CSE 1710

Lecture 8

Working with Images I

To work with images, we need to:

- work with the file system
- work with the operating system's window manager and the platform's graphics hardware
- understand colour models and representation formats
- iterate and construct conditions [next lecture]

About files… pathnames are system dependent

- Windows Local File System (LFS):
  - `C:\USER\DOCS\LETTER.TXT`
- Windows Uniform Naming Convention (UNC)
  - `\\Server\Volume\File`
- Unix-like OS
  - `/home/user/docs/Letter.txt`

Which details are system dependent?

What needs to be abstracted away?

- separator (e.g., `/`, `\`)
- system prefix (e.g., `/`, `\\`, `C:\`)

also **lists** of pathnames are system dependent

- Windows Local File System (LFS):
  - `C:\USER\DOCS\;C:\BIN`
- Unix-like OS
  - `/home/user/docs/:/usr/bin/:/sbin/`

Which details are system dependent?

What needs to be abstracted away?

- path separator (e.g., `;`, `:`)

Useful class: `java.io.File`

- not a utility class; encapsulates `File` objects
  - a *file* in this context can be
    - a directory
    - a "normal file" (i.e., not a directory)
  - files constructed from *pathnames*
- provides static features
  - system-dependent elements
    - separator, path separator
    - demo: `L8App1`

The encapsulation of a `File`…

- provides delegation of file-related tasks:
  - does **this** file exist?
  - is **this** file a directory or a normal file?
  - can I write to **this** file?
  - which files are in **this** directory, if any?
    - assumes this file is a directory
  - make a directory, as specified by **this** file
    - assumes pathname is not already in use and operation is allowed

The encapsulation of a `File`…

- **does not** provide the means to *write* to the file object ☹
  - for this, you need the services of `FileWriter`
  - a `FileWriter` object encapsulates all of the working of writing content to a `File` object
  - defer this aspect for the time being

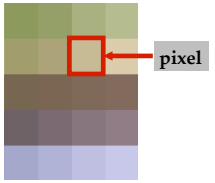
How do I get my hands on a `File` object?

- **construct** one from scratch
  - `L8App2`
- let the user **specify** one for you
  - `L8App3`

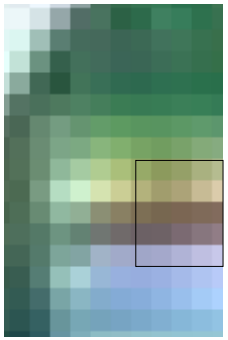
Digital Images

- storage
  - files contain *pixel* and/or *vector* data
    - pixel – a single point at a given coordinate that has specific colour attributes
    - vector data – information about graphic primitives, such as lines, curves, shapes
      - e.g., "draw a circle with radius r with center point at location (x,y) and with a solid black stroke and a solid fill"
- display
  - whatever the file format, the file is *rasterized* to pixels for the graphic display

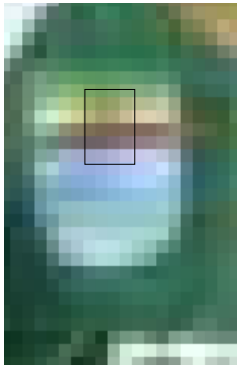
A Pixel Image



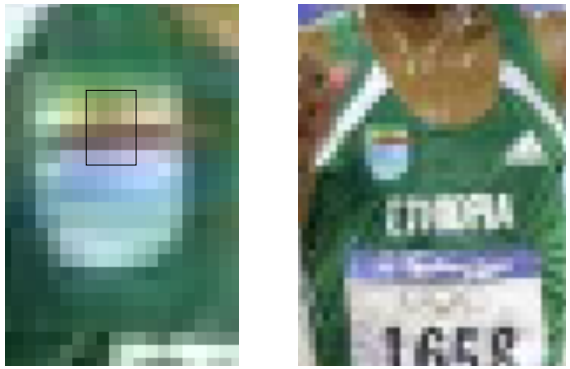
A Pixel Image



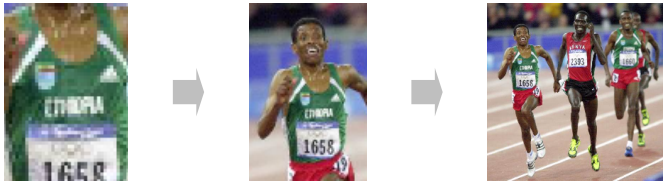
A Pixel Image



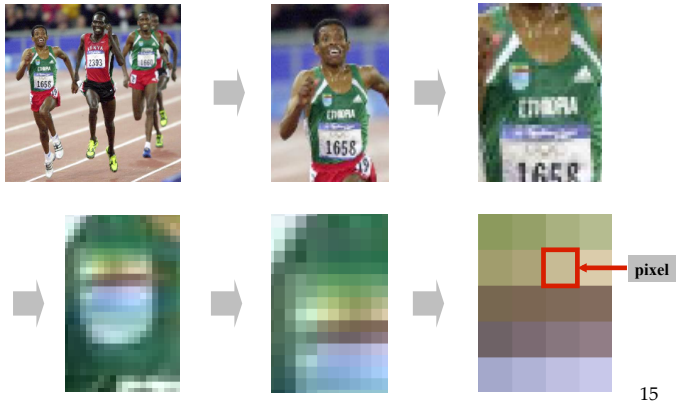
A Pixel Image



A Pixel Image



A Pixel Image



Raster

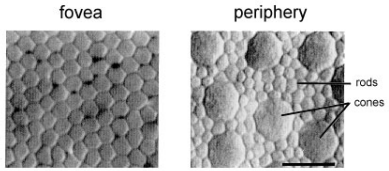
- a rectangular grid of pixels
- each element has a (*x*,*y*) coordinate
  - the convention is that (0,0) is in the upper left hand corner
  - the *x* part of coordinate indicates the column
  - the *y* part of the coordinate indicates the row
  - ***in*** the door and ***down*** the stairs
- `L8App4` – demo of picture explorer

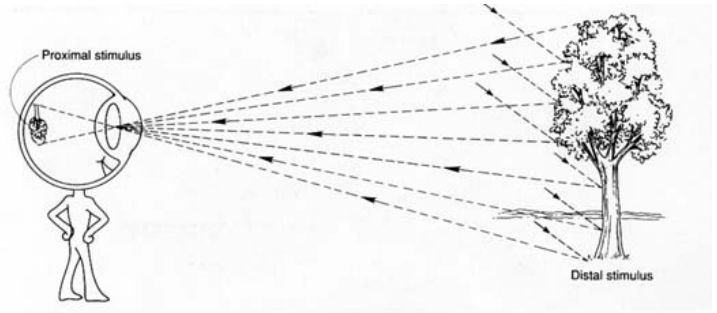
What is the RGB model? Why is it intuitive?

- First, we will discuss the basics of vision…
- the **retina** of the human eye is the location of the photoreceptors
  - *rods*
  - *cones*

Areas of the Retina

- the center, the *fovea*
  - only cone receptors, tightly packed
    - three types of cones: short-, medium-, