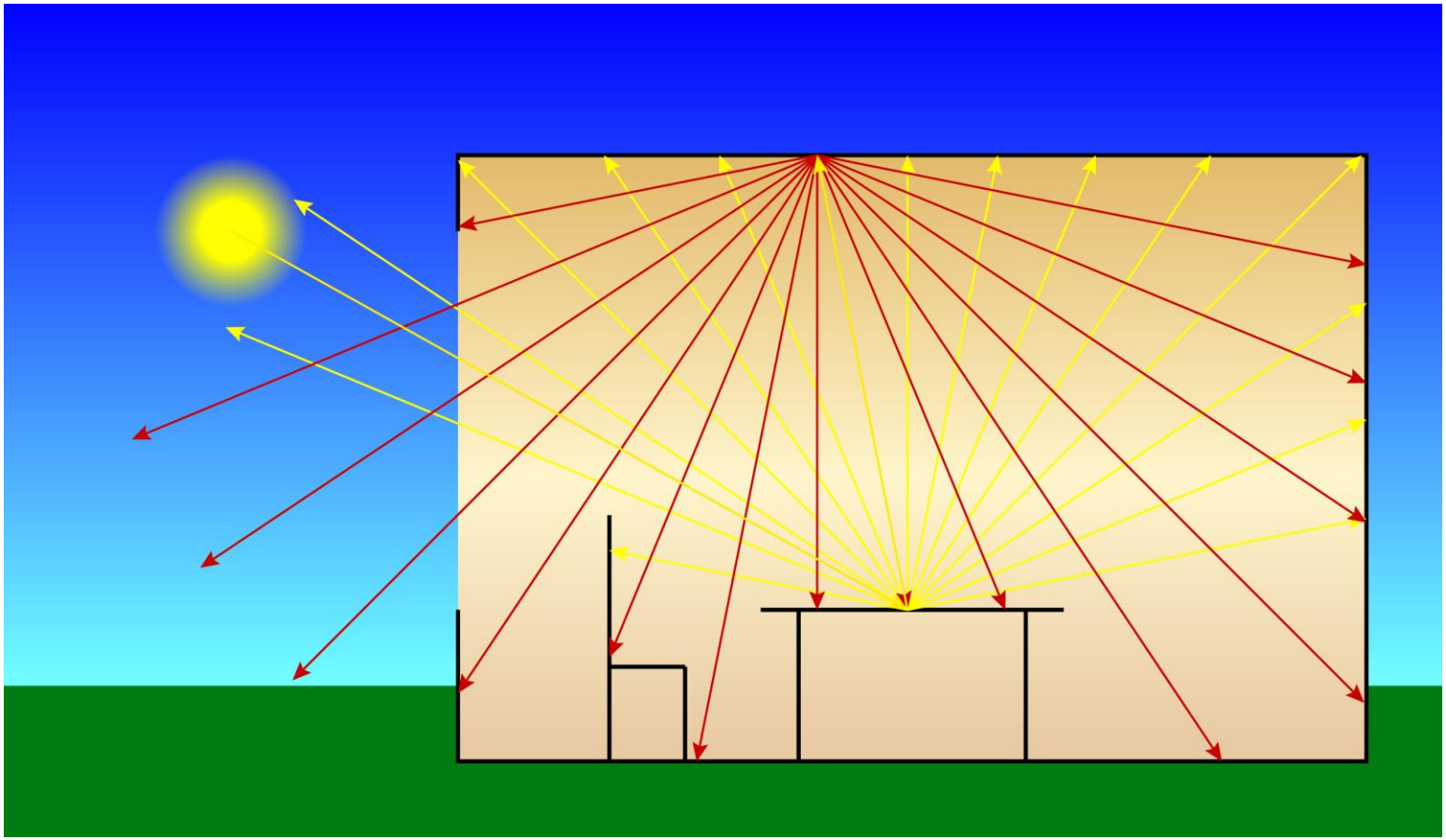
… of the surfaces in the scene.



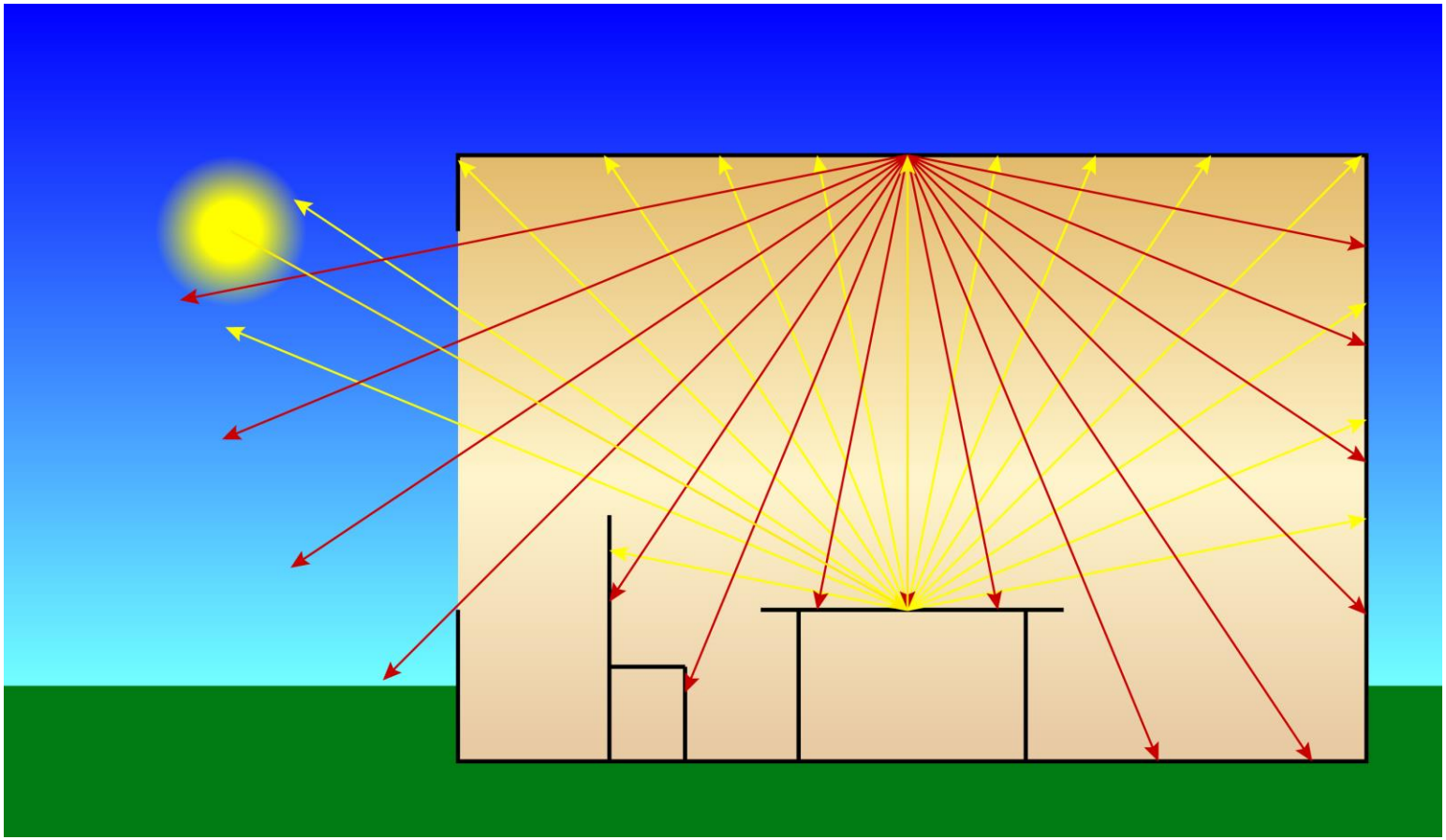
Materials are transparent …



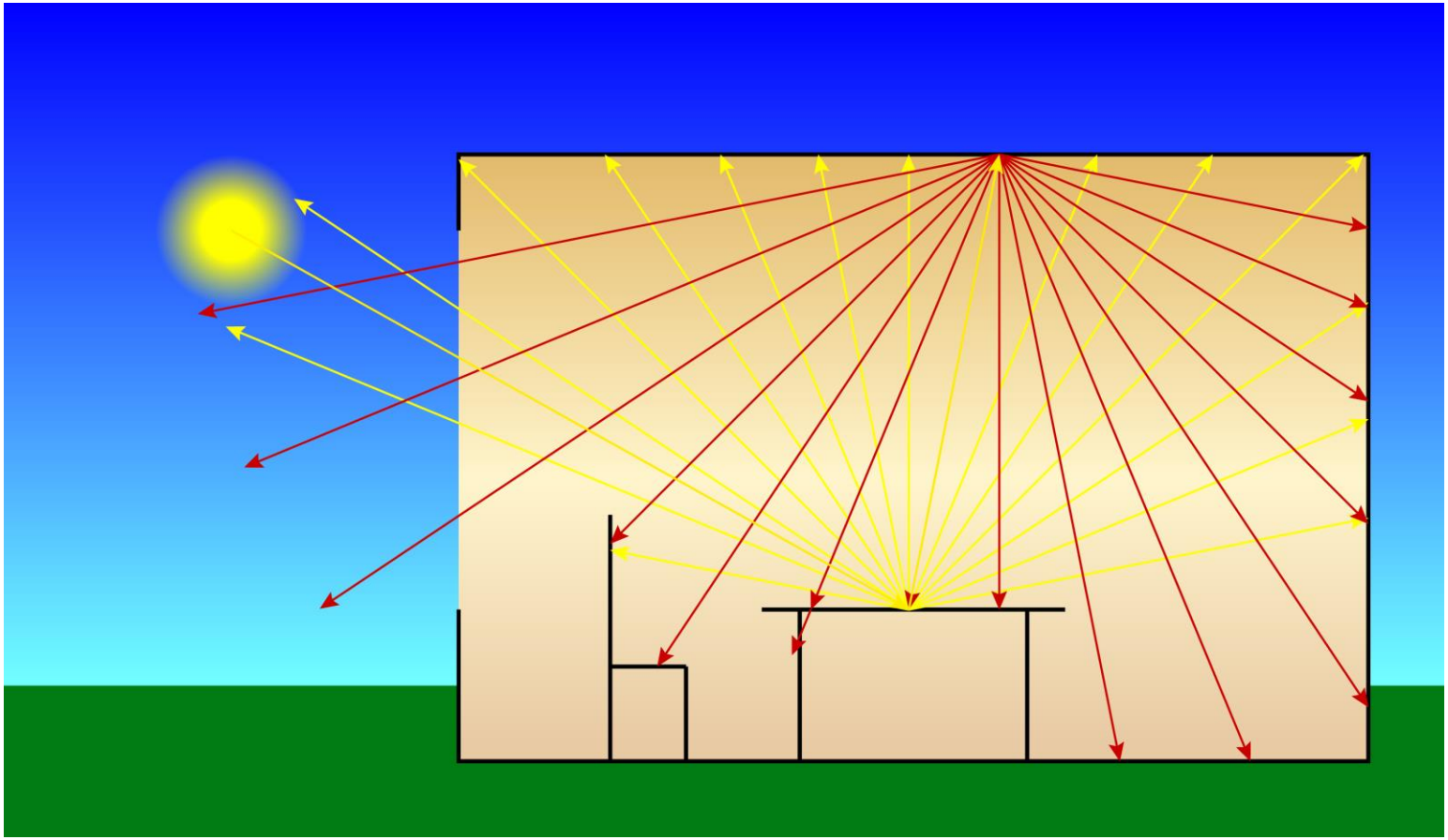
… to some wavelengths …



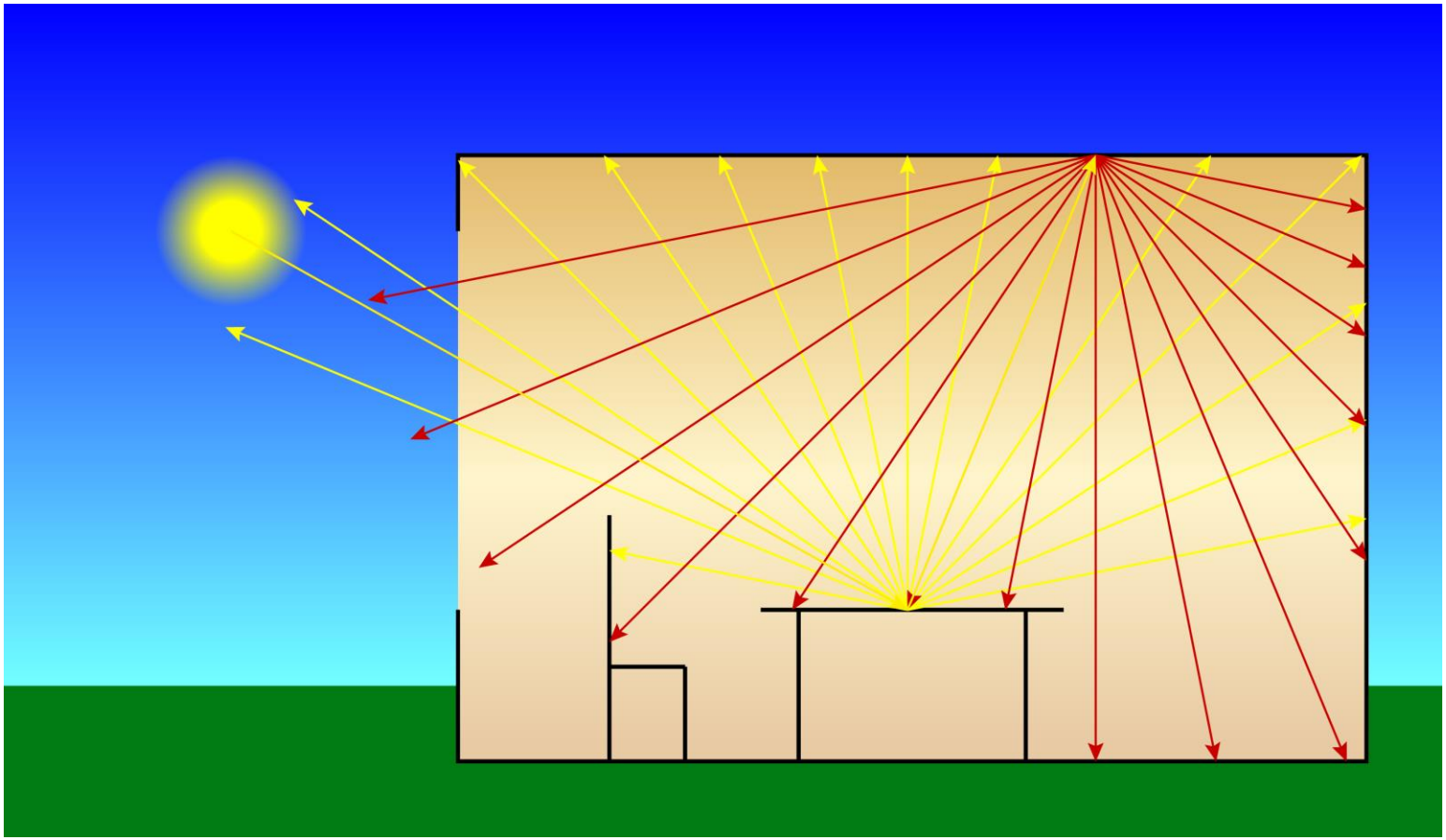
… opaque to others …



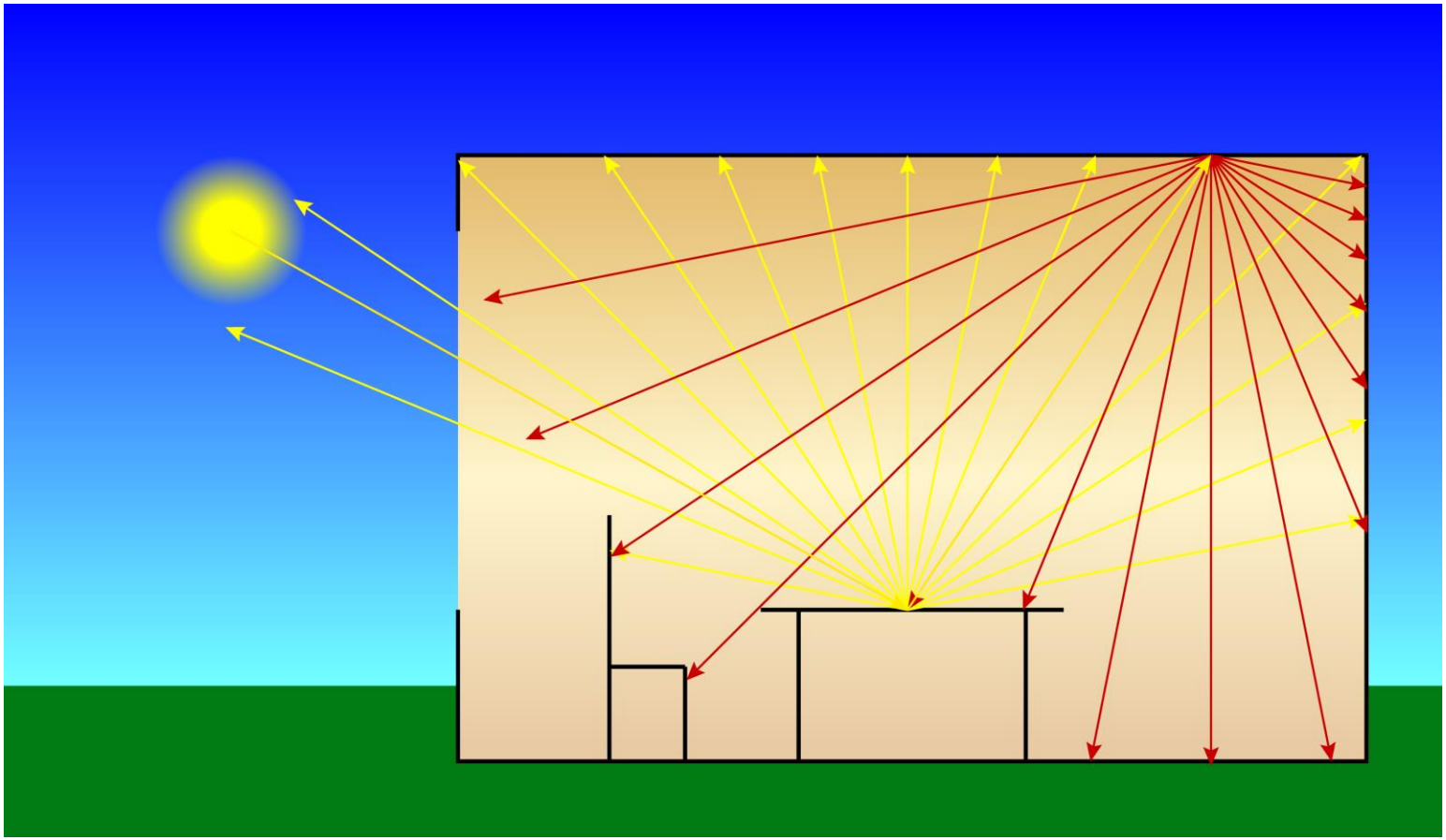
… and reflect yet other wavelengths.



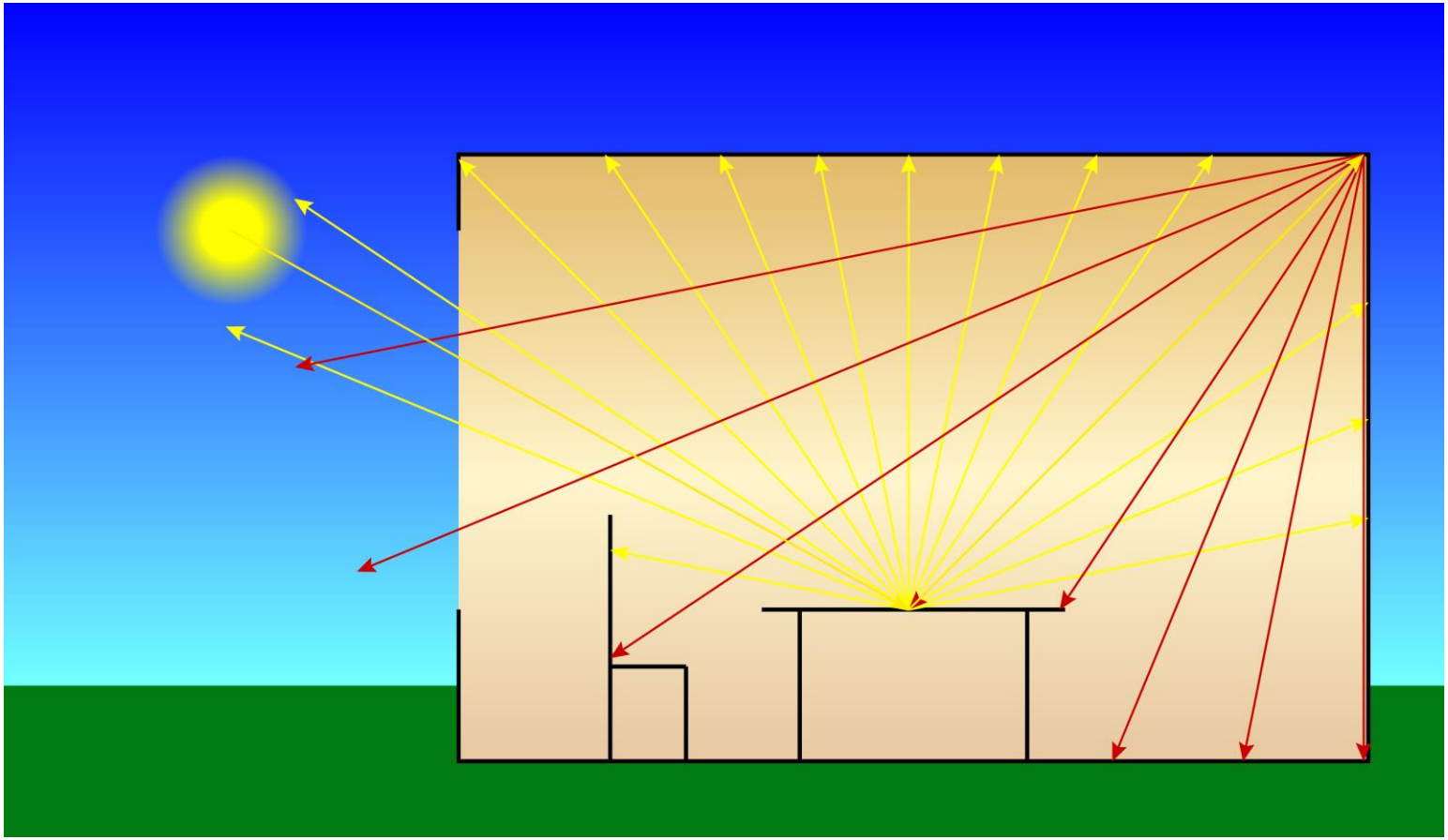
Smooth surfaces may be mirror-like, …



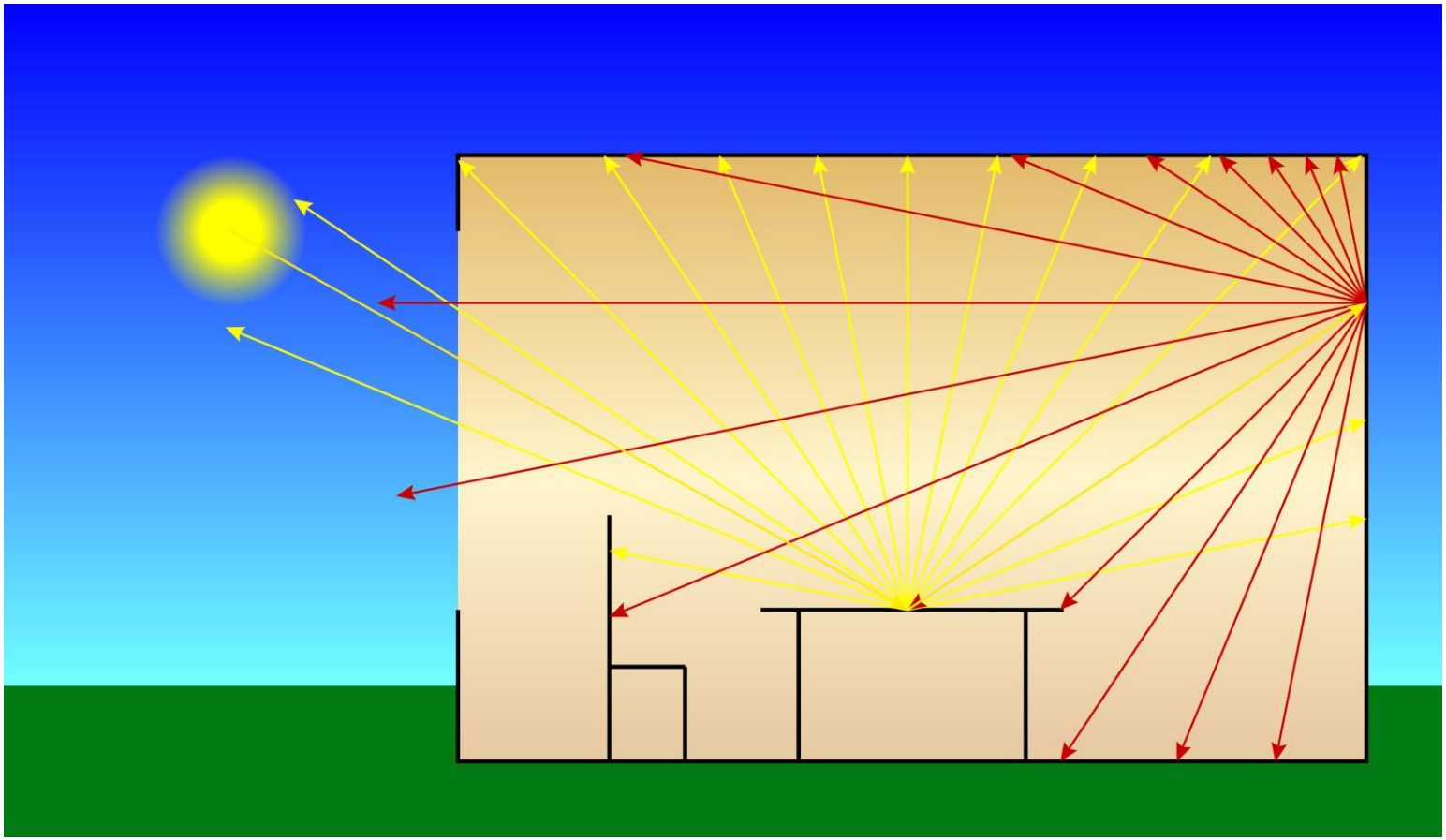
…rough surfaces, matte or diffuse.



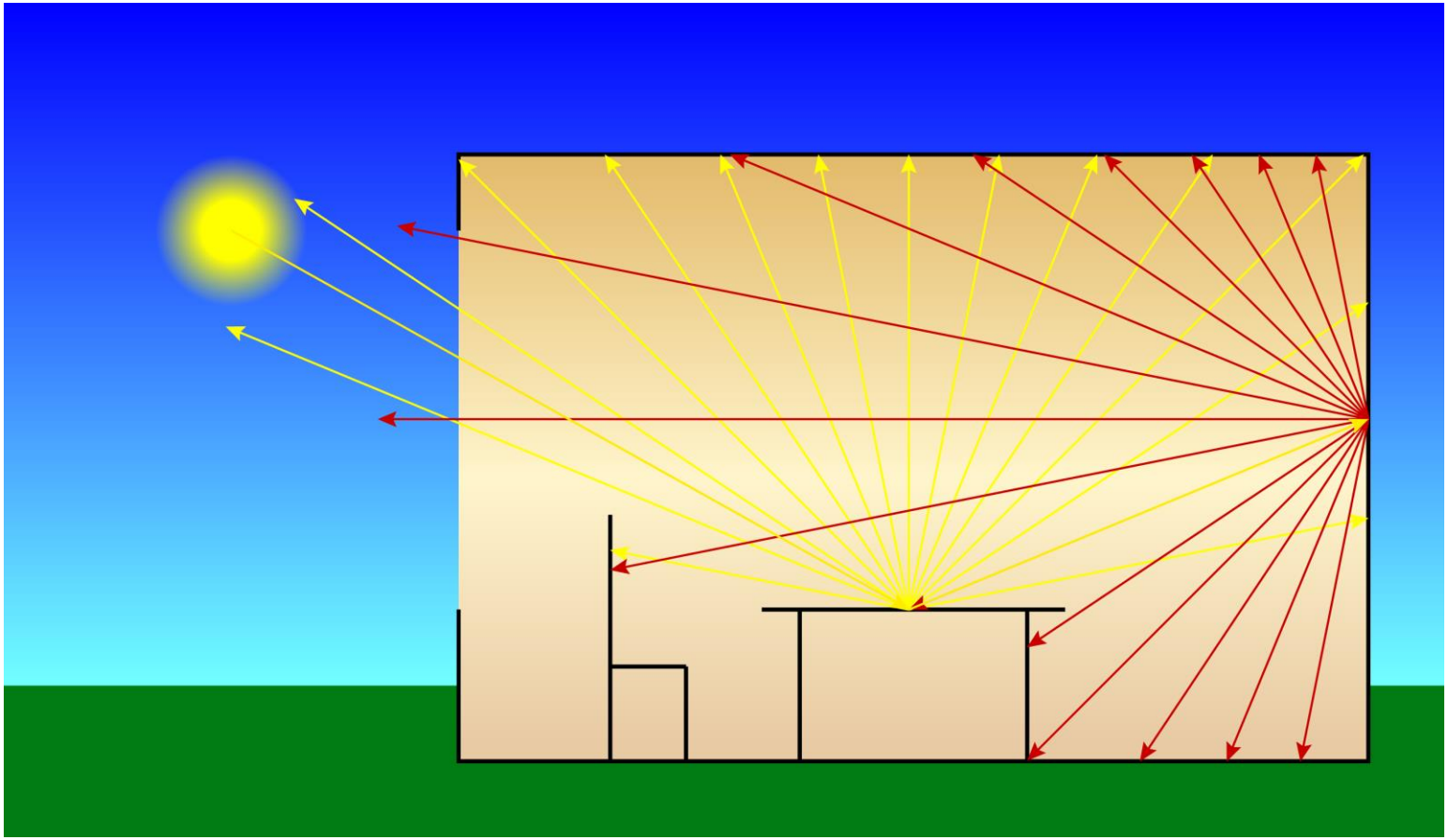
Anisotropic surfaces scatter more in …



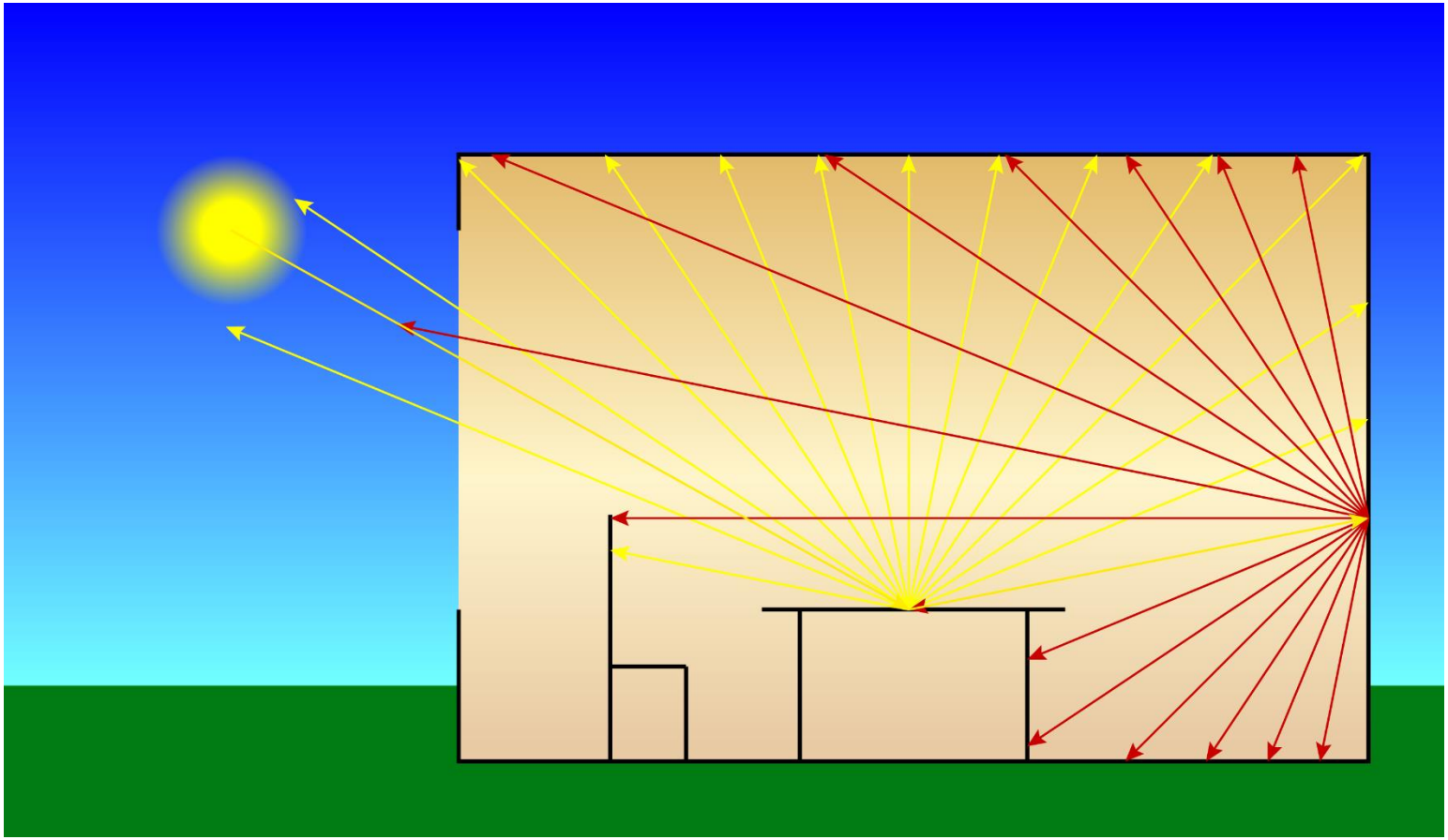
… in one direction than others.



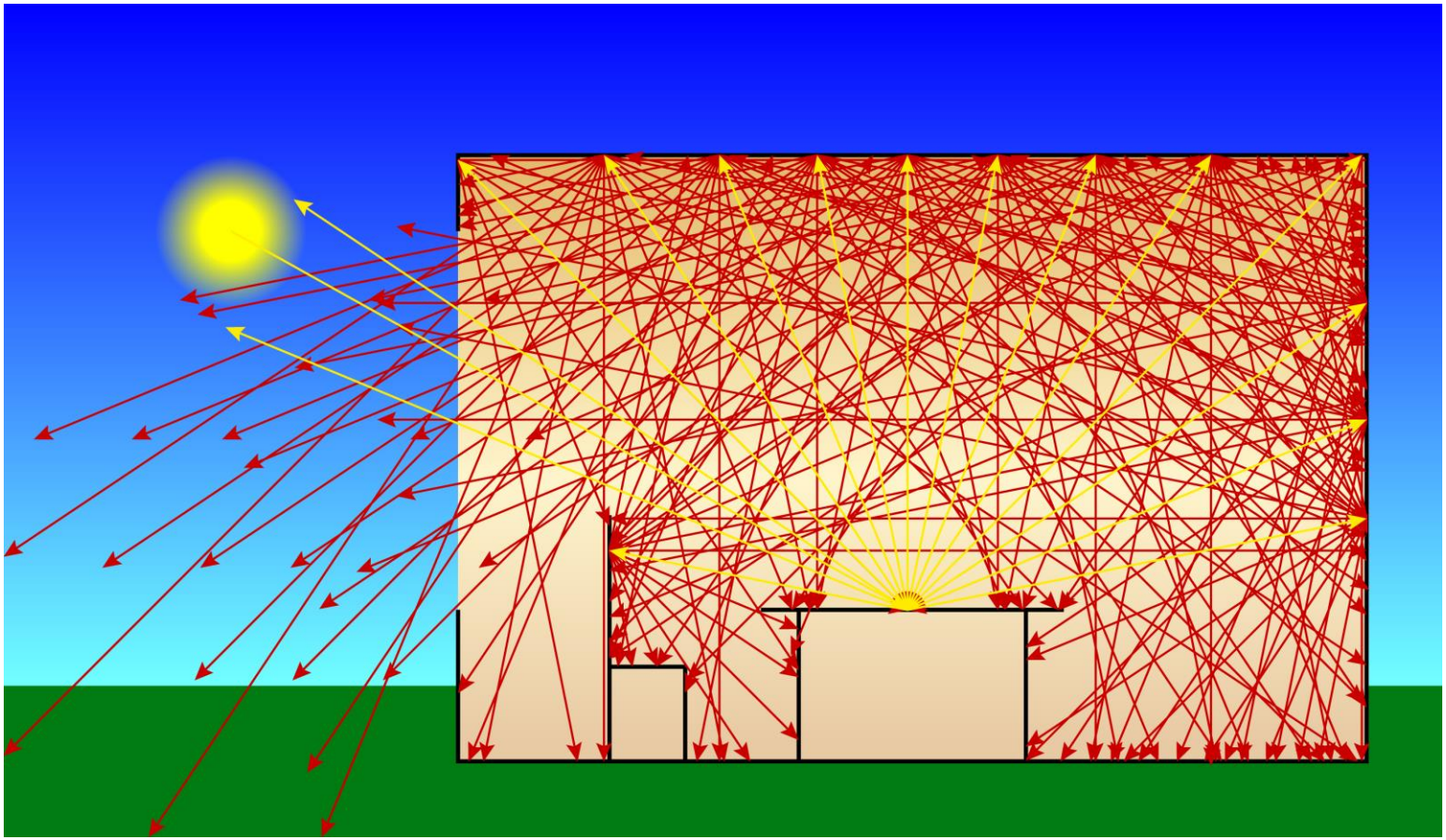
The color of a surface affects those …



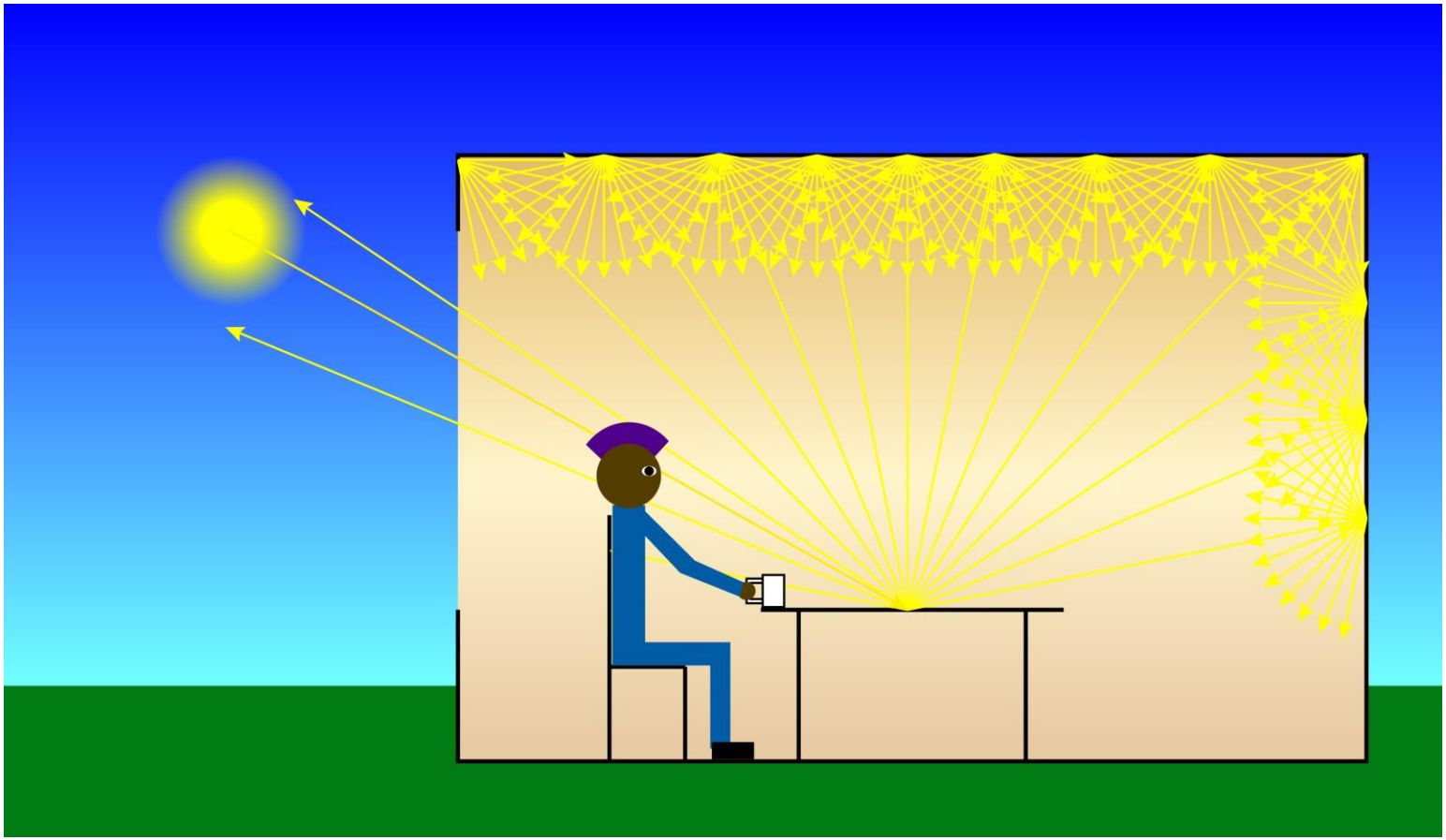
… of nearby surfaces. The result …



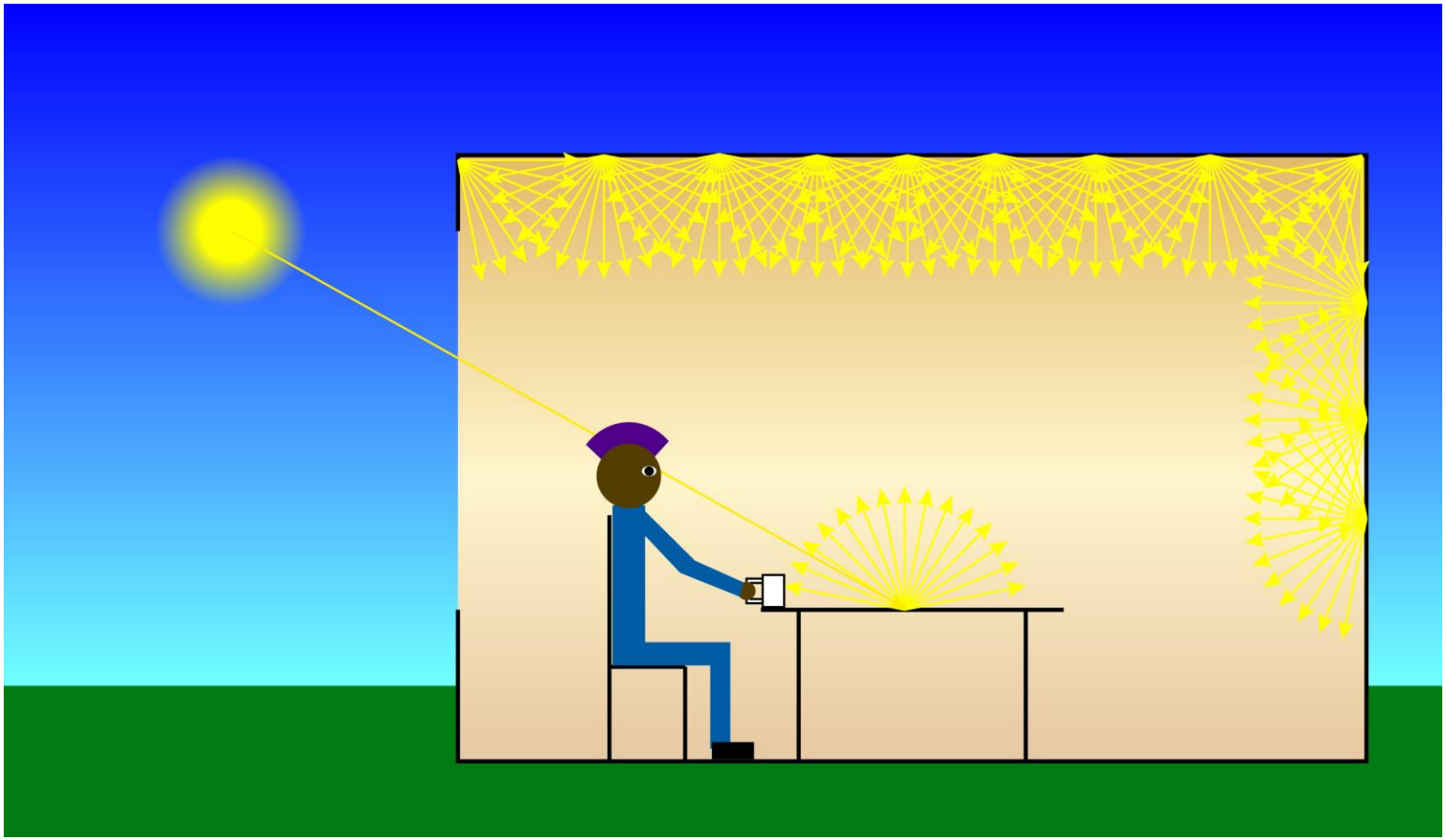
… is highly complex.



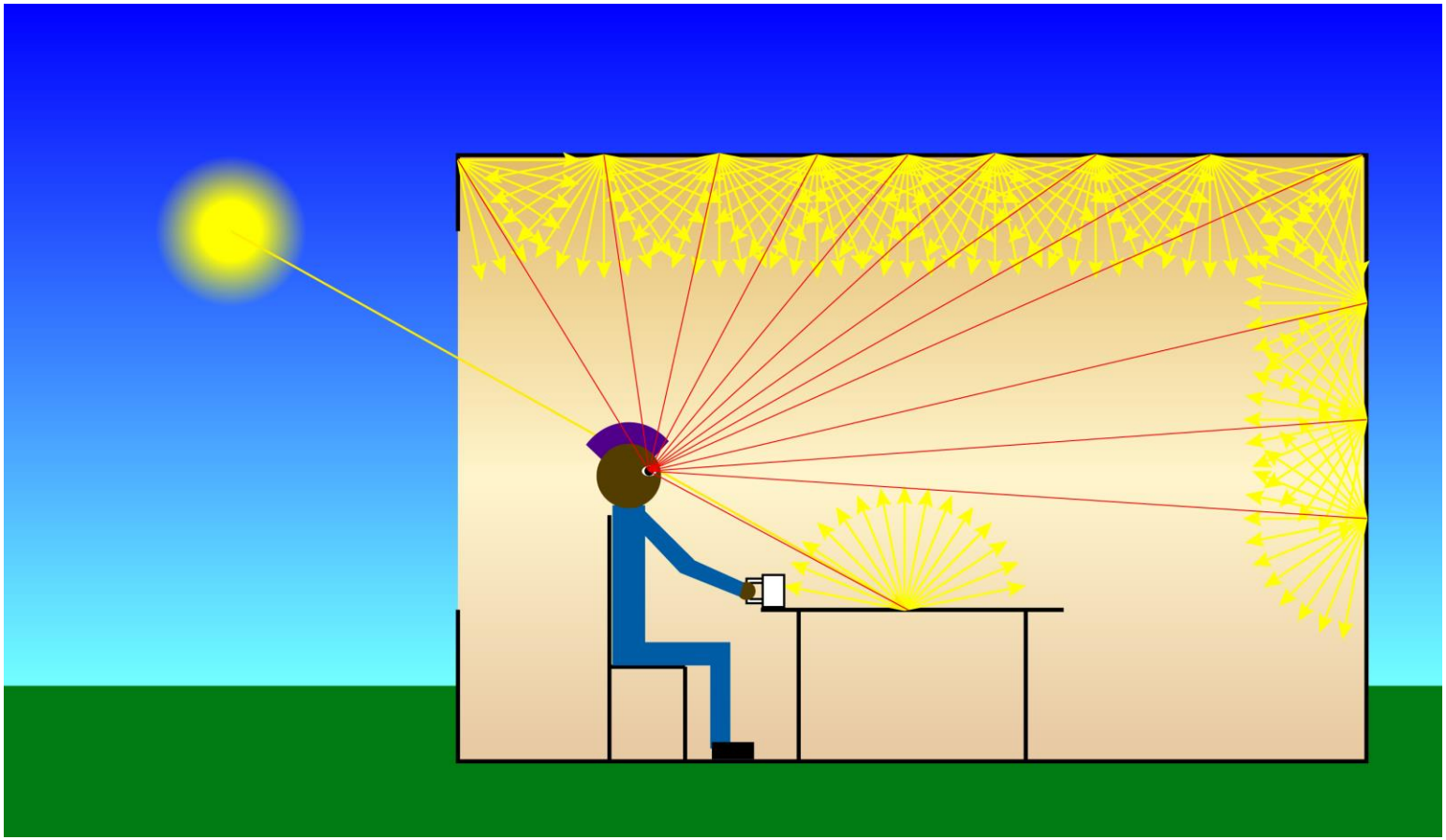
An observer, awash in all the light, …

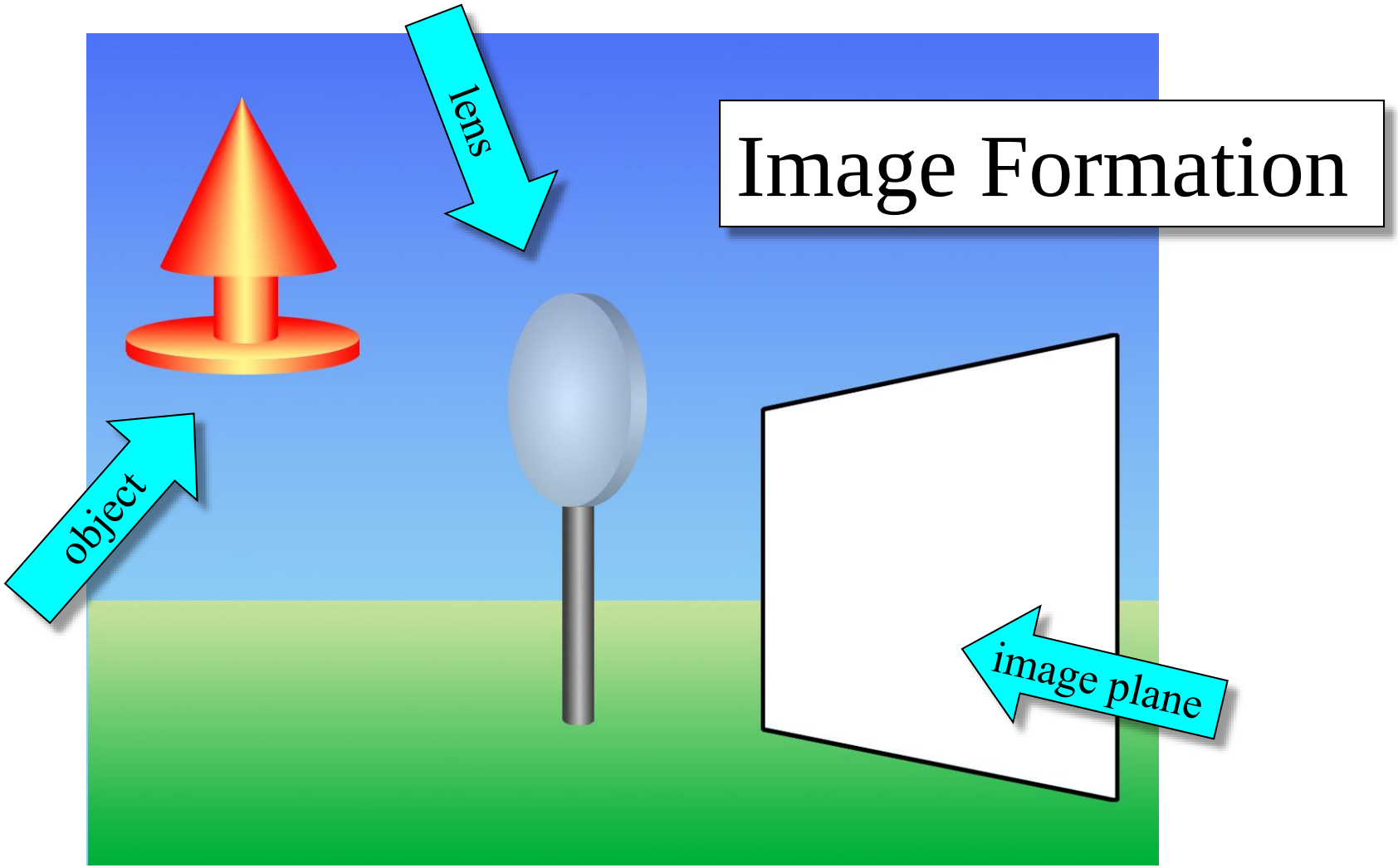


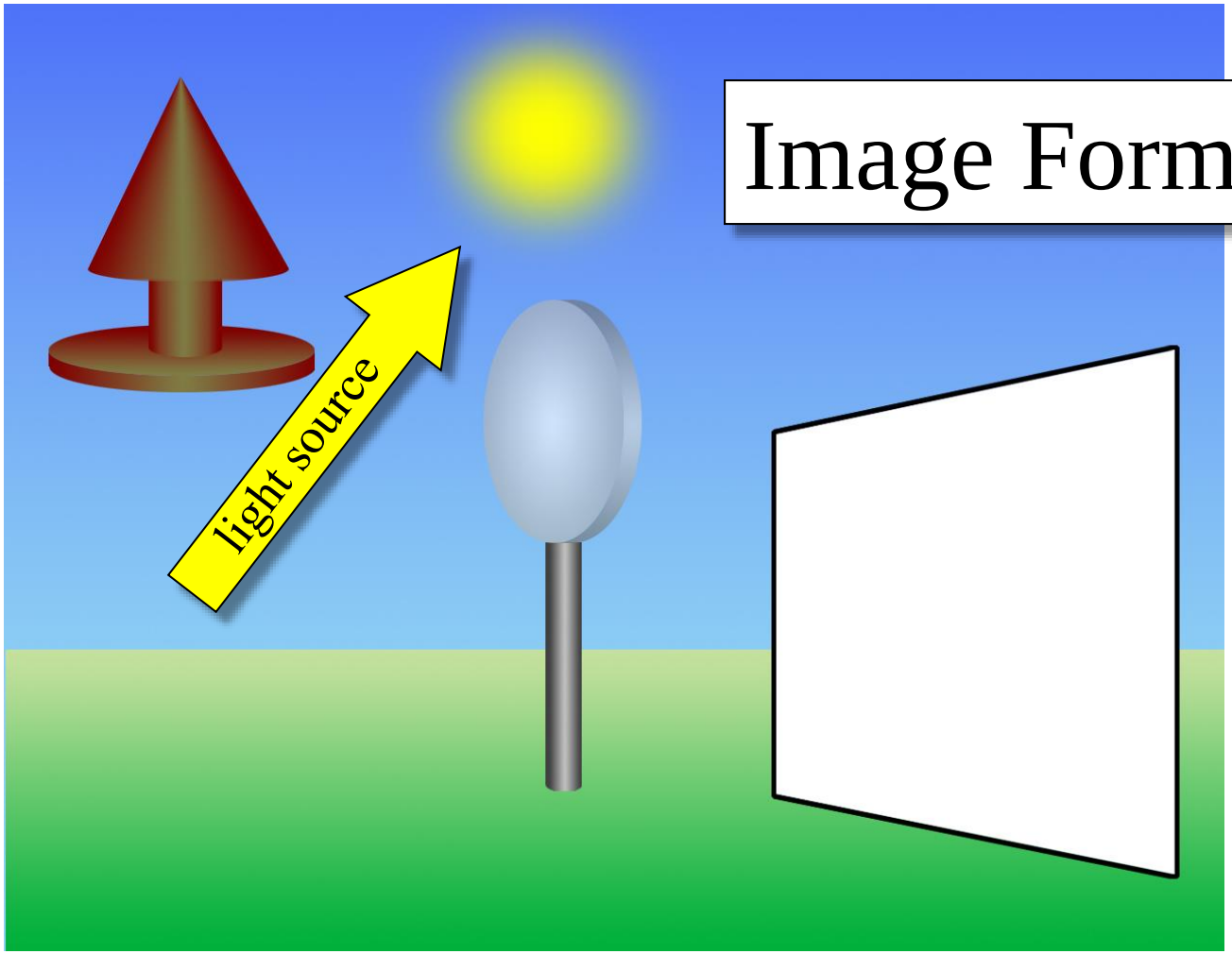
… sees the result of both illumination …

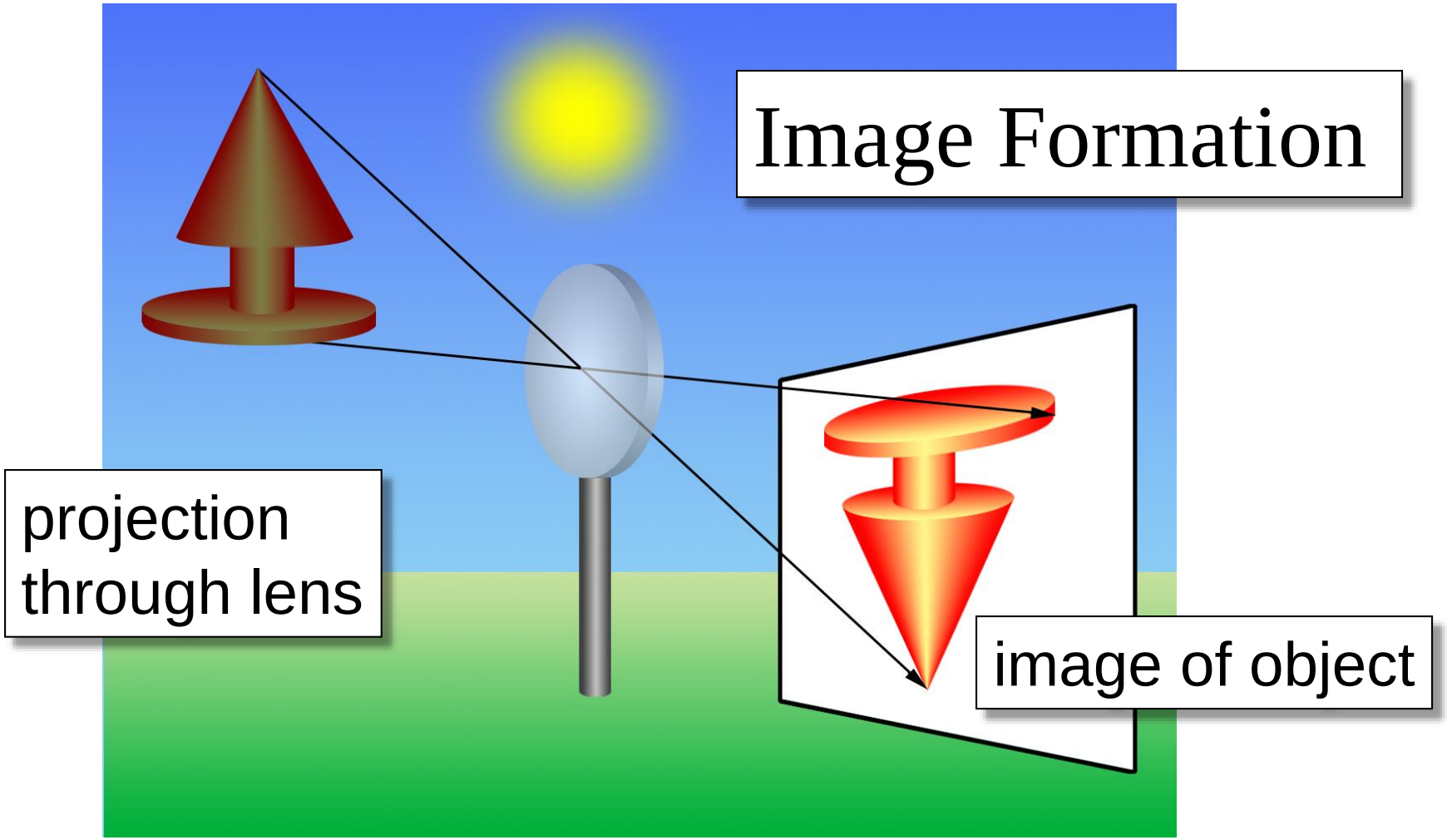


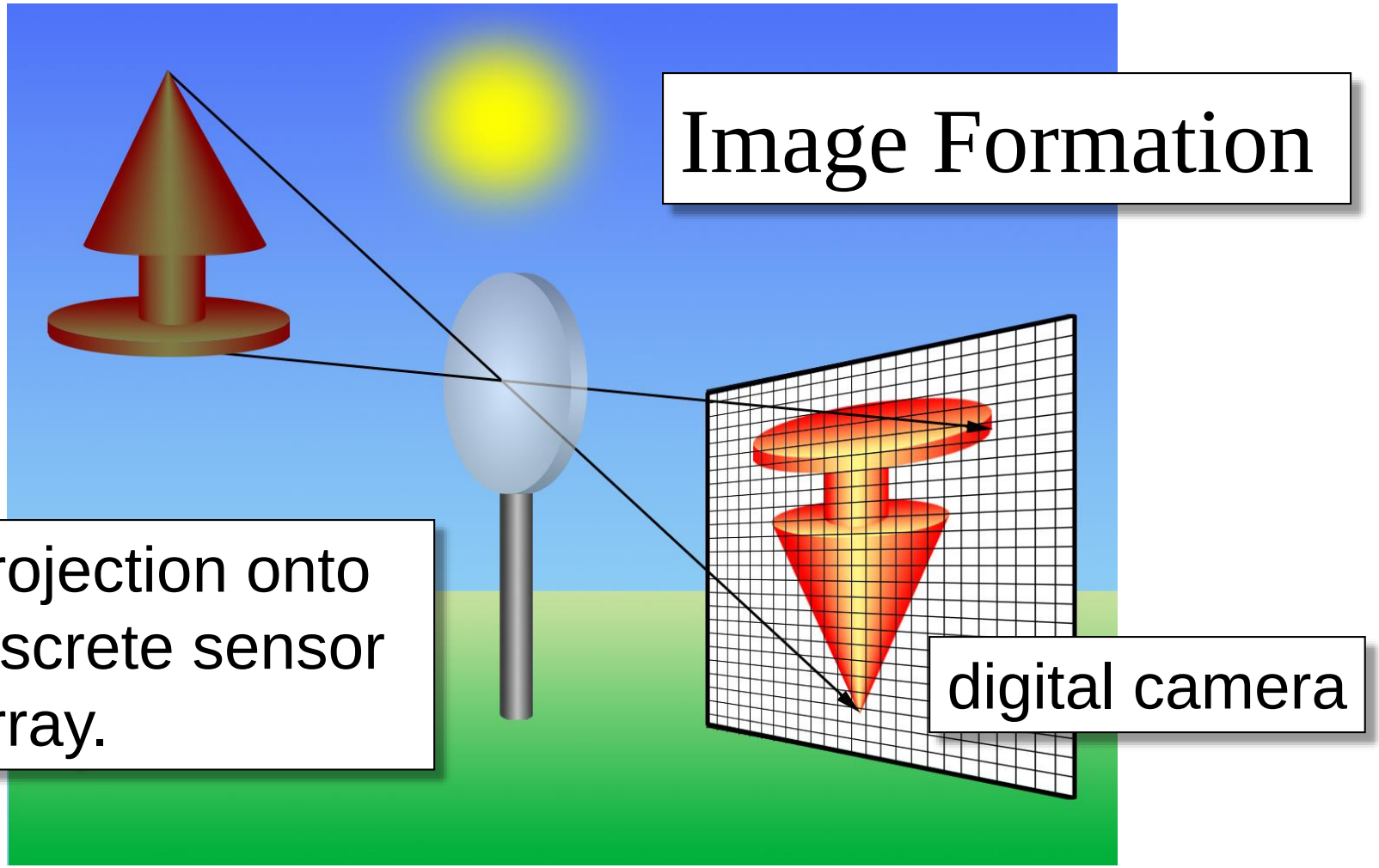
… and surface interreflection.

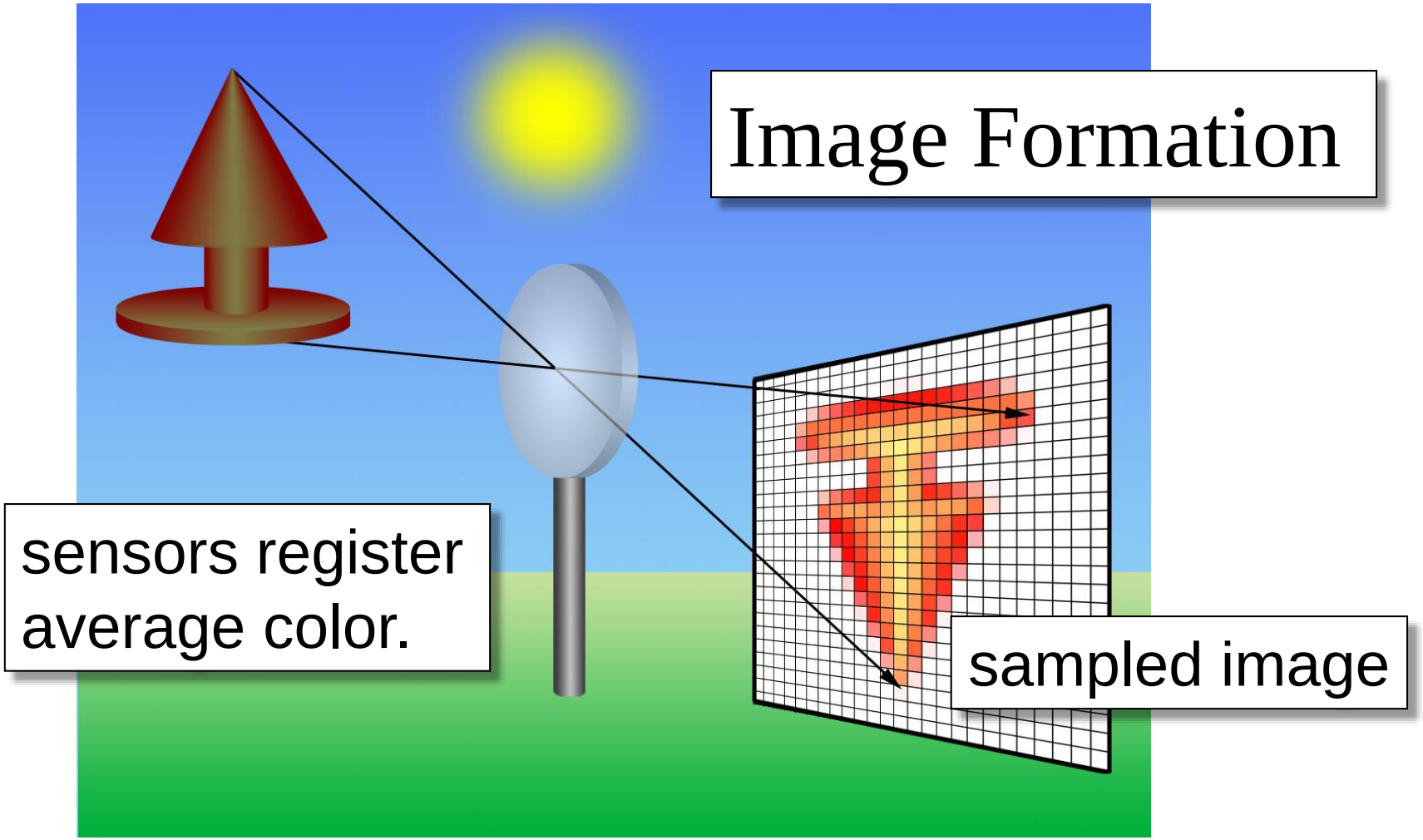


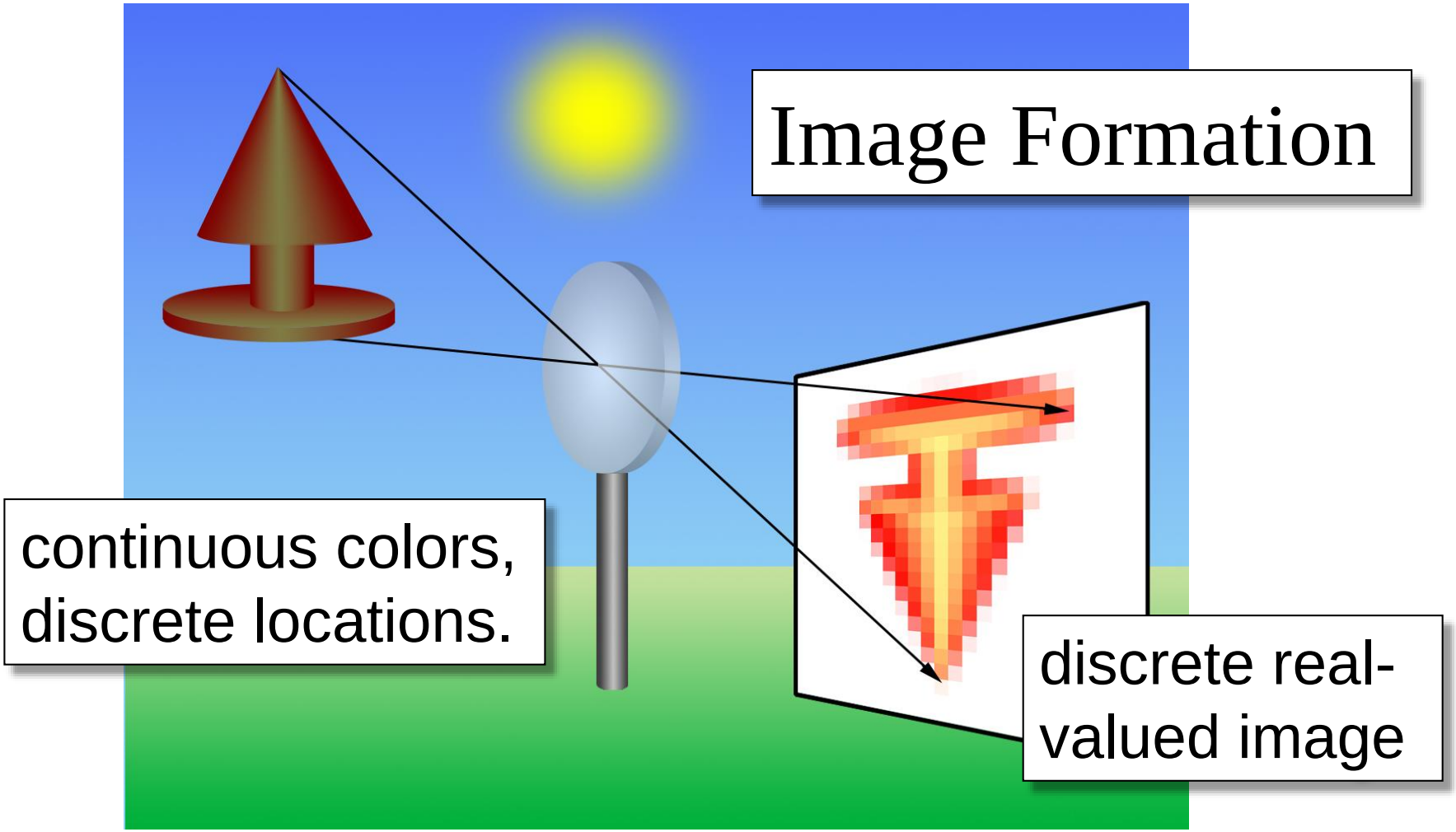




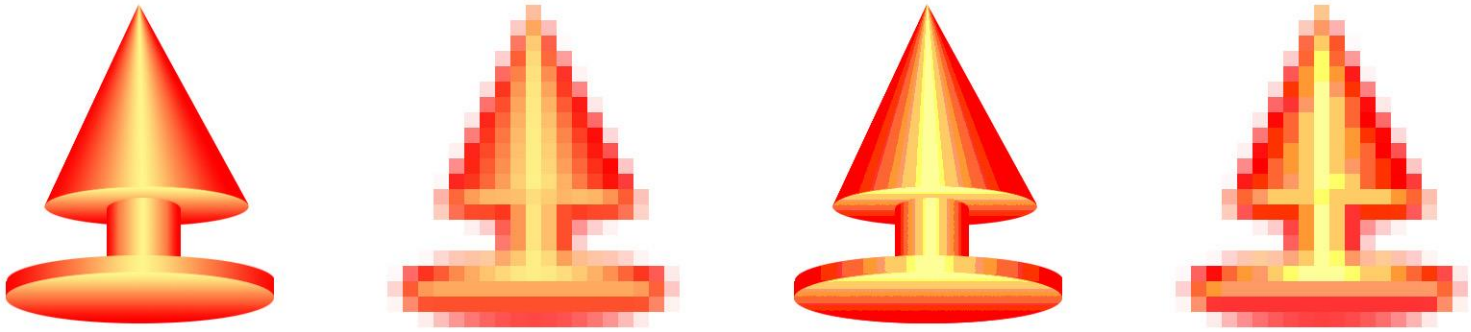






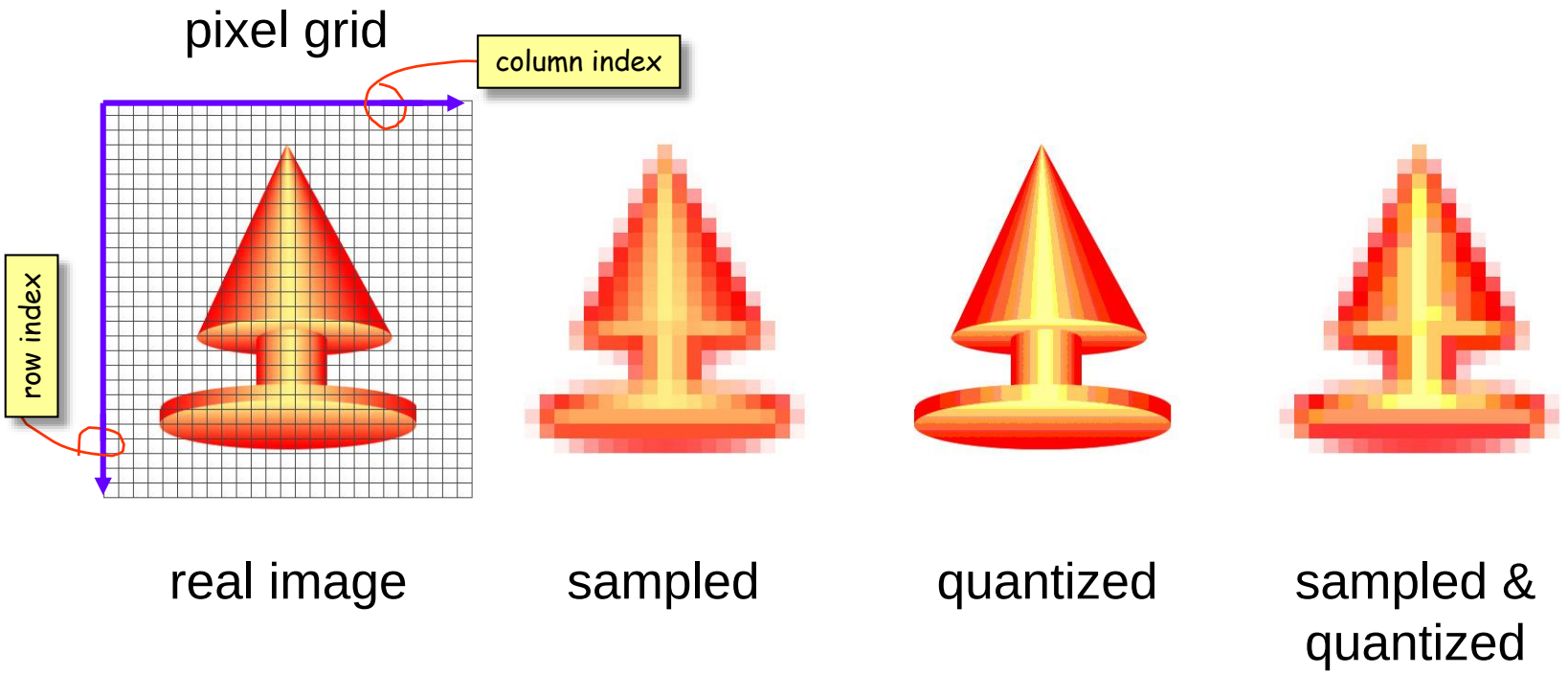


Sampling and Quantization



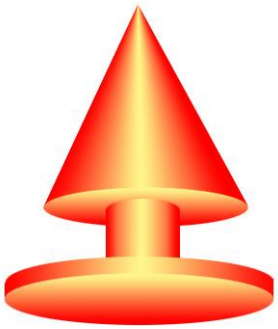
real image | sampled | quantized | sampled & quantized

Sampling and Quantization



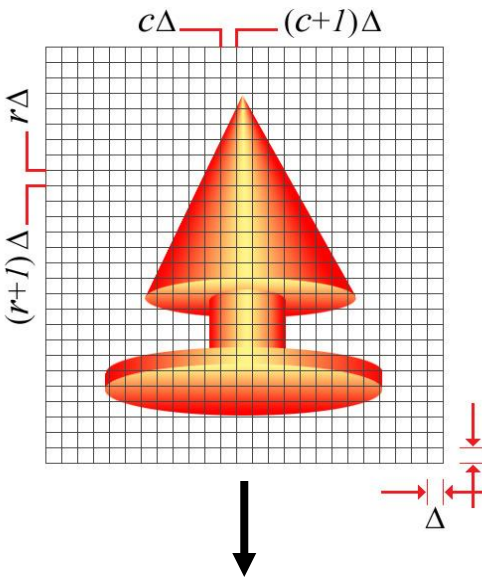
Sampling

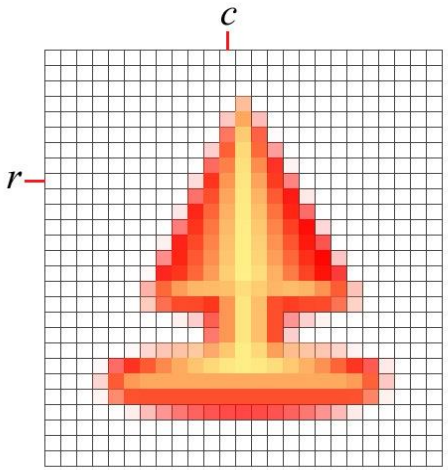
Take the average within each square.



$\mathbf{I}_C(\rho,\chi)$

continuous image





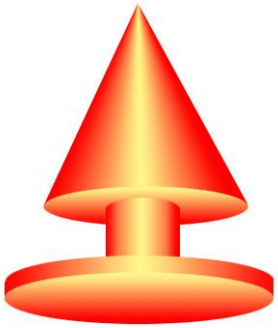
$\mathbf{I}_S(r,c)$

sampled image

$$\mathbf{I}_S(r,c)\frac{1}{\Delta^2}\int_{r\Delta}^{(r+1)\Delta}\int_{c\Delta}^{(c+1)\Delta}\mathbf{I}_C(\rho,\chi)\,\delta\rho\delta\chi$$

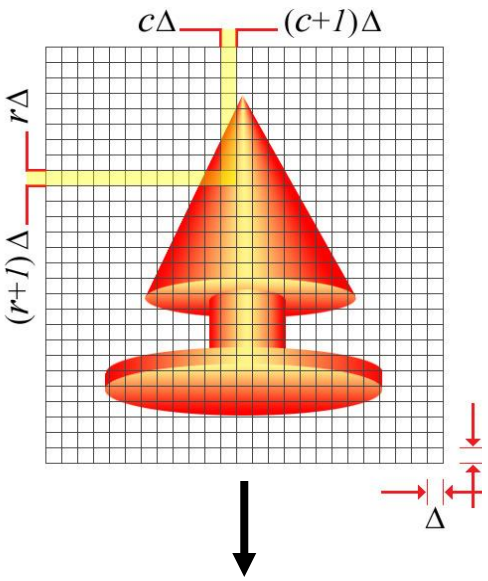
Sampling

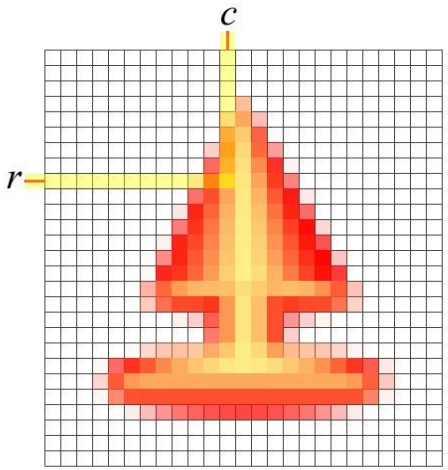
Take the average within each square.

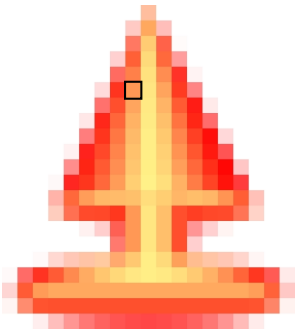


$\mathbf{I}_C(\rho,\chi)$

continuous image







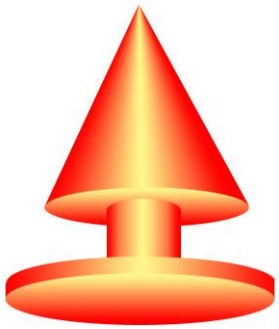
$\mathbf{I}_S(r,c)$

sampled image

$$\mathbf{I}_S(r,c)\frac{1}{\Delta^2}\int_{r\Delta}^{(r+1)\Delta}\int_{c\Delta}^{(c+1)\Delta}\mathbf{I}_C(\rho,\chi)\,\delta\rho\delta\chi$$

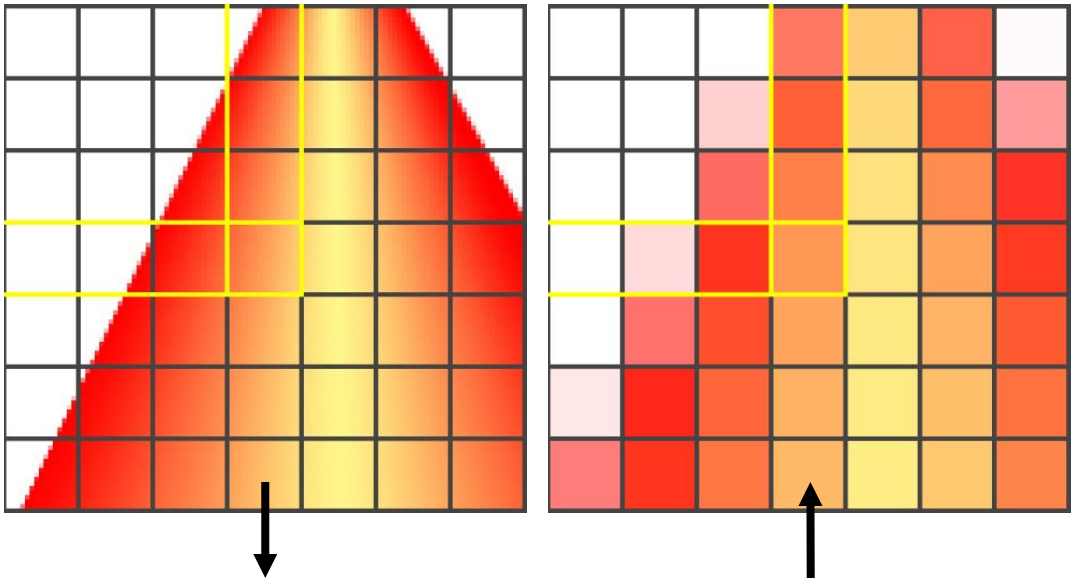
Sampling

Take the average within each square.



$\mathbf{I}_C(\rho,\chi)$

continuous image



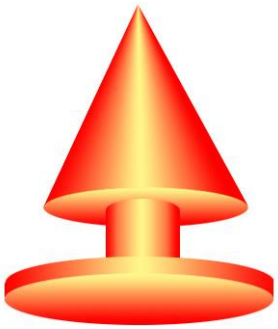
$\mathbf{I}_S(r,c)$

sampled image

$$\mathbf{I}_S(r,c)\frac{1}{\Delta^2}\int_{r\Delta}^{(r+1)\Delta}\int_{c\Delta}^{(c+1)\Delta}\mathbf{I}_C(\rho,\chi)\,\delta\rho\delta\chi$$

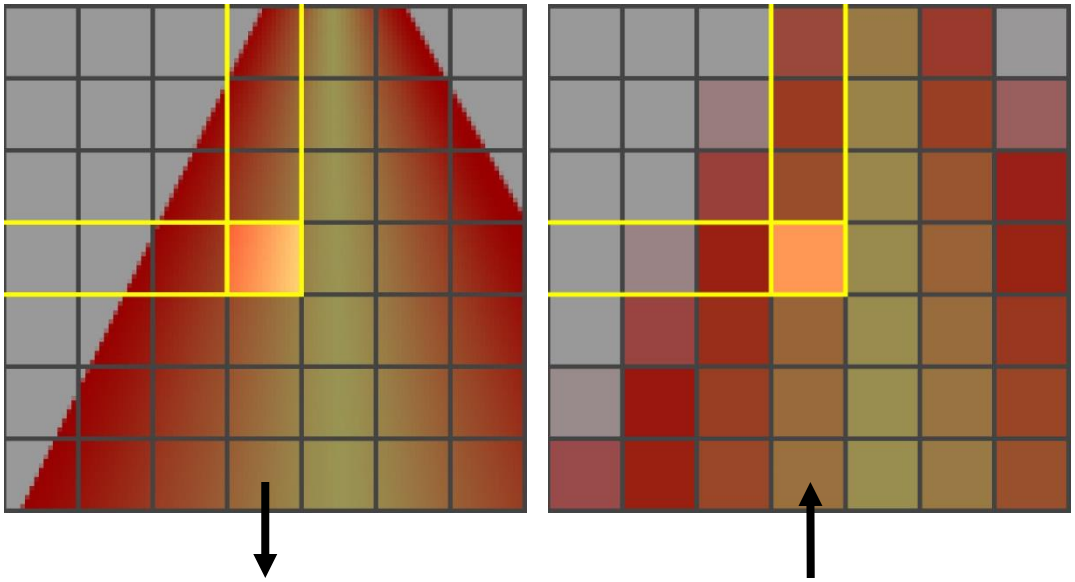
Sampling

Take the average within each square.



$\mathbf{I}_C(\rho,\chi)$

continuous image

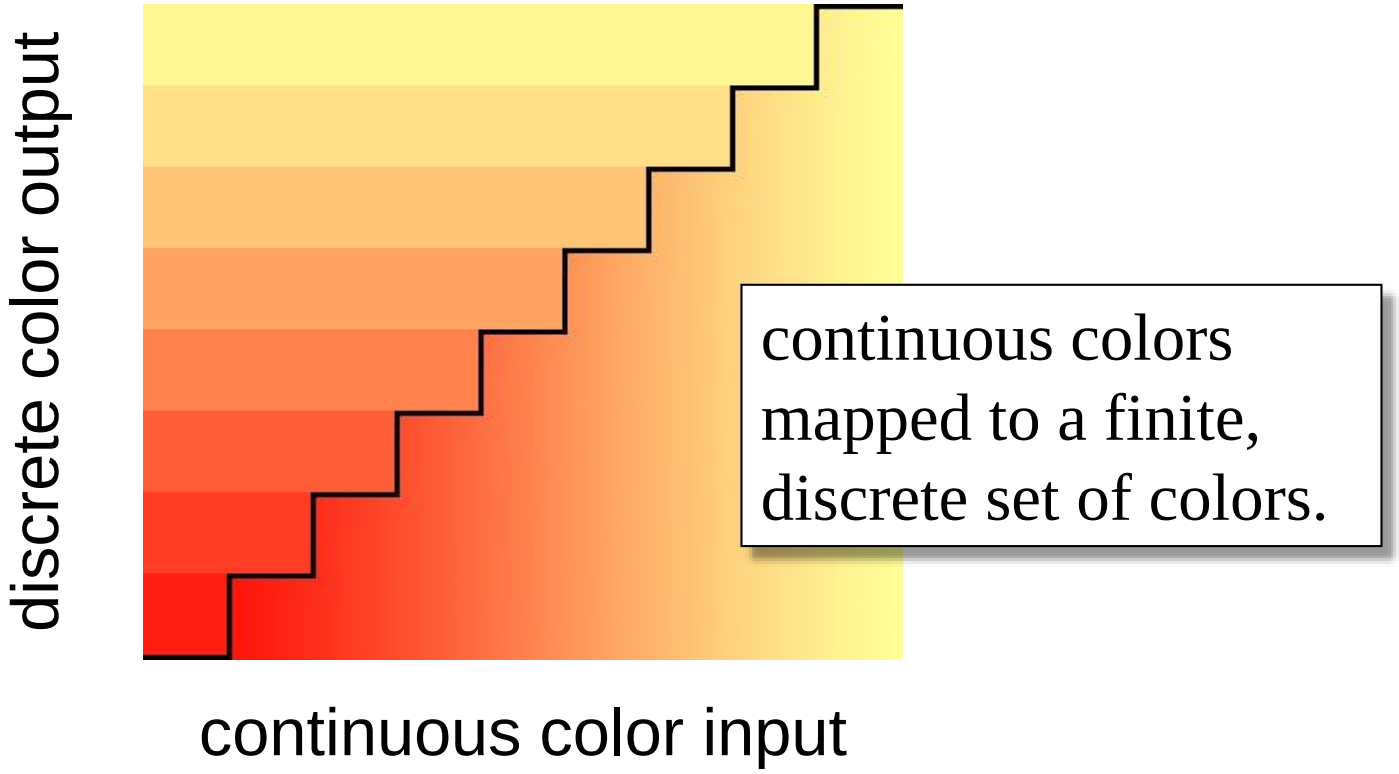


$$\mathbf{I}_S(r,c)\frac{1}{\Delta^2}\int_{r\Delta}^{(r+1)\Delta}\int_{c\Delta}^{(c+1)\Delta}\mathbf{I}_C(\rho,\chi)\,\delta\rho\delta\chi$$

$\mathbf{I}_S(r,c)$

sampled image

Digital Image Formation: Quantization

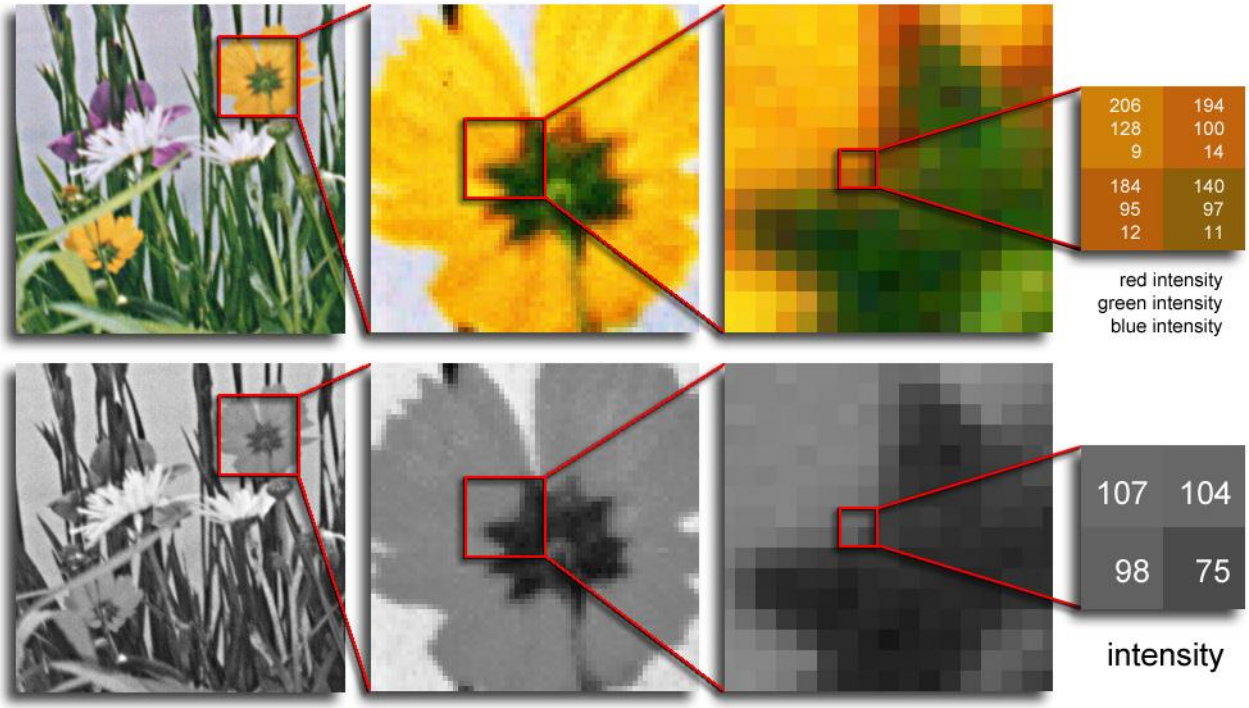


Digital Image

Color images have 3 values per pixel; monochrome images have 1 value per pixel.

a grid of squares, each of which contains a single color

each square is called a pixel (for *picture element*)



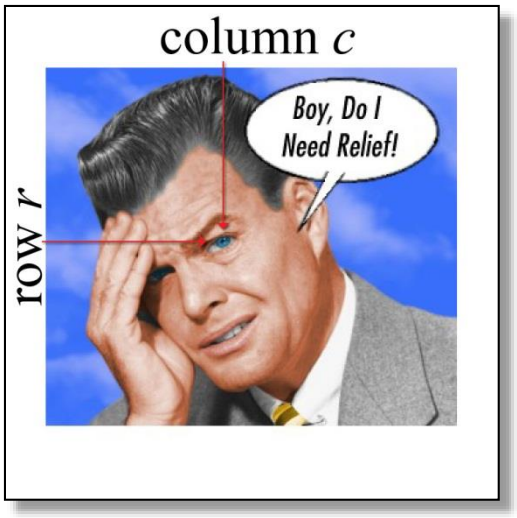
Pixels

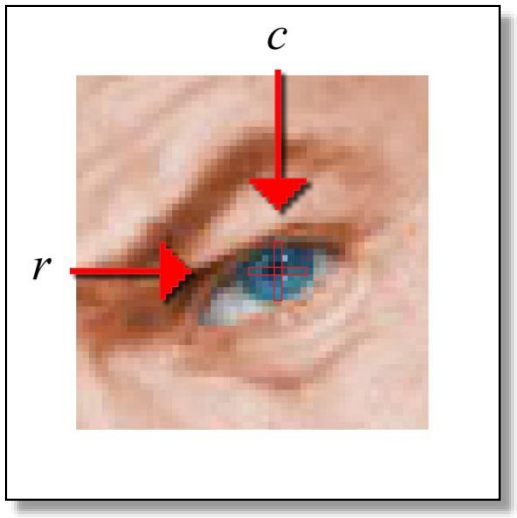
- A digital image, **I**, is a mapping from a 2D grid of uniformly spaced discrete points, $\{\mathbf{p} = (r,c)\}$, into a set of positive integer values, $\{\mathbf{I}(\mathbf{p})\}$, or a set of vector values, *e.g.*, $\{[\mathbf{R}\ \mathbf{G}\ \mathbf{B}]^{\mathsf{T}}(\mathbf{p})\}$.
- At each column location in each row of **I** there is a value.
- The pair (**p**, **I**(**p**)) is called a "pixel" (for *picture element*).

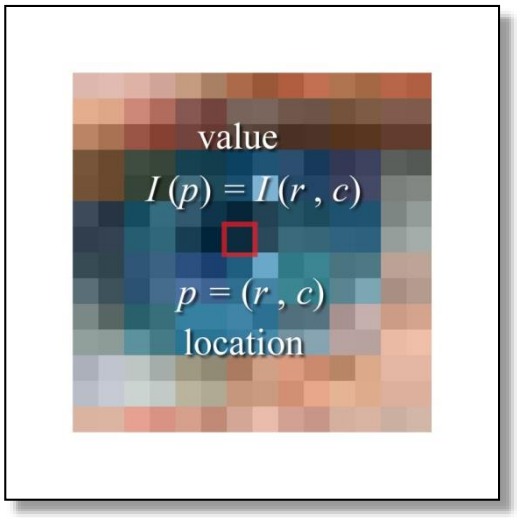
Pixels

- $\mathbf{p} = (r,c)$ is the pixel location indexed by row, r, and column, c.
- $\mathbf{I}(\mathbf{p}) = \mathbf{I}(r,c)$ is the value of the pixel at location $\mathbf{p}$.
- If $\mathbf{I}(\mathbf{p})$ is a single number then $\mathbf{I}$ is monochrome.
- If $\mathbf{I}(\mathbf{p})$ is a vector (ordered list of numbers) then $\mathbf{I}$ has multiple bands (*e.g.*, a color image).

Pixels







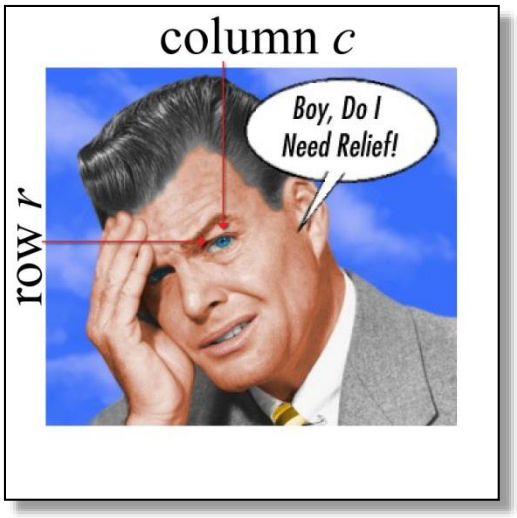
Pixel Location: $\mathbf{p} = (r, c)$

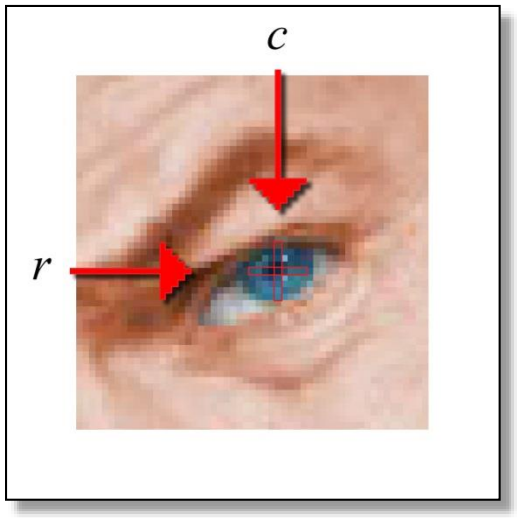
Pixel Value: $\mathbf{I(p)} = I(r, c)$

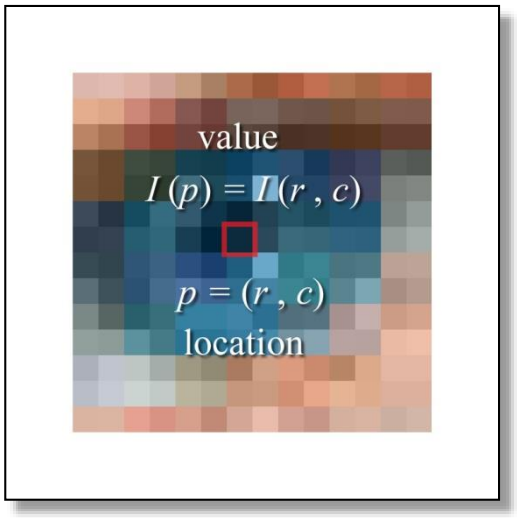
Pixel : $[\mathbf{p}, \mathbf{I(p)}]$

Pixels

Pixel : [**p**, **I(p)**]





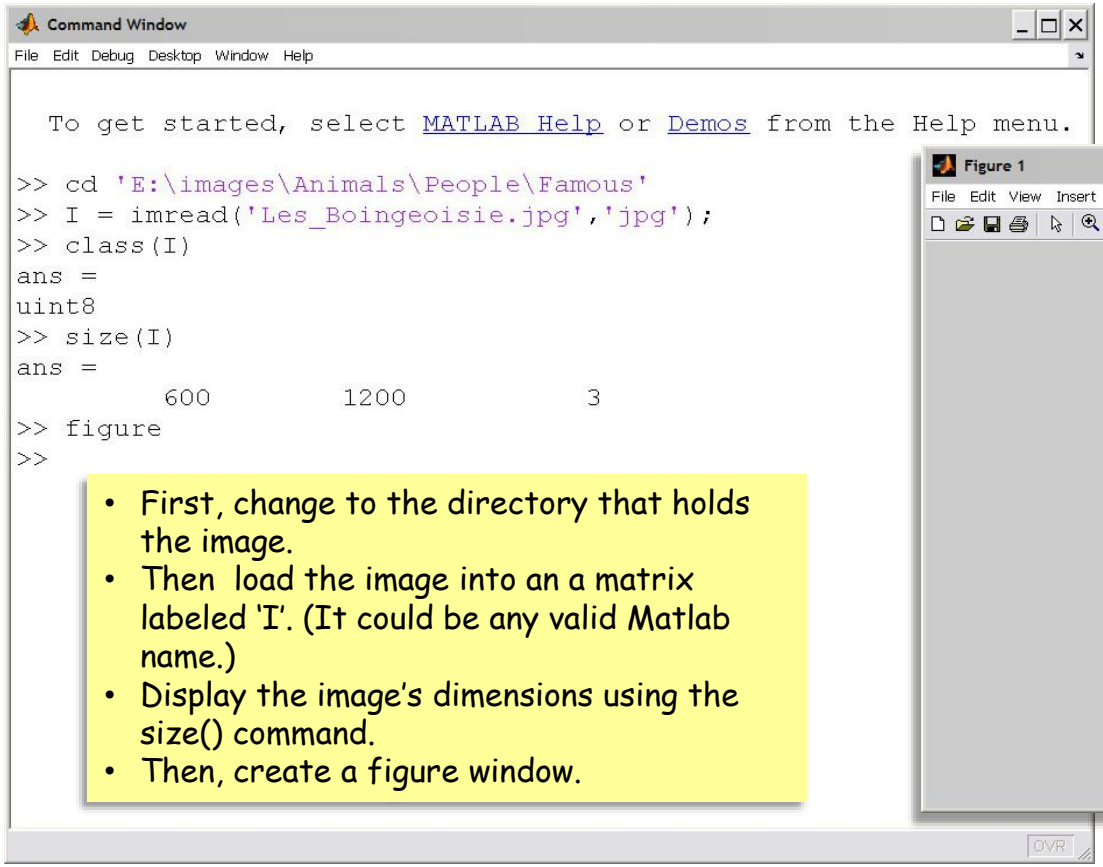


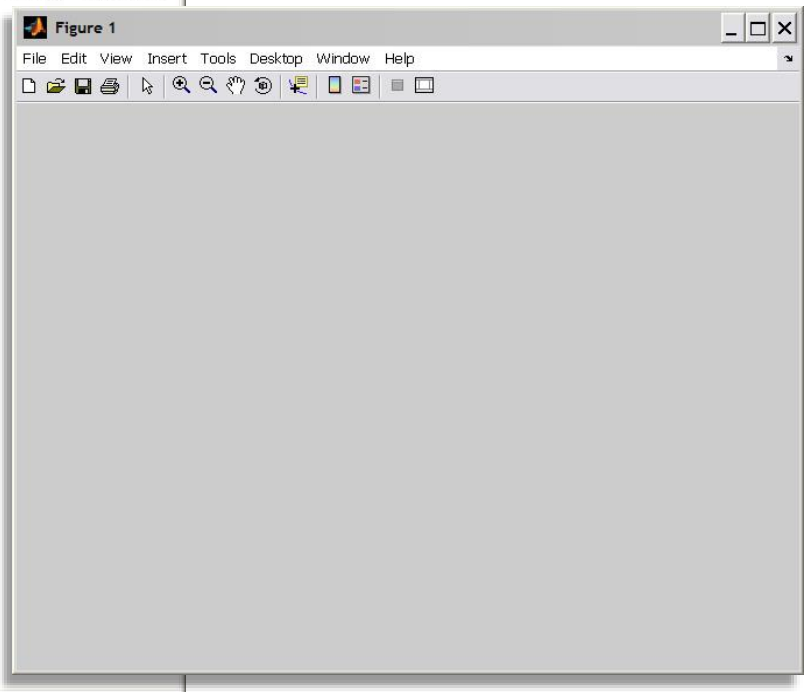
$$\mathbf{p} = (r, c) = (\text{row\#},\ \text{col\#}) = (272,\ 277)$$

$$\mathbf{I(p)} = \begin{bmatrix} \text{red} \\ \text{green} \\ \text{blue} \end{bmatrix} = \begin{bmatrix} 12 \\ 43 \\ 61 \end{bmatrix}$$

The next few slides demonstrate the results of the commands shown in the command window. Each successive slide starts at the end of the previous one.

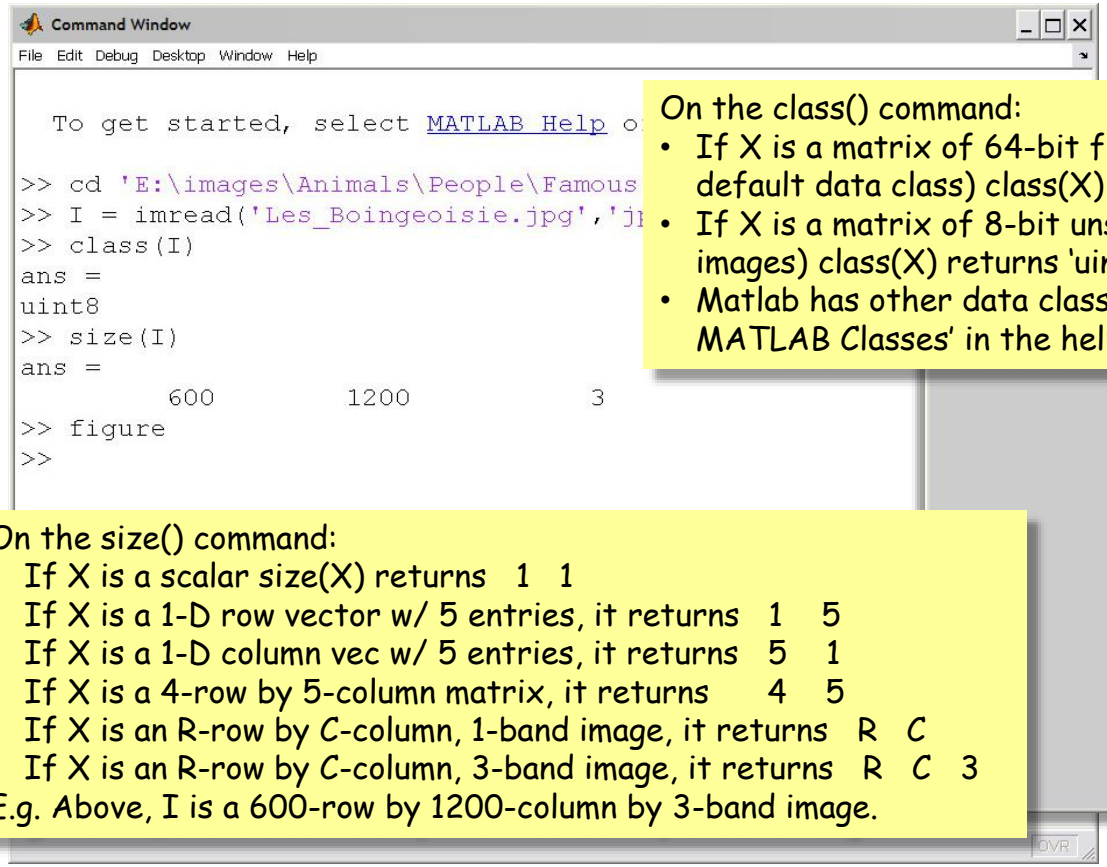
Read a Truecolor Image into Matlab





- First, change to the directory that holds the image.
- Then load the image into an a matrix labeled 'I'. (It could be any valid Matlab name.)
- Display the image's dimensions using the size() command.
- Then, create a figure window.

Read a Truecolor Image into Matlab



On the class() command:
- If X is a matrix of 64-bit floating-point numbers (Matlab's default data class) class(X) returns 'double' (without the quotes).
- If X is a matrix of 8-bit unsigned integers (like most common images) class(X) returns 'uint8').
- Matlab has other data classes, too. Search for 'Fundamental MATLAB Classes' in the help window for a description of them.

Related to the size() command: length(X)
- returns the size of the largest dimension of X.
- length(X(:)) returns the number of elements in X.

On the size() command:
- If X is a scalar size(X) returns 1 1
- If X is a 1-D row vector w/ 5 entries, it returns 1 5
- If X is a 1-D column vec w/ 5 entries, it returns 5 1
- If X is a 4-row by 5-column matrix, it returns 4 5
- If X is an R-row by C-column, 1-band image, it returns R C
- If X is an R-row by C-column, 3-band image, it returns R C 3

E.g. Above, I is a 600-row by 1200-column by 3-band image.

Read a Truecolor Image into Matlab

Matlab scales the image to fit the figure window so both the images size and aspect ratio are usually incorrect.

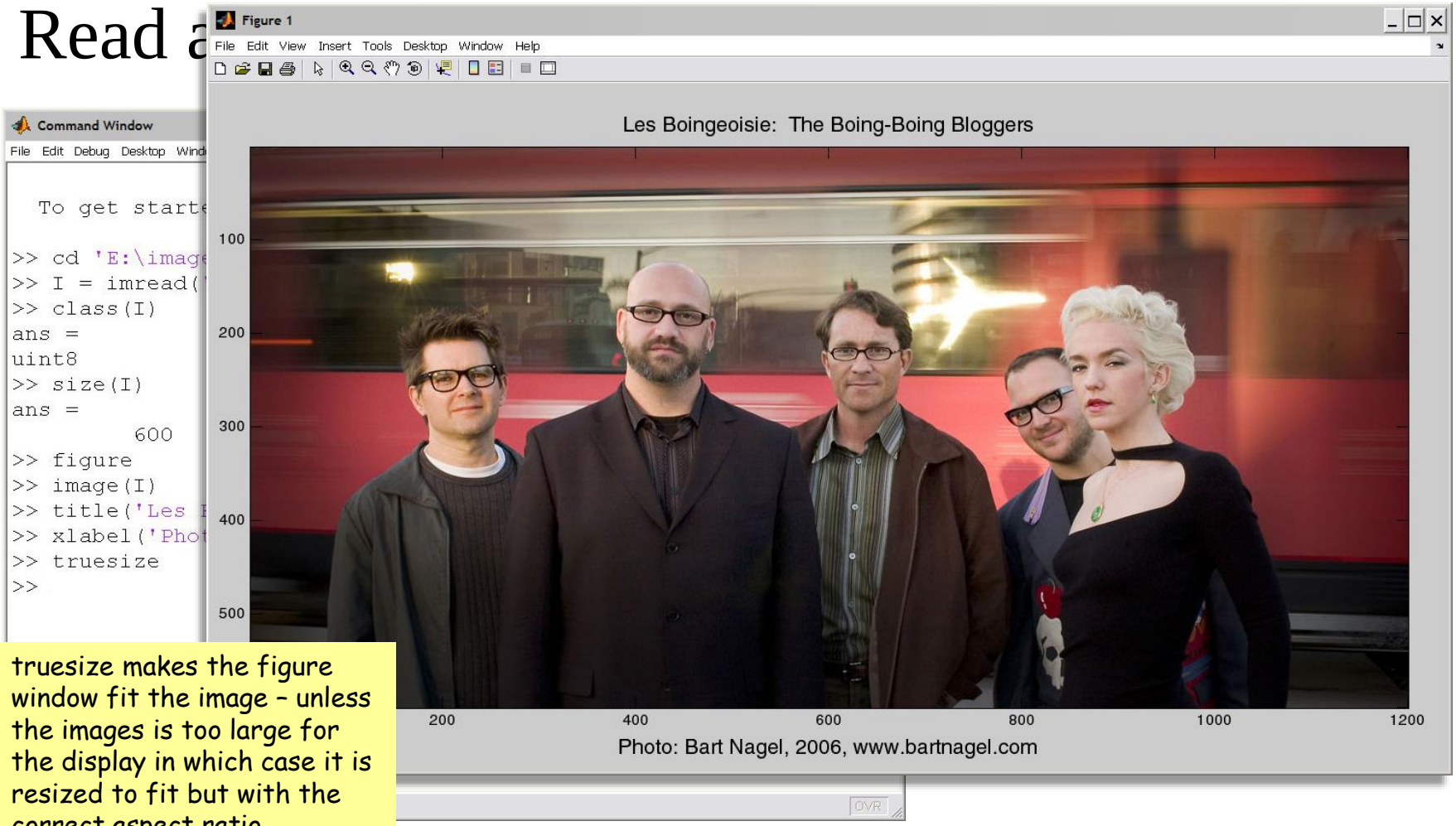
```
Command Window
File  Edit  Debug  Desktop  Window  Help

  To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'E:\images\Animals\People\Famous'
>> I = imread('Les_Boingeoisie.jpg','jpg');
>> class(I)
ans =
uint8
>> size(I)
ans =
         600        1200           3
>> figure
>> image(I)
>> title('Les Boingeoisie:  The Boing-Boing Bloggers')
>> xlabel('Photo: Bart Nagel, 2006, www.bartnagel.com')
>>
```

- Display the image in the figure window using image().
- Place a title at the top.
- Put a label under the x-axis using xlabel().
- ylabel() puts a label to the left of the y-axis, oriented 90° from horizontal

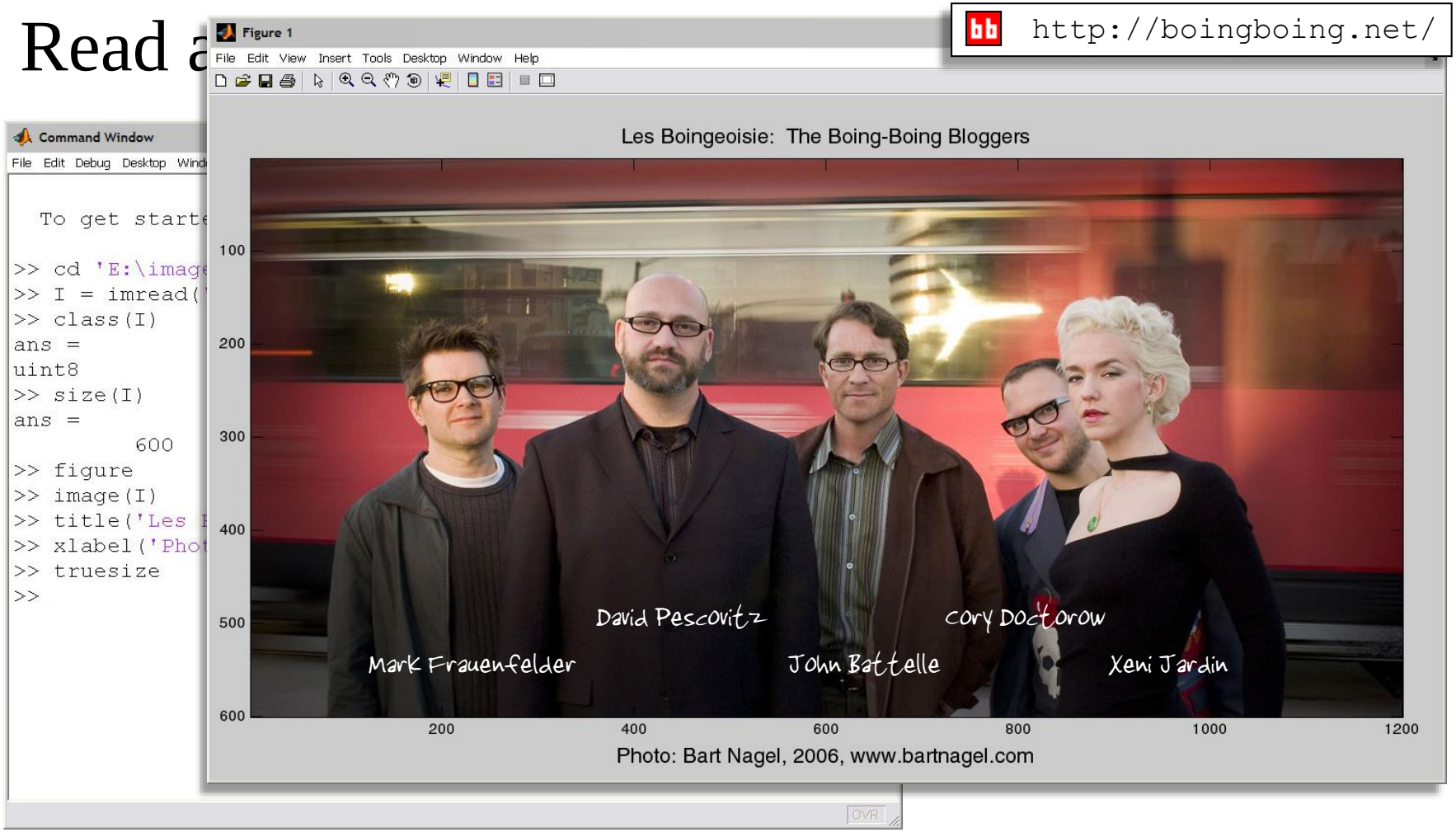
Read a



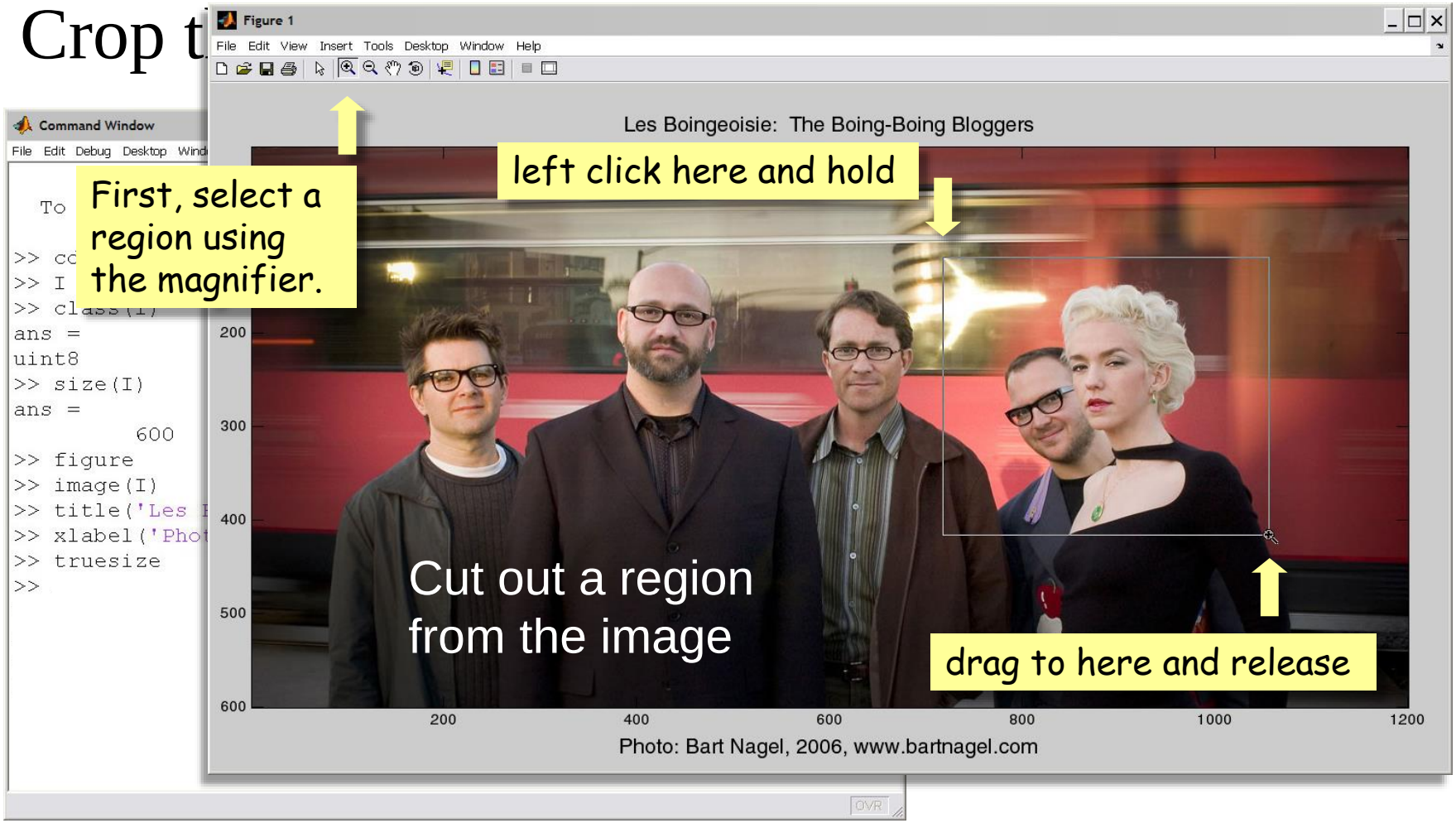
truesize makes the figure window fit the image – unless the images is too large for the display in which case it is resized to fit but with the correct aspect ratio.

Read a

http://boingboing.net/



Crop t



Crop t

From this close-up we can estimate the coordinates of the region:

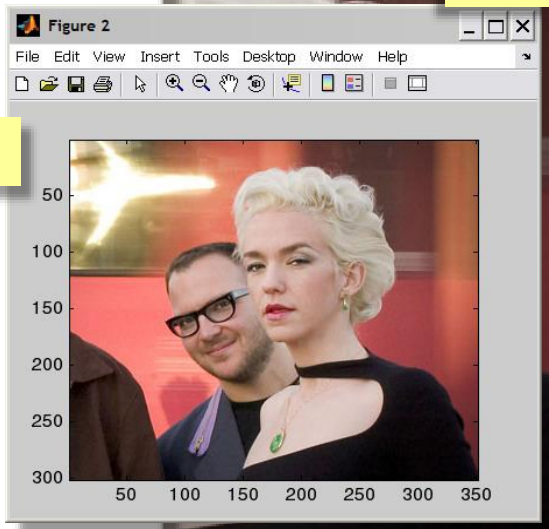
Crop t

```
To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'E:\images\Animals\People\Famous'
>> I = imread('Les_Boingeoisie.jpg','jpg');
>> class(I)
ans =
uint8
>> size(I)
ans =
         600        1200           3
>> figure
>> image(I)
>> title('Les Boingeoisie:  The Boing-Boing Bloggers')
>> xlabel('Photo: Bart Nagel, 2006, www.bartnagel.com')
>> truesize
>> J = I(125:425,700:1050,:);
>> figure
>> image(J)
>> truesize
>>
```

I(:) means all I's elements in a vertical list: 1st col followed by 2nd, etc. until the 1st band is catenated. Same procedure is applied to remaining bands for one long list.

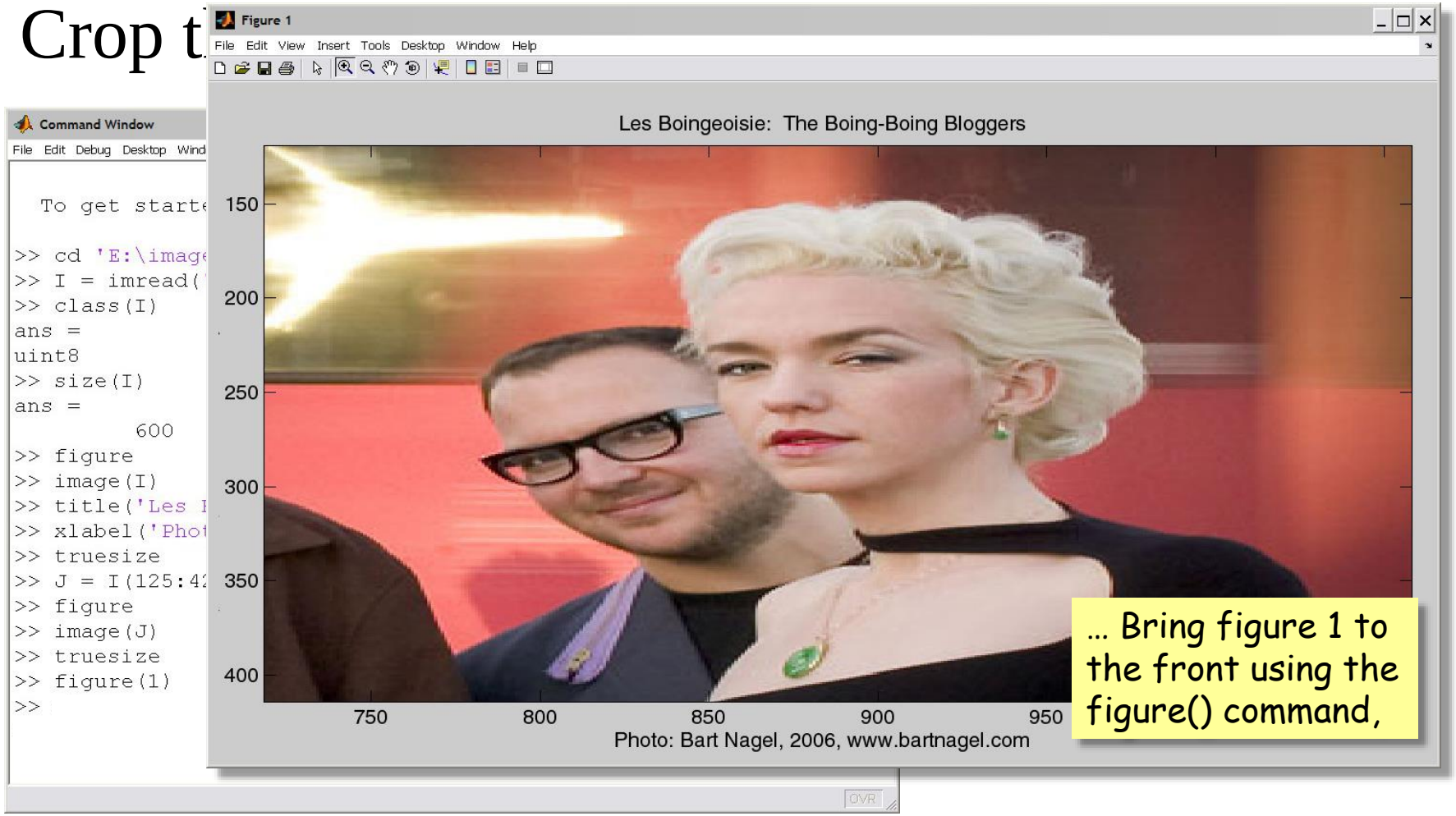
Here it is:



J = I(first row:last row, first column:last column, all bands)
':' alone means 'all bands' here. In the 1st position, ':' means 'all rows'; in the 2nd it means 'all columns'.

Now to close figure 1…

Crop t



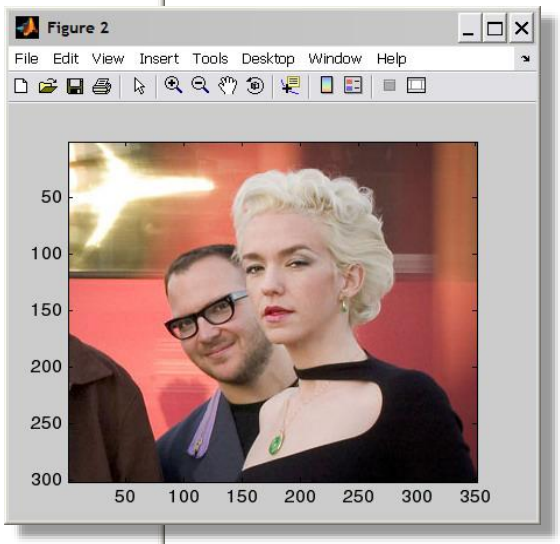
Crop the Image

Command Window

File Edit Debug Desktop Window Help

```
  To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'E:\images\Animals\People\Famous'
>> I = imread('Les_Boingeoisie.jpg','jpg');
>> class(I)
ans =
uint8
>> size(I)
ans =
         600        1200           3
>> figure
>> image(I)
>> title('Les Boingeoisie:  The Boing-Boing Bloggers')
>> xlabel('Photo: Bart Nagel, 2006, www.bartnagel.com')
>> truesize
>> J = I(125:425,700:1050,:);
>> figure
>> image(J)
>> truesize
>> figure(1)
>> close
>>
```



then type 'close' at the prompt.

Read a Colormapped Image into Matlab

Command Window

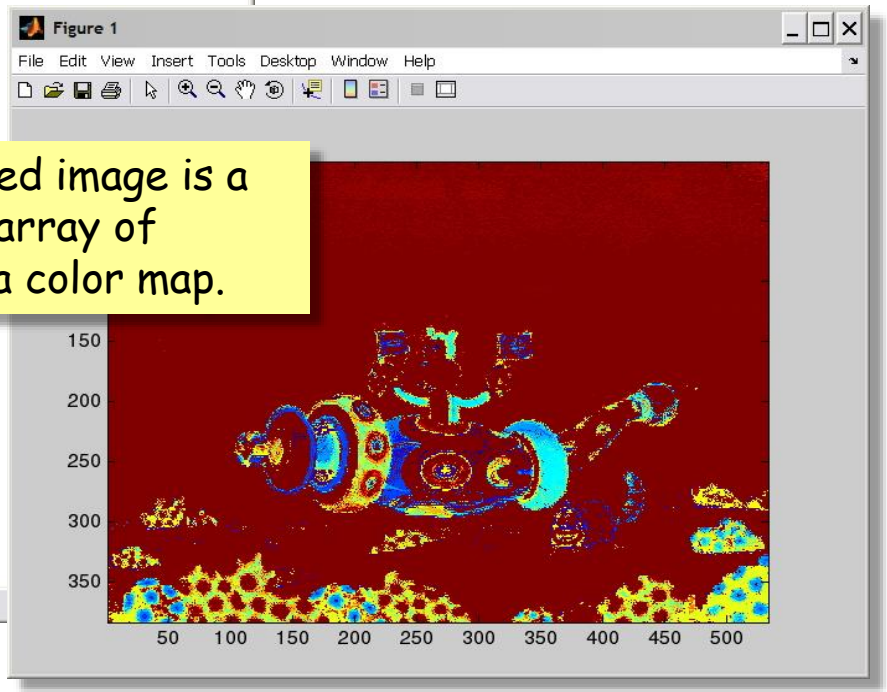
File Edit Debug Desktop Window Help

```
  To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'D:\classes\EECE253\Fall 2006\graphics\matlab intro'
>> [I,cmap] = imread('Jim Woodring - PlusMinus.gif','gif');
>> figure
>> image(I)
>> class(I)
ans =
uint8
>> size(I)
ans =
   383   533
>>
```

A colormap is an n X 3 table of color values. 8-bits per pixel implies n = 256 (at most) colors out of 256^3 = 16777216

A colormapped image is a rectangular array of indices into a color map.



Matlab assumes that a 1-band, 8-bit / pixel image is colormapped. Initial display uses a default colormap which is highly likely to be incorrect for any given image.

Read a Colormapped Image into Matlab

```
Command Window
File  Edit  Debug  Desktop  Window  Help

  To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'D:\classes\EECE253\Fall 2006\graphics\matlab intro'
>> [I,cmap] = imread('Jim Woodring - PlusMinus.gif','gif');
>> figure
>> image(I)
>> class(I)
ans =
uint8
>> size(I)
ans =
   383   533
>> colormap(cmap)
>> title('Plus Minus');
>> xlabel('Jim Woodring (http://www.jimwoodring.com/)'
>> truesize
>>
```

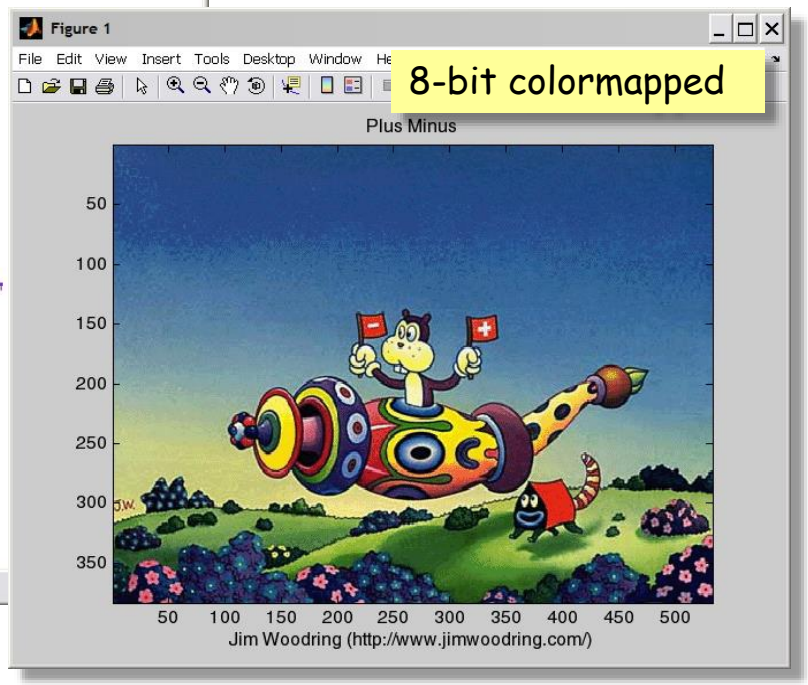
Read the correct colormap from the image file along with the index array.

[I, cmap]=imread(…

I = index array

cmap = colormap

Display the index array using image(). Then load the colormap using colormap().



8-bit colormapped

Colormapped vs. Truecolor in Matlab

Command Window

File Edit Debug Desktop Window Help

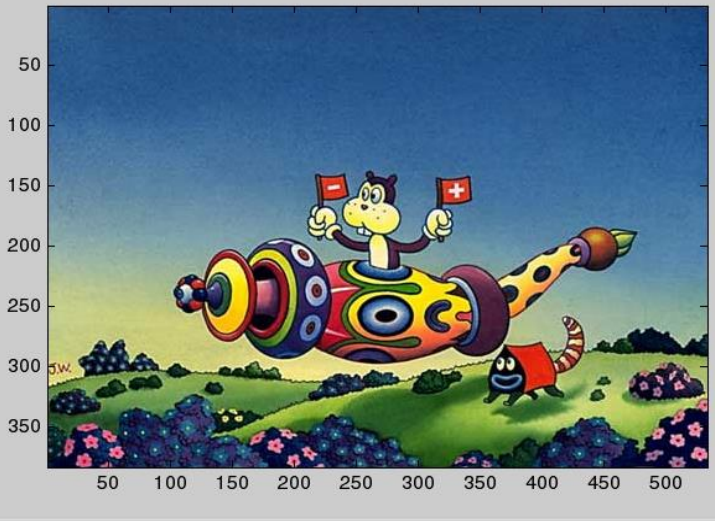
```
  To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'D:\classes\EECE253\Fall 2006\graphics\matlab intro'
>> [I,cmap] = imread('Jim Woodring - PlusMinus.gif','gif');
>> figure
>> image(I)
>> class(I)
ans =
uint8
>> size(I)
ans =
   383   533
>> colormap(cmap)
>> title('Plus Minus');
>> xlabel('Jim Woodring (http://www.jimwoodring.
>> truesize
>> T = imread('Jim Woodring - PlusMinus.jpg','jp
>> figure
>> image(T)
>> truesize
>>
```

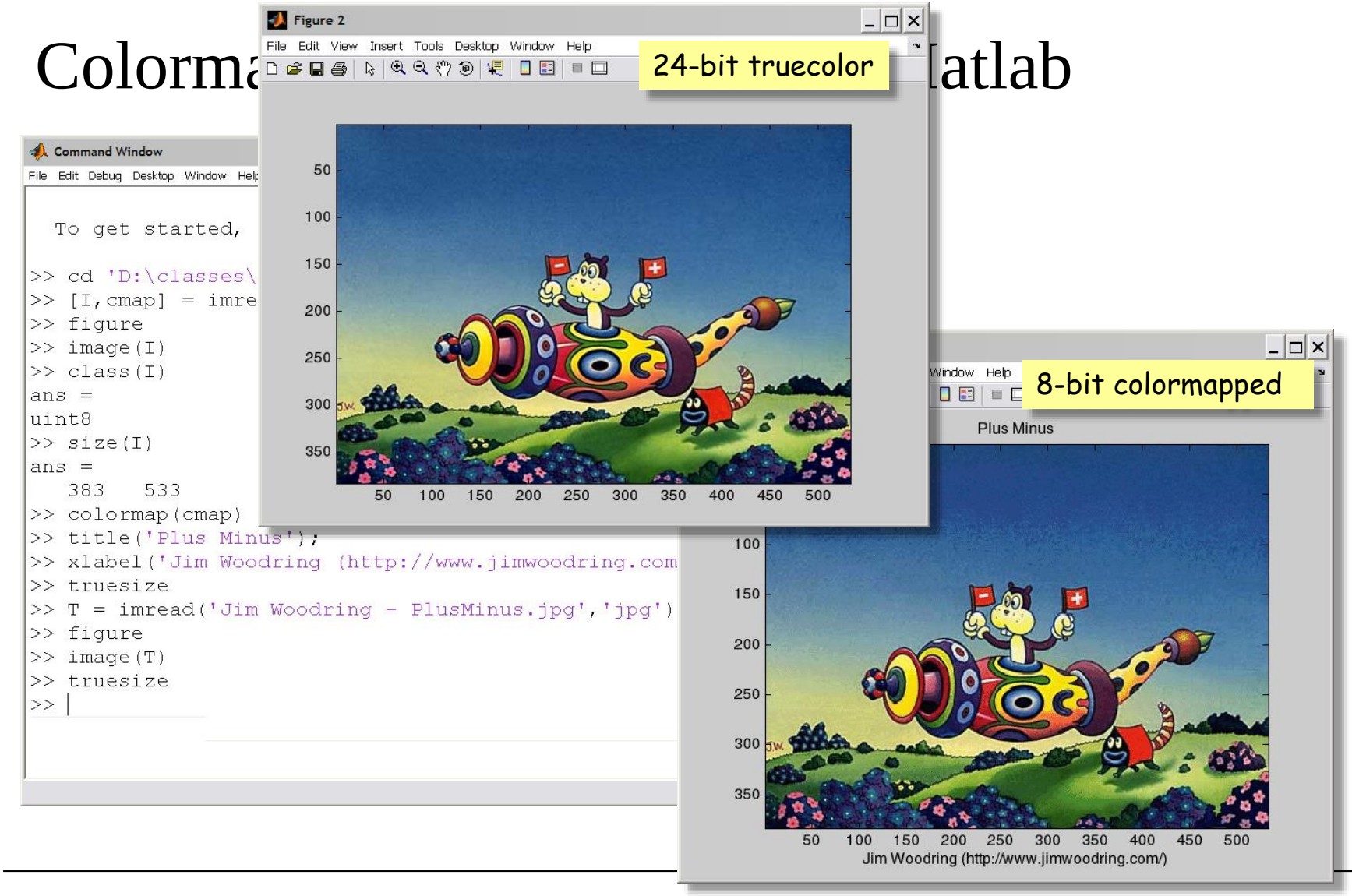
Figure 2

File Edit View Insert Tools Desktop Window Help

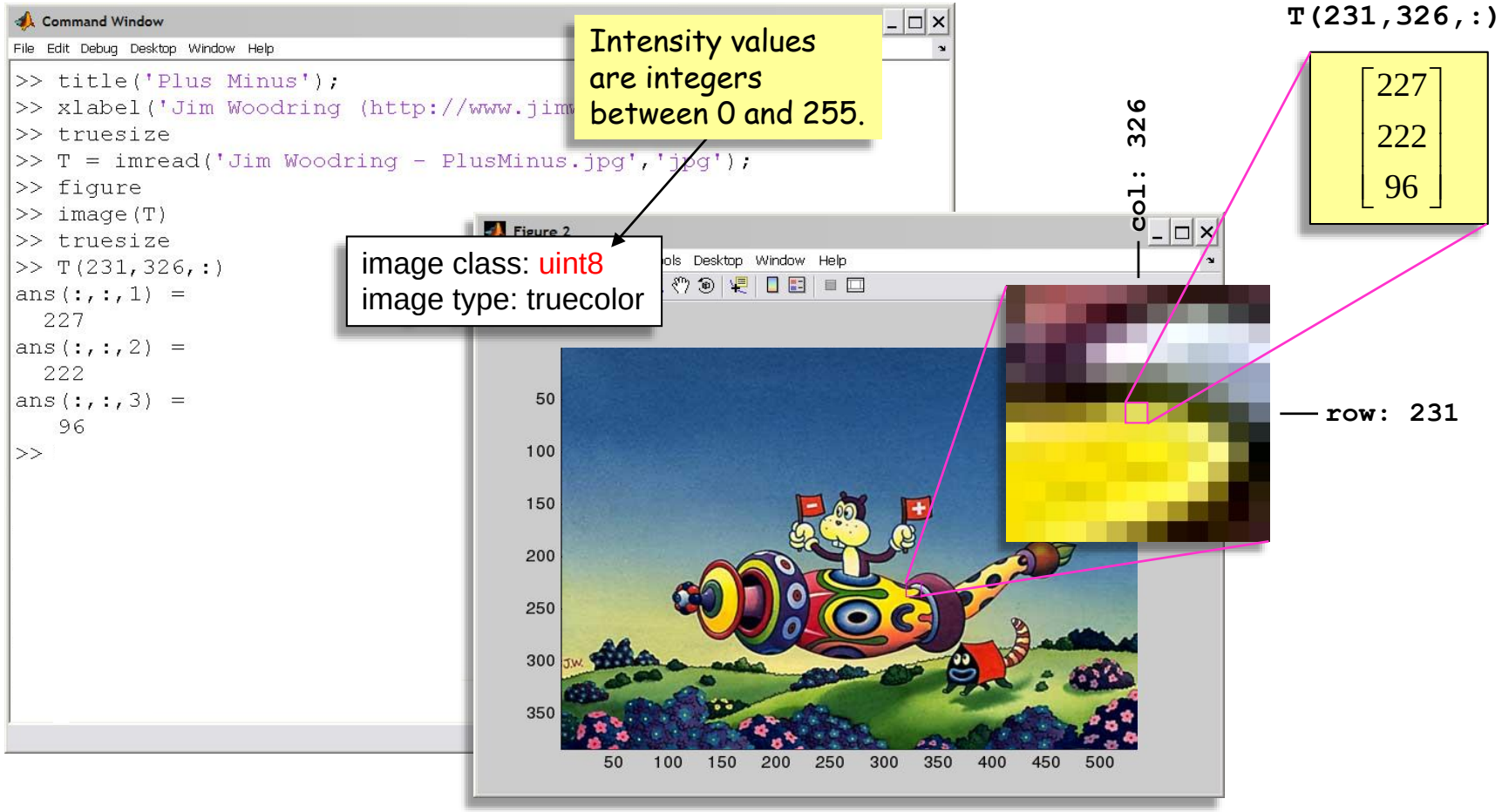
24-bit truecolor



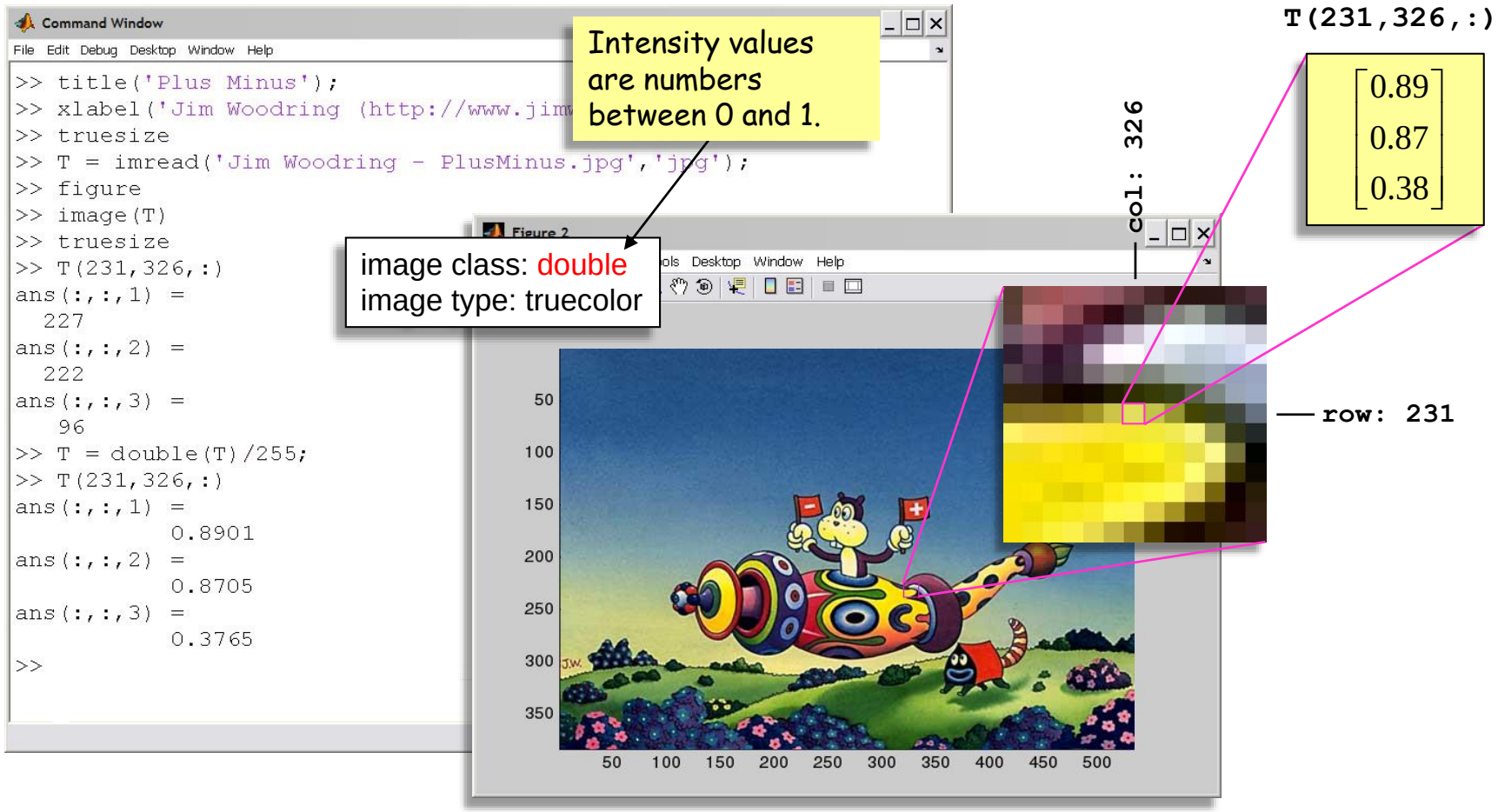
Colorma[illegible]atlab

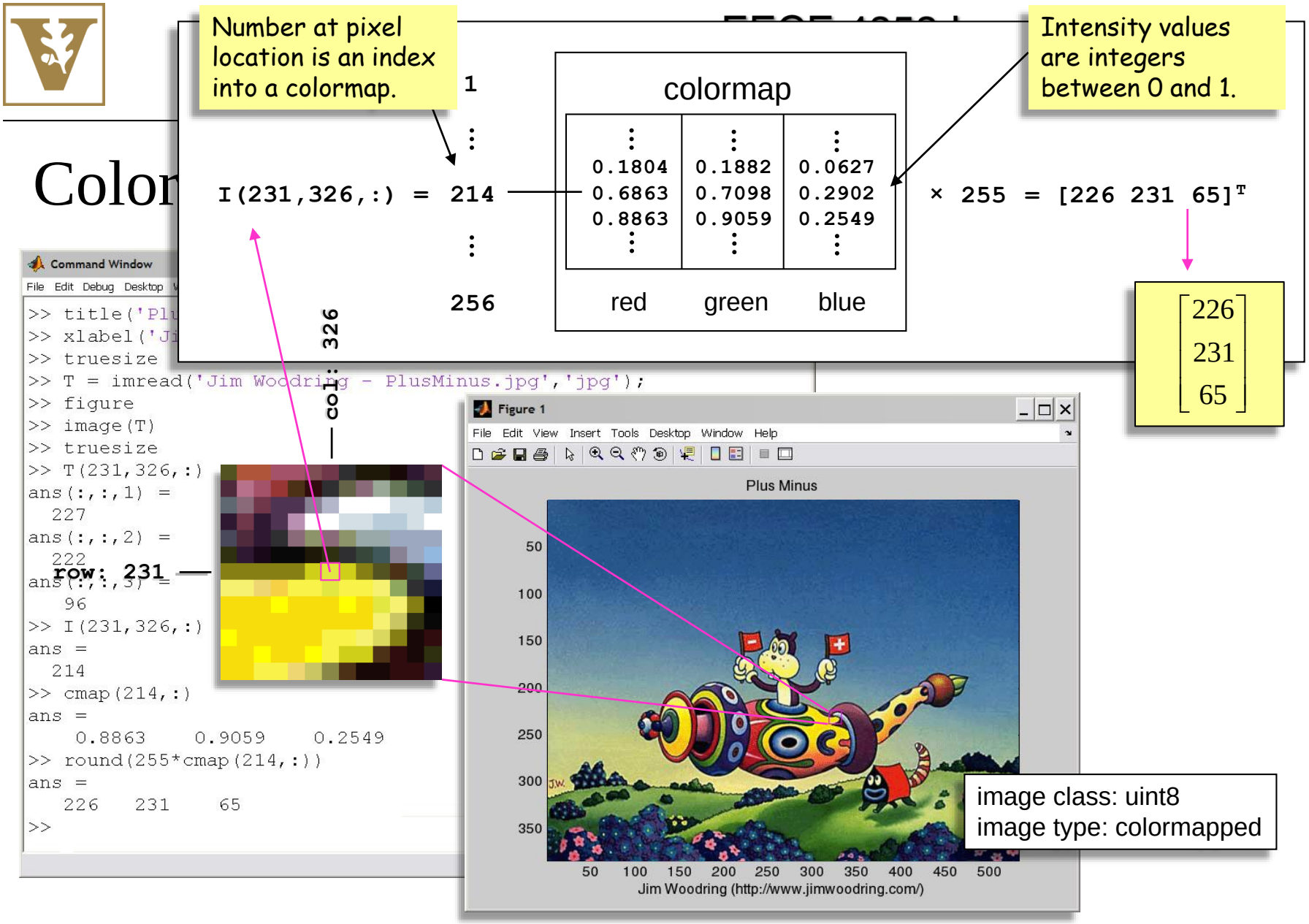


Colormapped vs. Truecolor in Matlab



Colormapped vs. Truecolor in Matlab



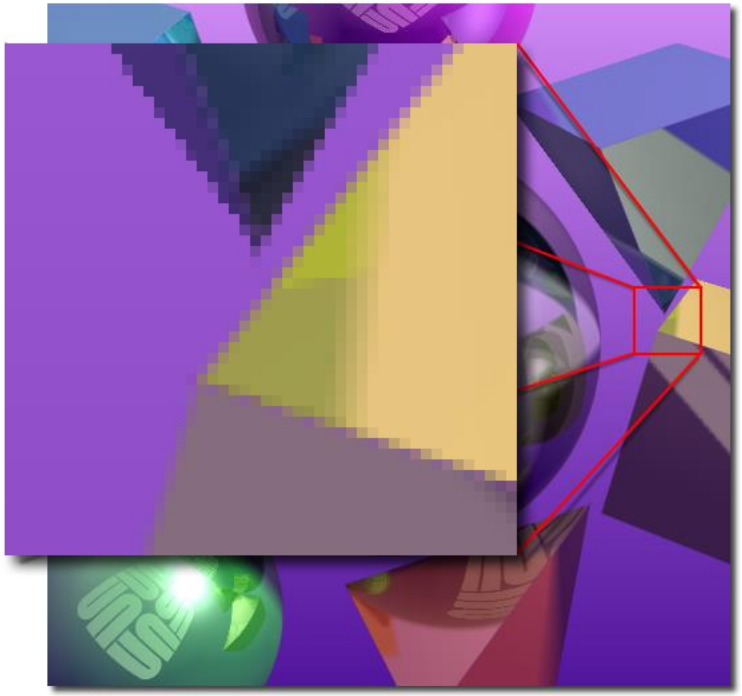


Example truecolor and colormapped images

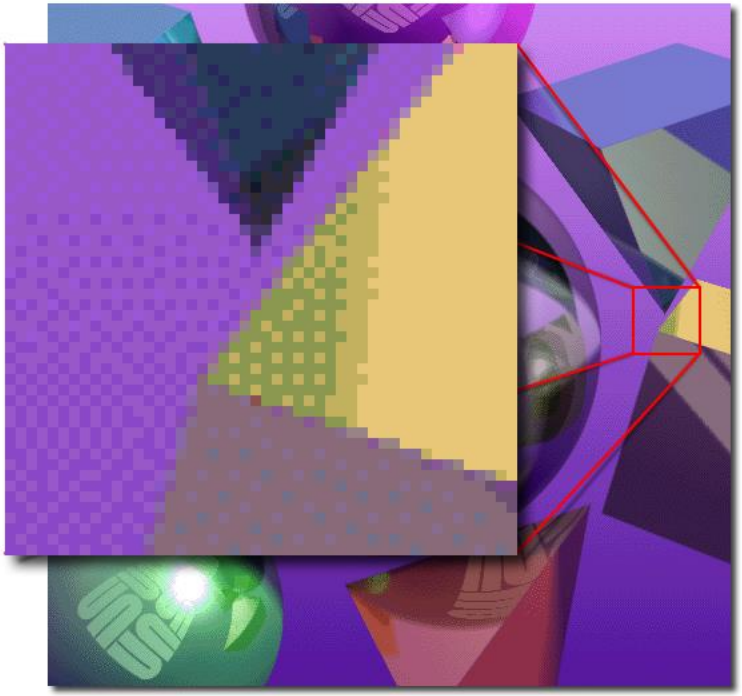
24-bit truecolor

8-bit colormapped to 24 bits

Example truecolor and colormapped images



24-bit truecolor



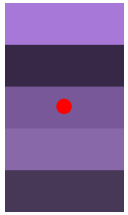
8-bit colormapped to 24 bits

Colormapped Image, Indices, & Color Map

```
>> [M,CMAP] = imread('button_mapped.bmp','bmp);
```

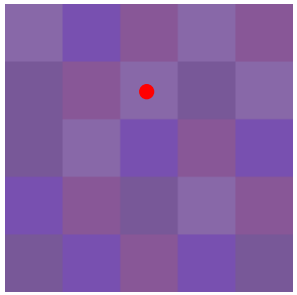
Indices contained in M(254:258,254:258)

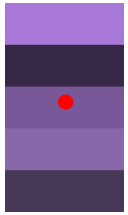
⋮	⋮	⋮	⋮	⋮	
…	111	121	48	111	48 …
…	110	48	111	110	111 …
…	110	111	121	48	121 …
…	121	48	110	111	48 …
…	110	121	48	121	110 …
⋮	⋮	⋮	⋮	⋮	



actual values in CMAP(109:113,:)

⋮	R ⋮	G ⋮	B ⋮
109	0.6588	0.4706	0.8471
110	0.2196	0.1569	0.2824
111	0.4706	0.3451	0.5961
112	0.5333	0.4078	0.6588
113	0.2824	0.2196	0.3451
⋮	⋮	⋮	⋮





255*CMAP(109:113,:)

⋮	R⋮	G⋮	B⋮
109	168	120	216
110	56	40	72
111	120	88	152
112	136	104	168
113	72	56	88
⋮	⋮	⋮	⋮

Last 3 cols. only

How to convert a colormapped image to true color

M is a 512x512x1, 8-bit image. It has 262,144 pixels. Each pixel has a value between 0 & 255.

cmap is the colormap that is stored in 'button_mapped.bmp' along with image. cmap is a 256x3 type-double matrix, each row of which lists a color in terms of its R, G, & B intensities, which are given as fractions between 0 and 1.

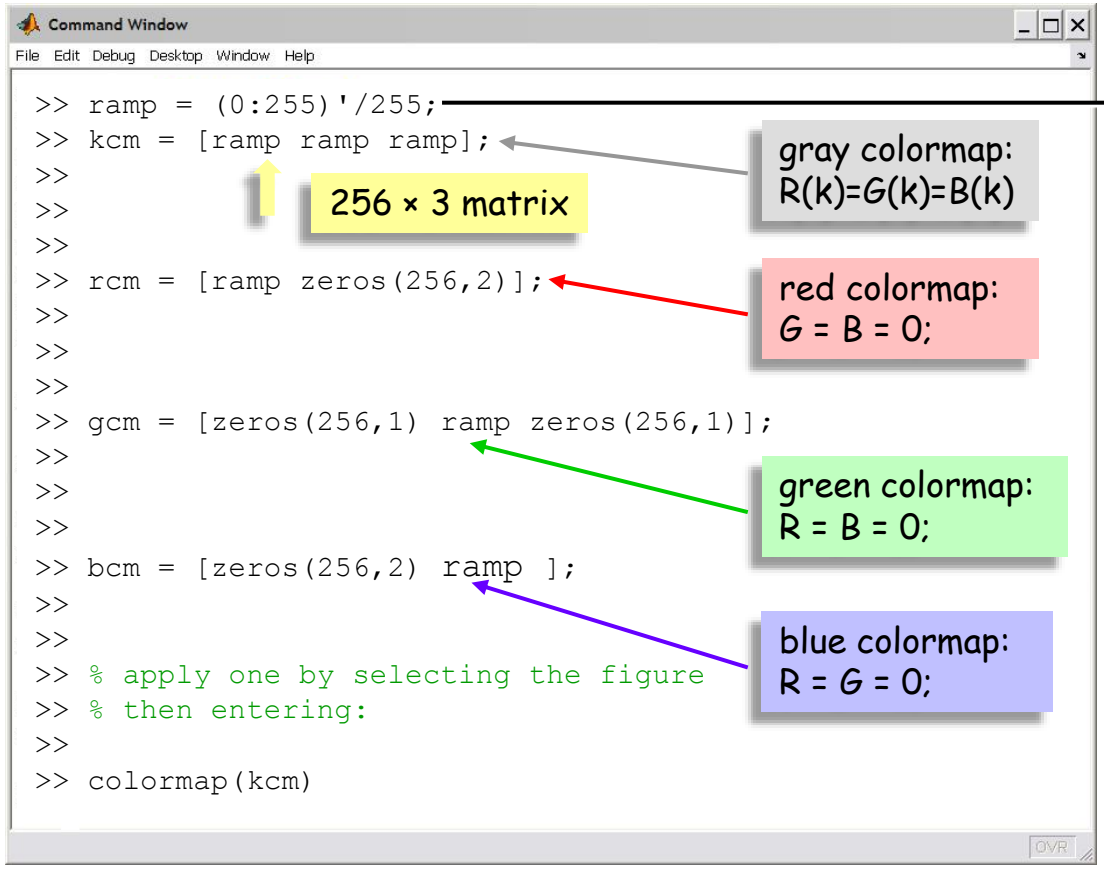
```
>> [M,cmap] = imread('button_mapped.bmp','bmp');
>> T = uint8(reshape(cmap(M+1,:),[size(M) 3])*255);
```

[512 512]

The 262,144 x 3 matrix of intensity values is reshaped into a 512x512x3 image of type double. The values are scaled to lie between 0 & 255 then converted to type uint8.

By concatenating M's columns, Matlab rearranges M into a 262,144 x 1 list. Each number in the list (if it has 1 added to it) refers to a row of the colormap. Then, cmap(M+1,:) produces a 262,144 x 3 matrix of intensity values of type double between 0 & 1.

How to Make Colormaps

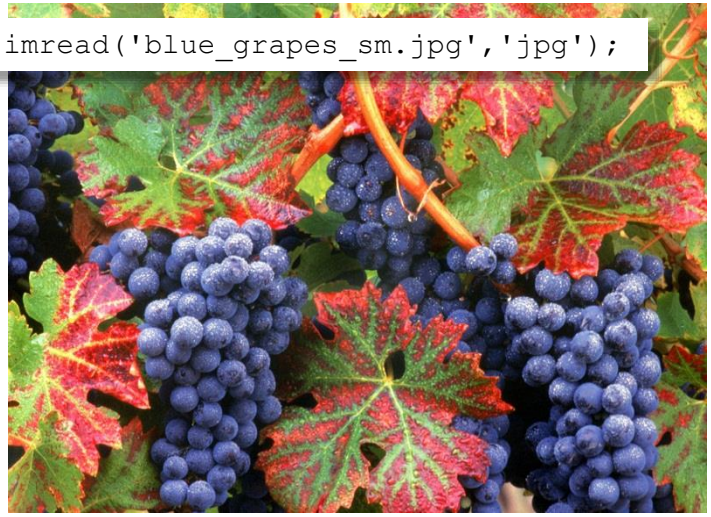


This code, 0:255 , generates a 1 row by 256 element vector of class double that contains numbers 0 through 255 inclusive.

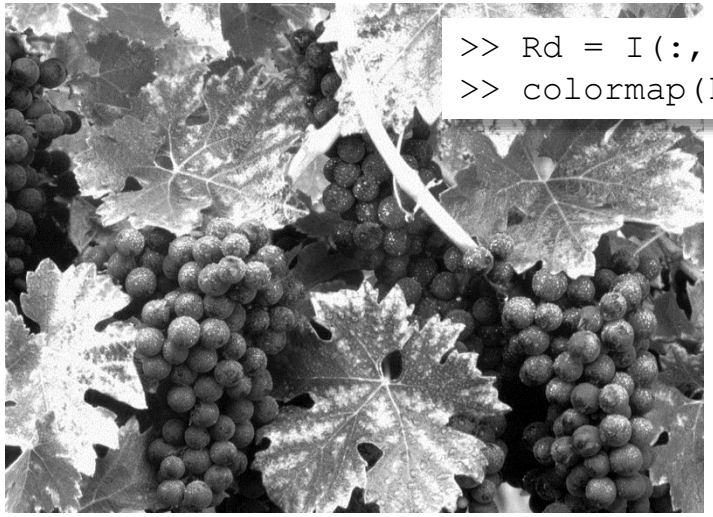
This, (0:255)' , has the same contents and class but is a 256 row by 1 column vector. The apostrophe (') is the matrix transpose operator.

R, G, & B bands of a truecolor image displayed with grayscale colormaps

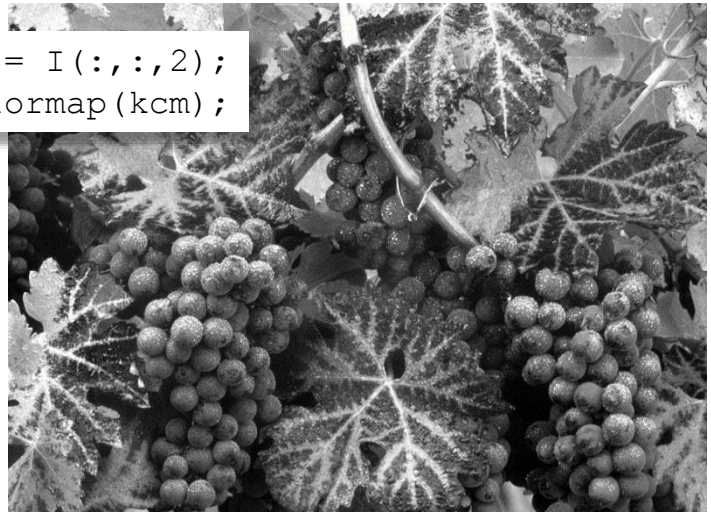
```
>> I = imread('blue_grapes_sm.jpg','jpg');
```



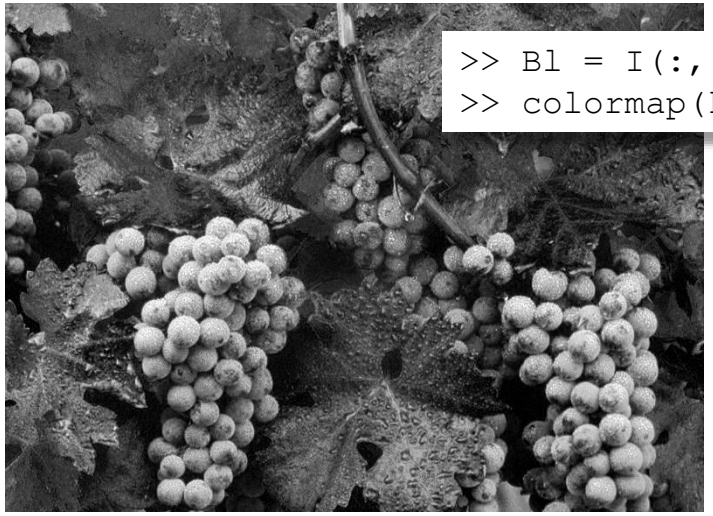
```
>> Rd = I(:,:,1);
>> colormap(kcm);
```



```
>> Gn = I(:,:,2);
>> colormap(kcm);
```



```
>> Bl = I(:,:,3);
>> colormap(kcm);
```



R, G, & B bands of a truecolor image displayed with grayscale colormaps

```
>> I =
```

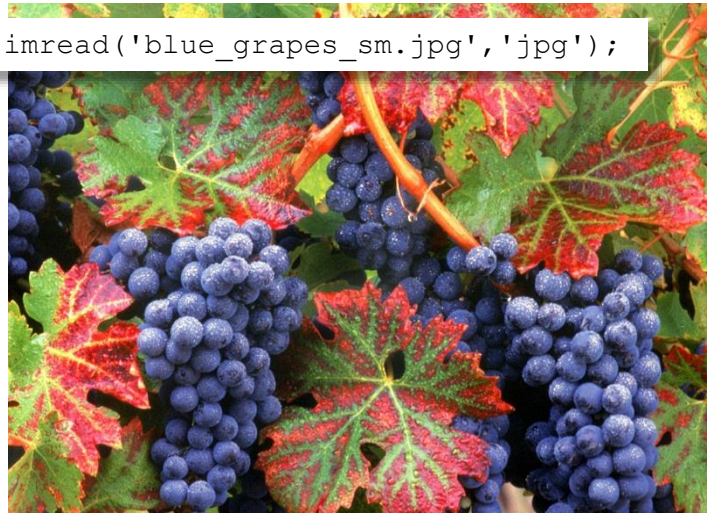

```
:,1);
kcm);
```

```
>> Gn
>> col
```


```
:,3);
kcm);
```

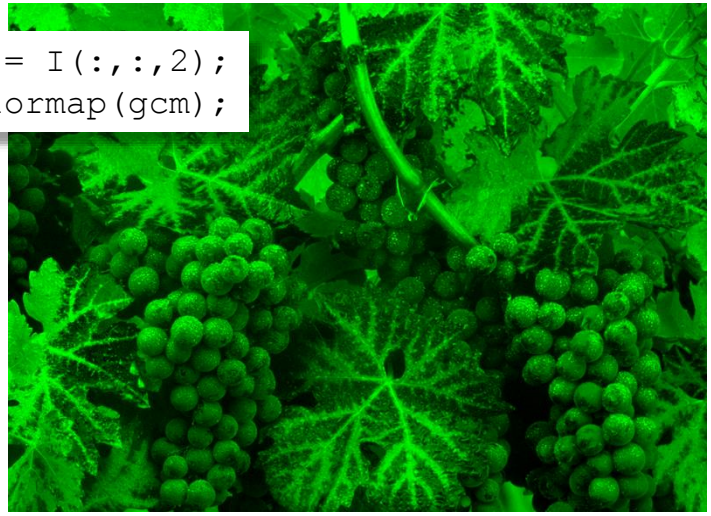
R, G, & B bands of a truecolor image displayed with tinted colormaps

```
>> I = imread('blue_grapes_sm.jpg','jpg');
```

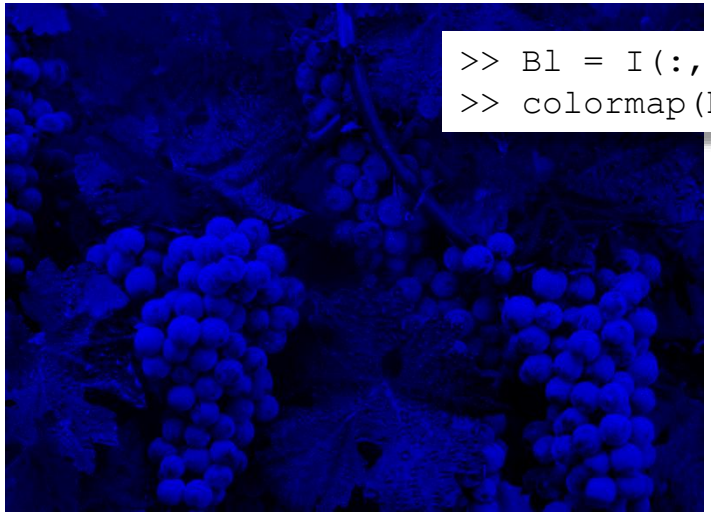


```
>> Rd = I(:,:,1);
>> colormap(rcm);
```

```
>> Gn = I(:,:,2);
>> colormap(gcm);
```



```
>> Bl = I(:,:,3);
>> colormap(bcm);
```

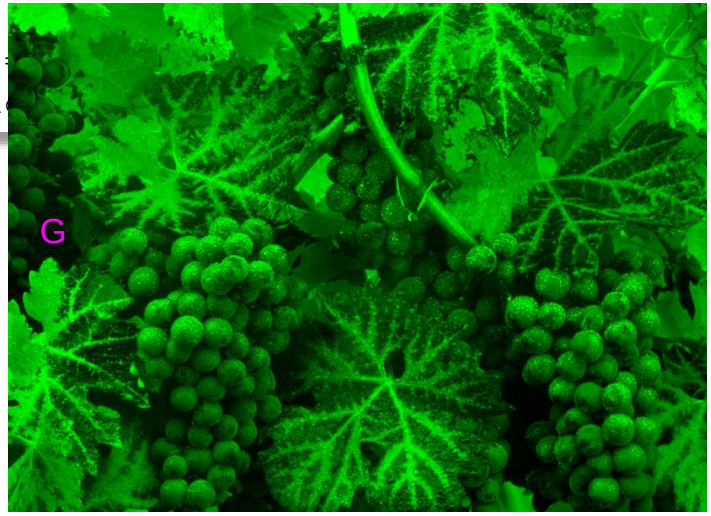


R, G, & B bands of a truecolor image displayed with tinted colormaps

```
>> I =
```

```
:,1);
rcm);
```


```
>> Gn
>> col
```




```
:,3);
bcm);
```

R, G, & B bands of a truecolor image displayed with grayscale colormaps

```
>> I =
```


```
:,1);
kcm);
```

```
>> Gn
>> col
```


```
:,3);
kcm);
```

Saving Images as Files

Command Window

File Edit Debug Desktop Window Help

```
>>
>> % truecolor as .bmp
>> imwrite(I,'image_name.bmp','bmp');
>>
>> % truecolor as .jpg (default quality = 75)
>> imwrite(I,'image_name.jpg','jpg');
>>
>> % truecolor as .jpg (quality = 100)
>> imwrite(I,'image_name.jpg','jpg','Quality',100);
>>
>> % colormapped as .bmp
>> imwrite(I,cmap,'image_name.bmp','bmp');
>>
>> % colormapped as .gif
>> imwrite(I,cmap,'image_name.gif','gif');
>>
```

OVR

Assuming that
'I' contains the image of the correct class,
that
'cmap' is a colormap,
and that
'image_name' is the file-name that you want.

Double Exposure: Adding Two Images

Jim Woodring - Bumperillo

Mark Rayden – The Ecstasy of Cecelia

Rayden Woodring – The Ecstasy of Bumperillo (?)

Double Exposure: Adding Two Images

Example Matlab Code

```
>> cd 'D:\Classes\EECE253\Fall 2006\Graphics\matlab intro'
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Double Exposure: Adding Two Images

```
>> cd 'D:\Classes\EECE253\Fall 2006\Graphics\matlab intro'
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Example Matlab Code

Cut a section out of the middle of the larger image the same size as the smaller image.

Double Exposure: Adding Two Images

```
>> cd 'D:\Classes\EECE253\Fall 2006\Graphics\matlab intro'
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Example Matlab Code

Note that the images are averaged, pixelwise.

Double Exposure: Adding Two Images

```
>> cd 'D:\Classes\EECE253\Fall 2006\Graphics\matlab intro'
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Example Matlab Code

Note the data class conversions.

Intensity Masking: Multiplying Two Images

Jim Woodring - Bumperillo

Mark Rayden – The Ecstasy of Cecelia

Rayden Woodring – Bumperillo Ecstasy (?)

Intensity Masking: Multiplying Two Images

Example Matlab Code

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Extacsy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
>> JWtimesMR = double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:)).*double(MR);
>> M = max(JWtimesMR(:));
>> m = min(JWtimesMR(:));
>> JWtimesMR = uint8(255*(double(JWtimesMR)-m)/(M-m));
>> figure
>> image(JWtimesMR)
>> truesize
>> title('EcstasyBumperillo')
```

Intensity Masking: Multiplying Two Images

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Extacsy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
>> JWtimesMR = double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:)).*double(MR);
>> M = max(JWtimesMR(:));
>> m = min(JWtimesMR(:));
>> JWtimesMR = uint8(255*(double(JWtimesMR)-m)/(M-m));
>> figure
>> image(JWtimesMR)
>> truesize
>> title('EcstasyBumperillo')
```

Example Matlab Code

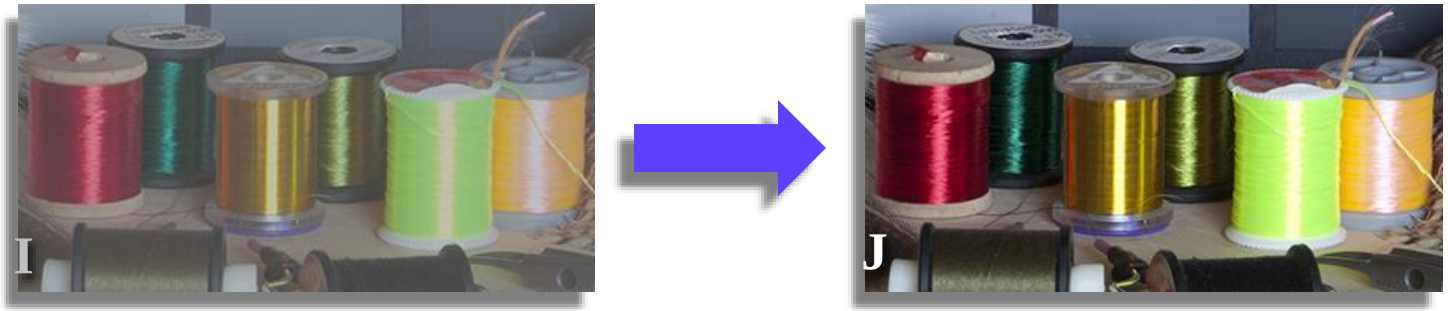
Note that the images are multiplied, pixelwise.

Note how the image intensities are scaled back into the range 0-255.

A note on image intensity scaling

In the code on the previous slide we scaled the result of the image multiplication so that it would have the maximum possible dynamic range (0-255). The formula for such linear scaling is

$$m = \min(\mathbf{I}), \quad M = \max(\mathbf{I}), \quad \text{and} \quad J = 255(\mathbf{I} - m)/(M - m).$$



In Matlab, if the images are of type `uint8`, this requires class conversions:

```
J = uint8(255*double(I-m)/double(M-m));
```

Without the `double` casts, the arithmetic is performed with 8 bits of precision, which yields incorrect results.

A note on image intensity scaling

In the code on the previous slide we scaled the result of the image multiplication so that it would have the m
In Matlab, the min and max are computed like this:
```
m = min(I(:)); M = max(I(:));
```
mula for such linear scaling is

$$m = \min(\mathbf{I}), \quad M = \max(\mathbf{I}), \quad \text{and} \quad J = 255(\mathbf{I} - m)/(M - m).$$

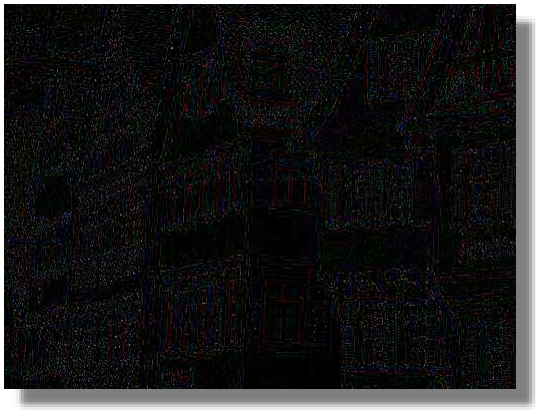
In Matlab, if the images are of type `uint8`, this requires class conversions:

```
J = uint8(255*double(I-m)/double(M-m));
```

Without the `double` casts, the arithmetic is performed with 8 bits of precision, which yields incorrect results.

More on image intensity scaling

Linear scaling is not always appropriate. If the image has some important features with very low intensity values and others that are large, the darker features may not be visible in a linearly scaled result. In such cases nonlinear scaling can perform better. The images below are differently scaled versions of `D = abs(I-J);` where **I** is the image to the left and **J** was a quality-0 jpeg copy of itself.



`uint8(255*(D-m)/(M-m))`

`L=log(D+1),uint8(255*L/max(L(:))`

`uint8(D.^2)`

Pixel Indexing in Matlab

"For" loops in Matlab are inefficient, whereas Matlab's native indexing procedures are very fast.

"IP_Function" is some arbitrary image processing function that you or someone else has written.

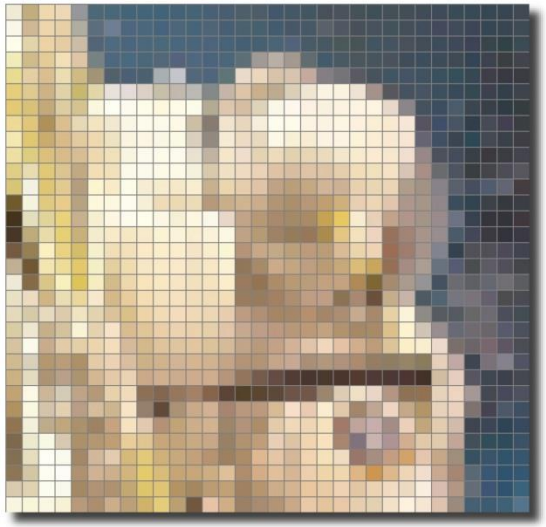
Rather than

```
for r = 1:R
   for c = 1:C
      J(r,c,:) = IP_Function(I(r,c,:));
   end
end
```

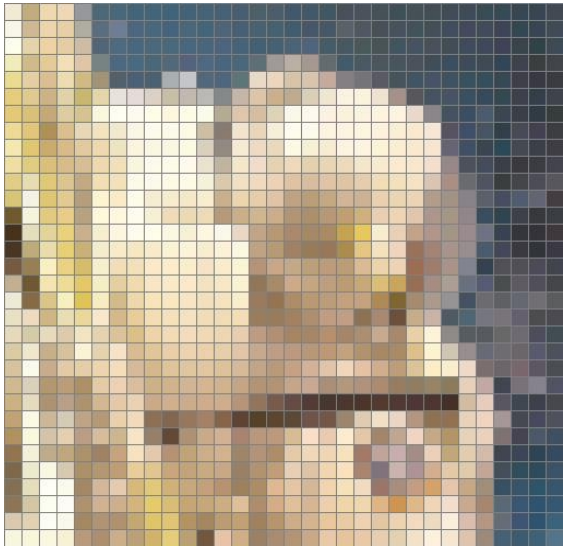
use, if possible

```
J = IP_Function(I);
```

But, sometimes that is not possible.
For example, if the output, **J**, is decimated with respect to the input, **I**, the above will not work (unless, of course, it is done within IP_function).



Pixel Indexing in Matlab



```
r = 1:n:R;
```

```
I(r,:,:)
```

```
c = 1:n:C;
```

```
I(:,c,:)
```

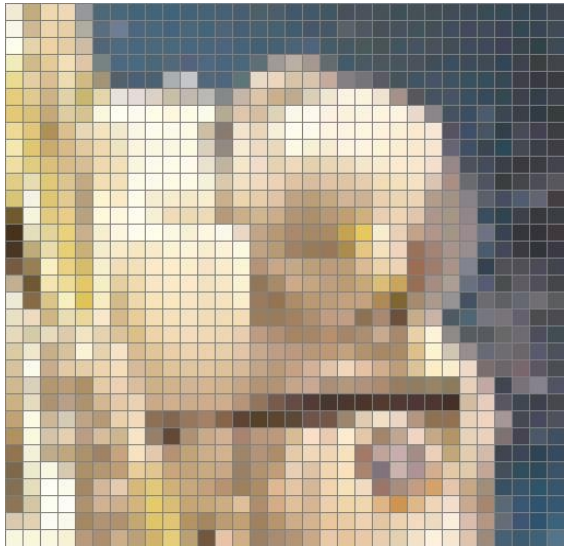
Here, n=3

r = [1 4 7 10 13 16 19 22 25 28 31]

c = [1 4 7 10 13 16 19 22 25 28 31]

To decimate the above image by a factor of *n*, create a vector, **r**, that contains the index of every *n*th row, and a similar vector, **c**.

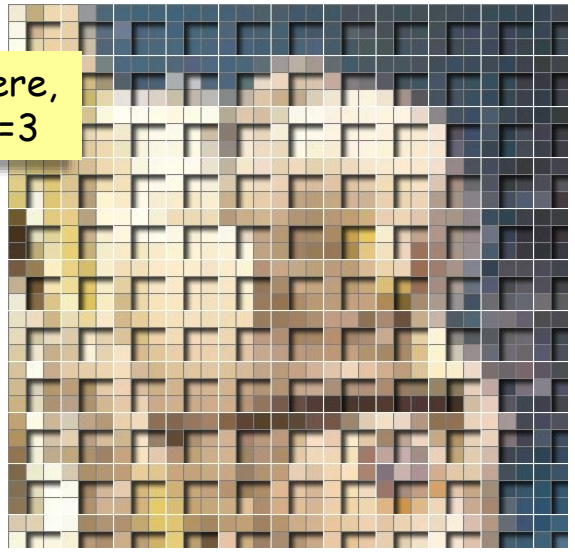
Pixel Indexing in Matlab

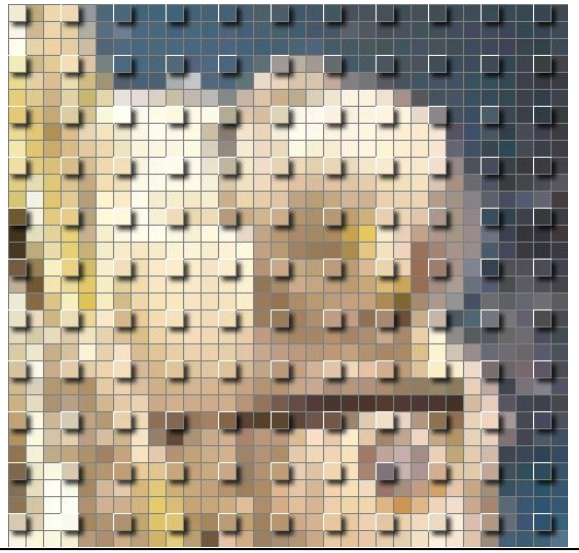


This is called, 'vectorizing'.

Take the pixels indexed by both *r* and *c*.

```
I(r,c)
```



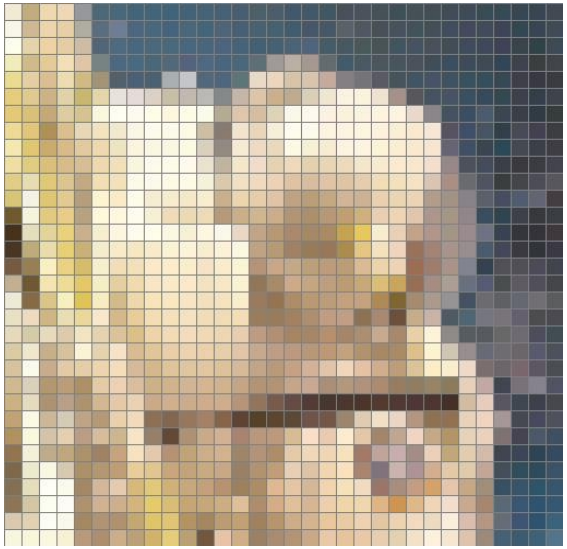


Here, n=3

Then, vectors **r** and **c** used as index arguments for image **I** select every *n*th column in every *n*th row.

Pixel Indexing in Matlab

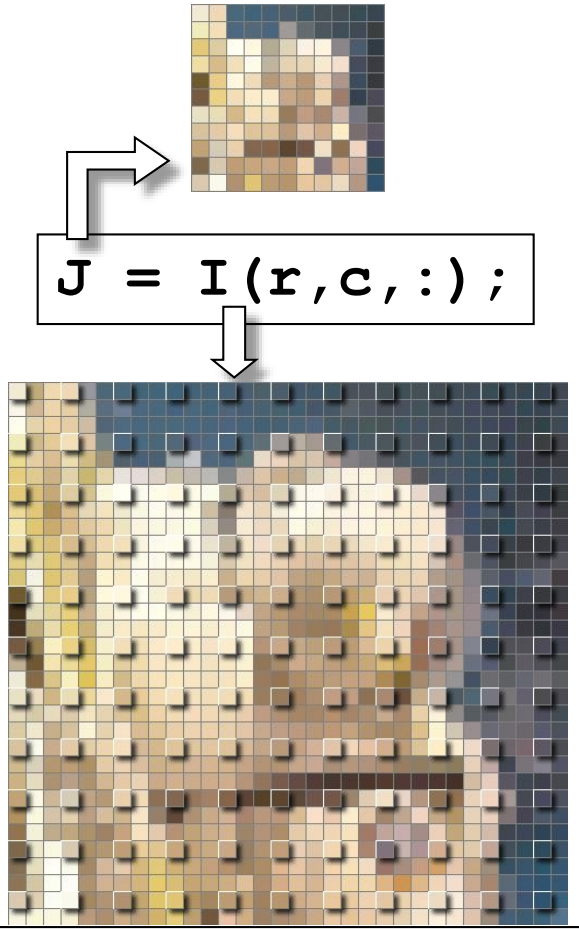
Here, n=3



image, `I`

```
r = 1:n:R;
```

```
c = 1:n:C;
```



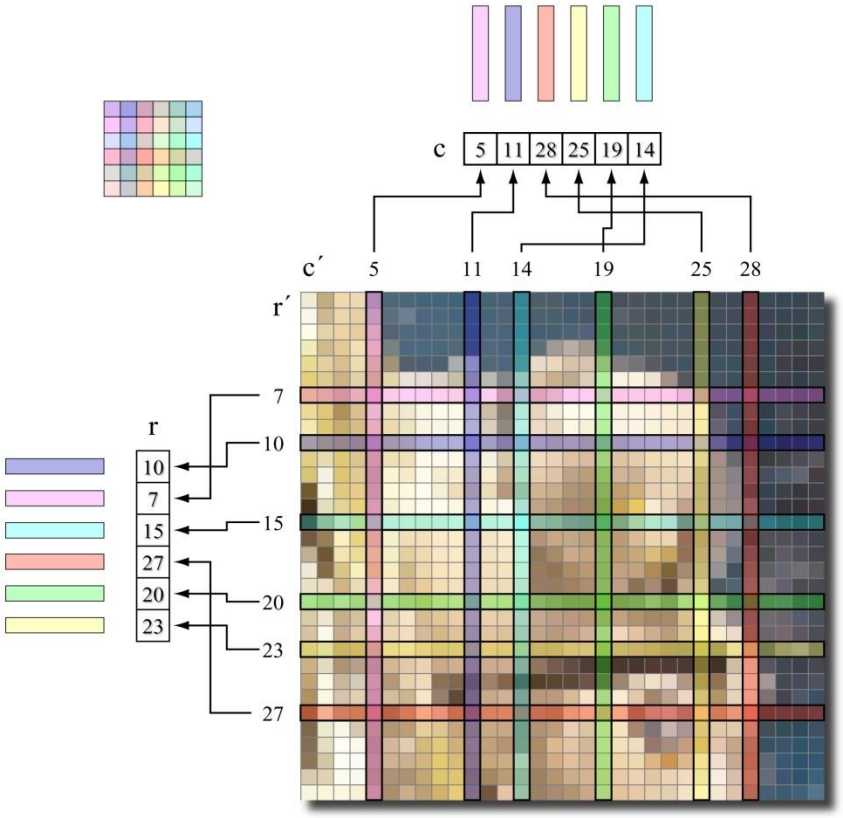
Pixel Indexing in Matlab

Indexing in Matlab is fully general.

If **I** is R x C x B, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of **I** (**r**,**c**) is an ordinal shuffling of the pixels from **I** as indexed by **r** and **c**.



Whenever possible, avoid using 'for' loops; vectorize instead.

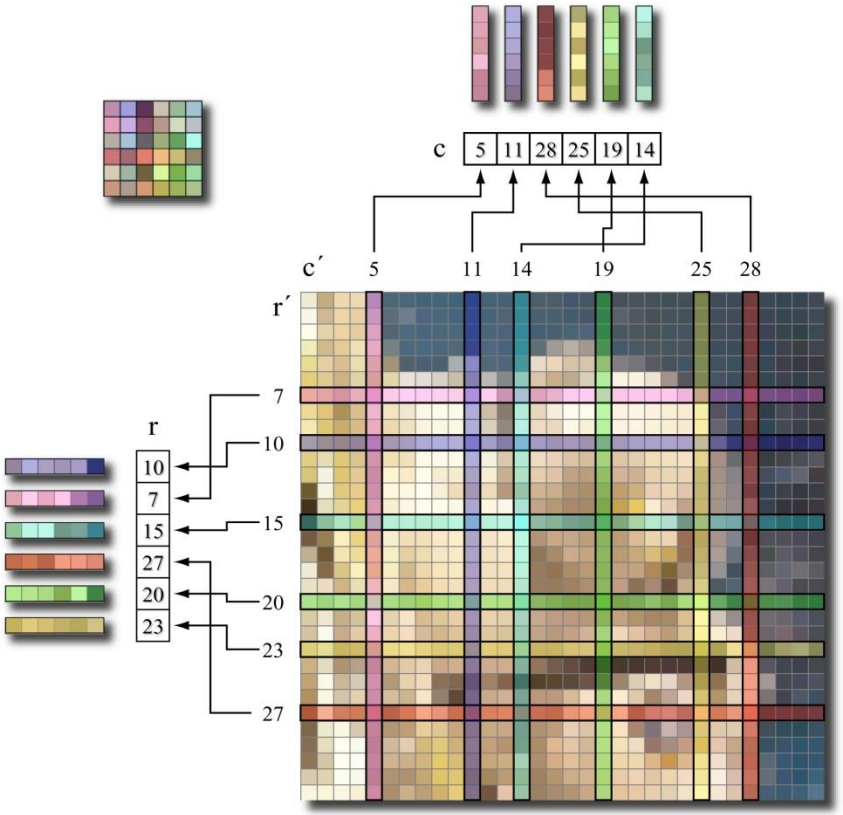
Pixel Indexing in Matlab

Indexing in Matlab is fully general.

If **I** is R x C x B, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of **I** (**r**,**c**) is an ordinal shuffling of the pixels from **I** as indexed by **r** and **c**.



Whenever possible, avoid using 'for' loops; vectorize instead.

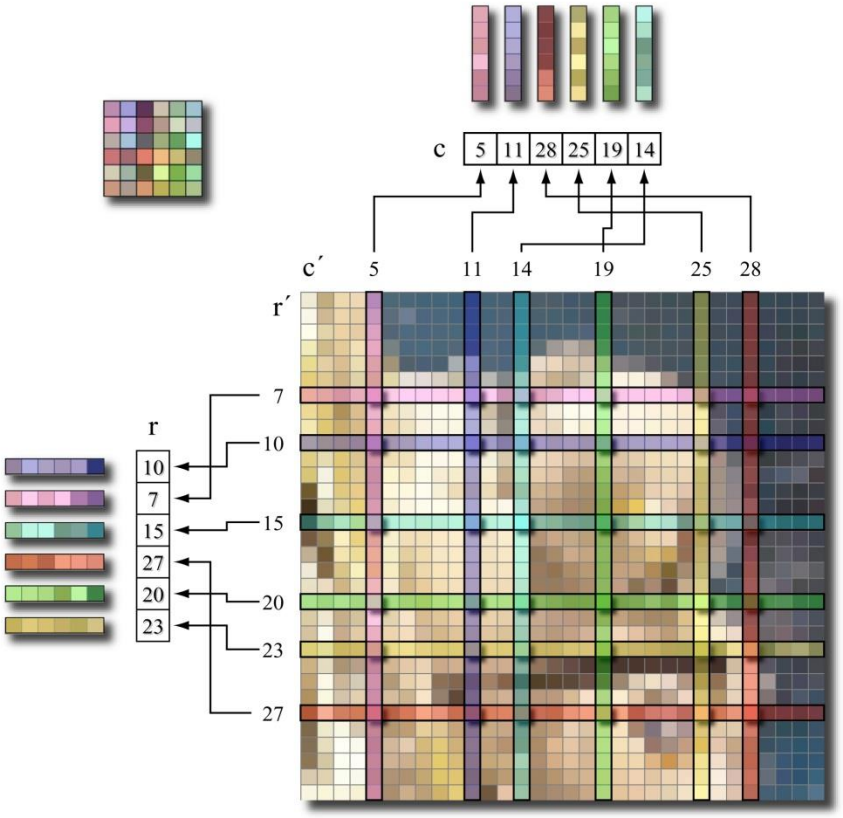
Pixel Indexing in Matlab

Indexing in Matlab is fully general.

If **I** is R x C x B, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of **I** (**r**,**c**) is an ordinal shuffling of the pixels from **I** as indexed by **r** and **c**.



Whenever possible, avoid using 'for' loops; vectorize instead.

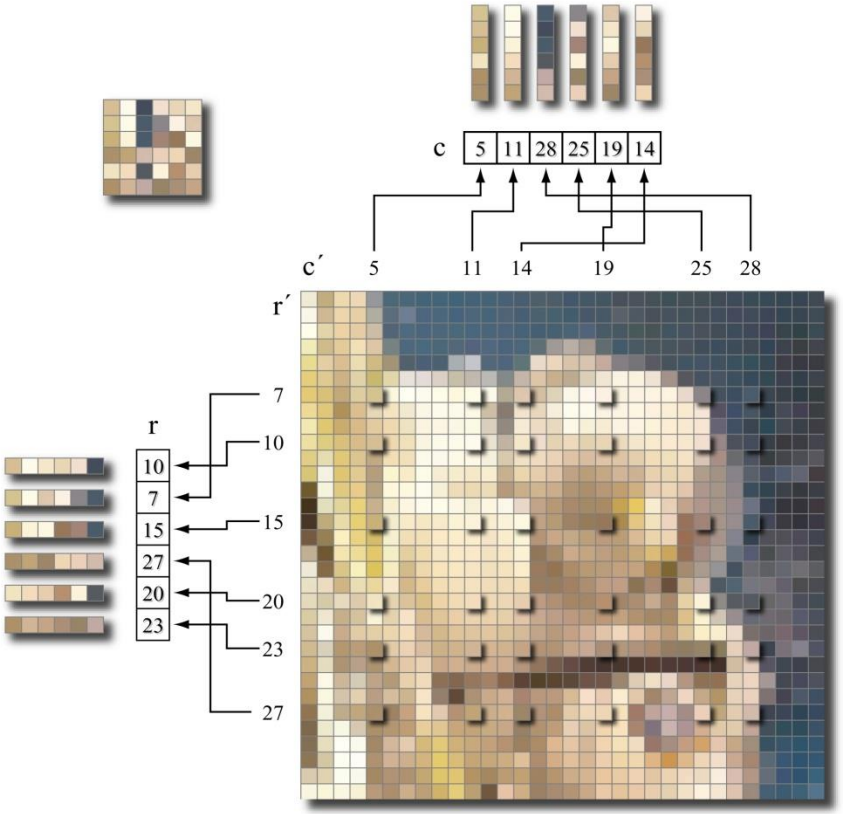
Pixel Indexing in Matlab

Indexing in Matlab is fully general.

If **I** is R x C x B, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of **I** (**r**,**c**) is an ordinal shuffling of the pixels from **I** as indexed by **r** and **c**.



Whenever possible, avoid using 'for' loops; vectorize instead.

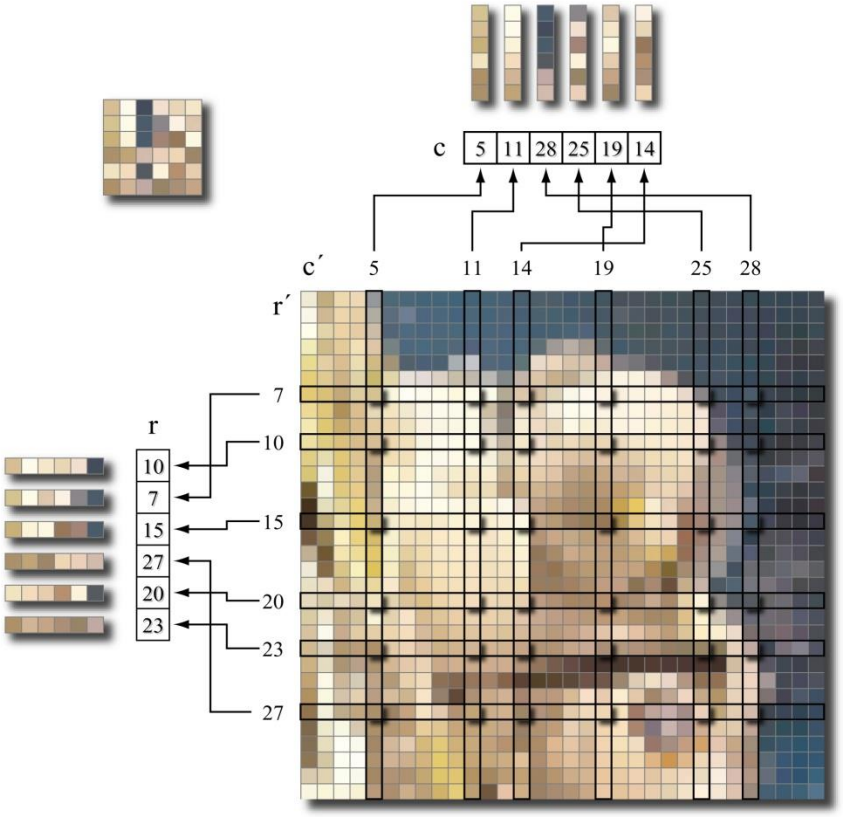
Pixel Indexing in Matlab

Indexing in Matlab is fully general.

If **I** is R x C x B, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of **I** (**r**,**c**) is an ordinal shuffling of the pixels from **I** as indexed by **r** and **c**.



Whenever possible, avoid using 'for' loops; vectorize instead.

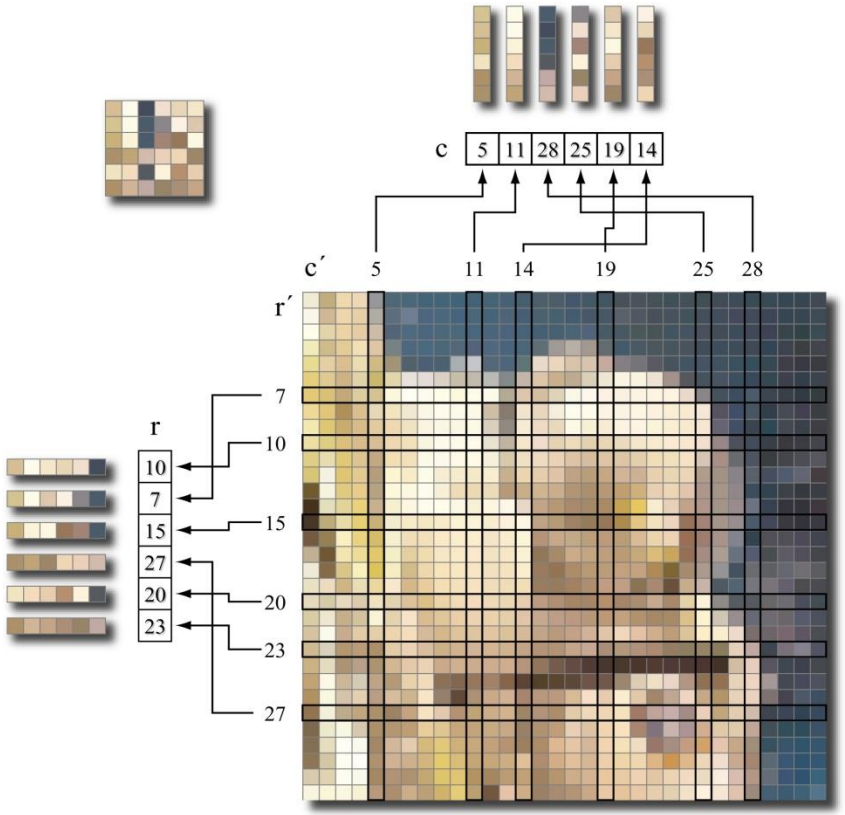
Pixel Indexing in Matlab

Indexing in Matlab is fully general.

If **I** is R x C x B, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

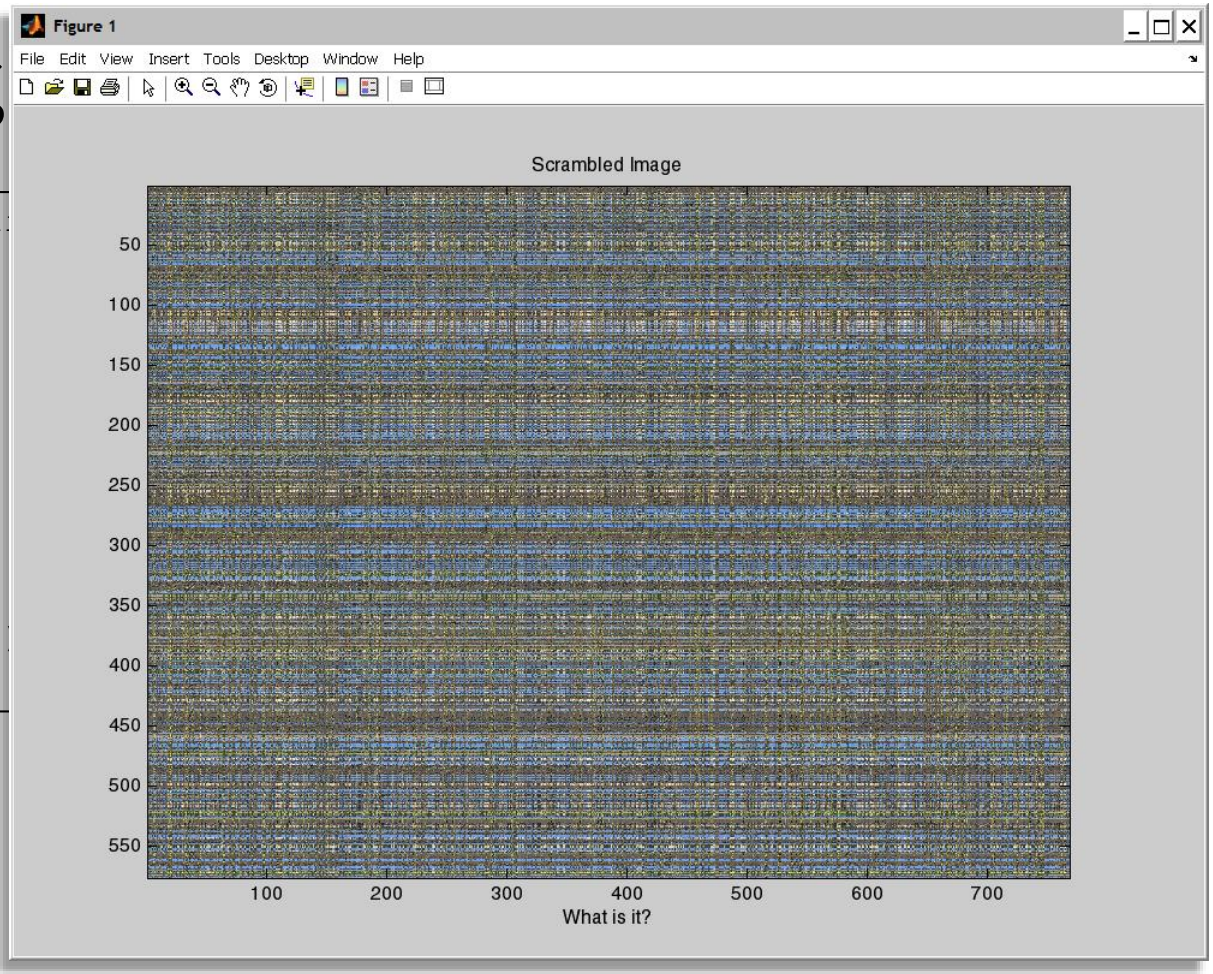
The result of **I** (**r**,**c**) is an ordinal shuffling of the pixels from **I** as indexed by **r** and **c**.



Whenever possible, avoid using 'for' loops; vectorize instead.

Pixel Indexing

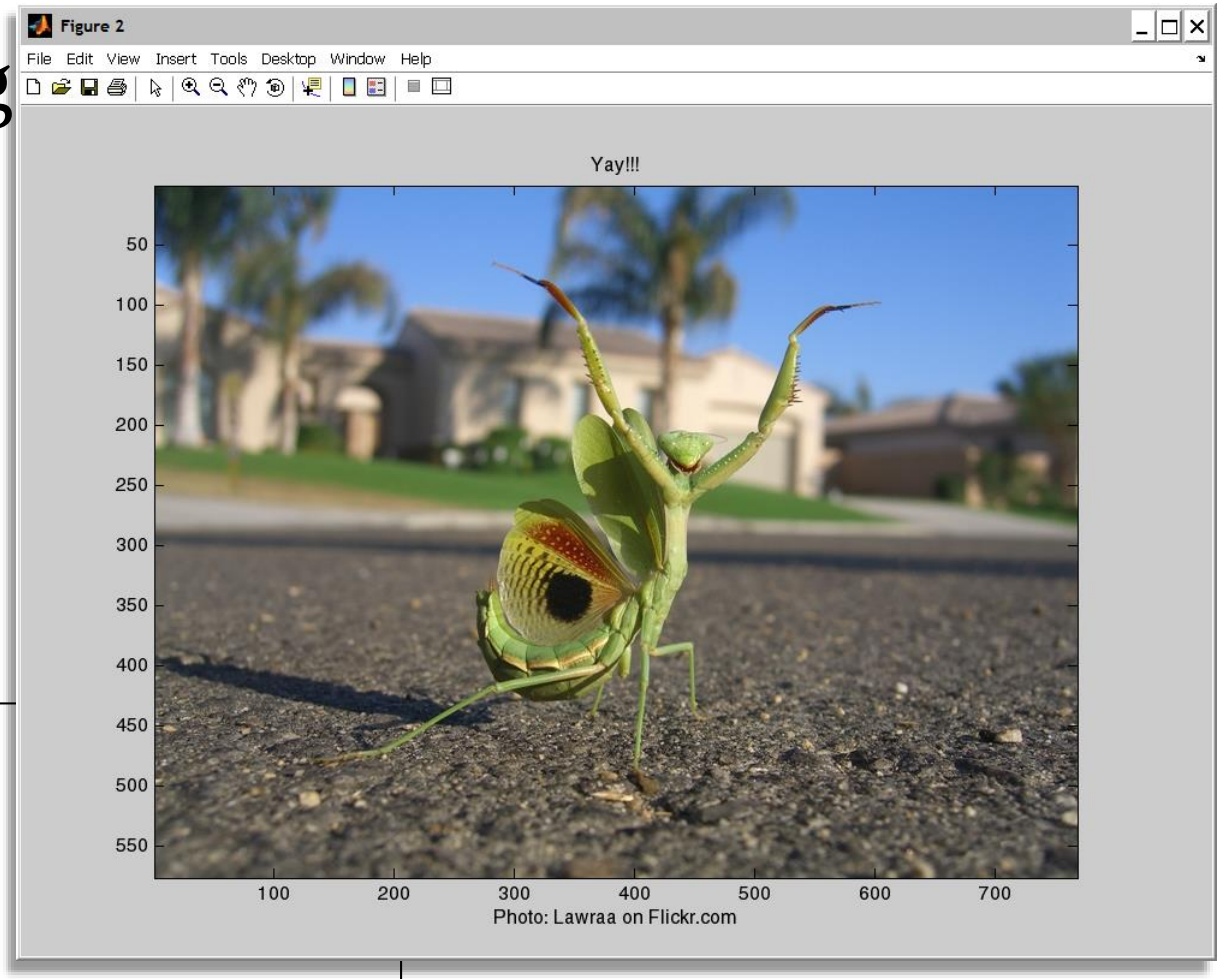
```
>> I = imread('Lawraa - Fl
>> size(I)
ans =
   576   768     3
>> r = randperm(576);
>> c = randperm(768);
>> J = I(r,c,:);
>> figure
>> image(J)
>> truesize
>> title('Scrambled Image'
>> xlabel('What is it?')
```



Fun (if you're an imaging geek) thing to try with Matlab indexing: Scramble an image!

Pixel Indexing

The image can be unscrambled using the row and column permutation vectors, **r** & **c**.




```
>> …
>> xlabel('What is it?')
>> K(r,c,:) = J;
>> figure
>> image(K)
>> truesize
>> title('Yay!!!')
>> xlabel('Photo: Lawraa on Flickr.com')
```

Matlab Vectorization

"tic" starts Matlab's code timer. "toc" stops the timer and reports the elapsed time.

Making a Matlab program run as fast as possible fast by avoiding loops is called "vectorization." It is not something that Matlab does automatically. Rather it is a skill that a good programmer learns.

Looped code

```
tic
for t = 1:1024
    y(t) = sin(2*pi*t/512);
end
toc

Elapsed time is 0.038923 seconds.
```

Vectorized code

```
tic
y = sin(pi*(0:(1/256):2));
toc

Elapsed time is 0.002019 seconds.
```

0.038923 / 0.002019 = 19.2784 ⇒ vectorized code is 20 times faster the looped!

There is a web page on vectorization at the MathWorks:

http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

Matlab Vectorization

Example: compute a distance matrix, the matrix of squared pair-wise distances between two sets of vectors.

$$\mathbf{D}^2(\mathbf{A},\mathbf{B}) = \begin{bmatrix} \|\mathbf{a}_1 - \mathbf{b}_1\|^2 & \cdots & \|\mathbf{a}_1 - \mathbf{b}_N\|^2 \\ \vdots & \ddots & \vdots \\ \|\mathbf{a}_1 - \mathbf{b}_N\|^2 & \cdots & \|\mathbf{a}_N - \mathbf{b}_N\|^2 \end{bmatrix}$$

$$\mathbf{A}_{M\times N} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_N \end{bmatrix},\ \mathbf{B}_{M\times N} = \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_N \end{bmatrix},$$

$$\mathbf{a}_m, \mathbf{b}_n \in \mathbb{R}^M,\ \ a_{m,i}, b_{n,j} \in \mathbb{R},$$

$$\mathbf{a}_m = \begin{bmatrix} a_{m,1} \\ \vdots \\ a_{m,M} \end{bmatrix},\ \mathbf{b}_n = \begin{bmatrix} b_{m,1} \\ \vdots \\ b_{m,M} \end{bmatrix}.$$

$$\begin{aligned} &\mathbf{D}^2_{m,n}(\mathbf{A},\mathbf{B}) \\ &= \|\mathbf{a}_m - \mathbf{b}_n\|^2 \\ &= \sum_{k=1}^{M} \left(a_{m,k} - b_{n,k}\right)^2 \\ &= \left(a_{m,1}^2 + \cdots + a_{m,N}^2\right) \\ &+ \left(b_{m,1}^2 + \cdots + b_{m,N}^2\right) \\ &-2\left(a_{m,1}b_{n,1} + \cdots + a_{m,N}b_{n,N}\right) \\ &= \|\mathbf{a}_m\|^2 + \|\mathbf{b}_m\|^2 - 2\mathbf{a}_m^{\mathrm{T}}\mathbf{b}_n. \end{aligned}$$

Matlab Vectorization

Looped code

```
tic
for r = 1:1000
   for c = (r+1):1000
      diff = A(:,r)-B(:,c);
      D(r,c) = diff'*diff;
   end
end
D = D + D';
toc
```

Elapsed time is 4.893803 seconds.

Example: compute a distance matrix.

`bsxfun(function,A,B)` applies the binary operation specified by `function`, to matrices A and B.

Vectorized code

```
tic
D = bsxfun(@plus,sum(A.^2)',bsxfun(@plus,sum(B.^2),-2*(A'*B)));
toc
```

Elapsed time is 0.009223 seconds.

4.893803 / 0.009223 = 530.609 ⇒ vectorized code is 530 times faster the looped!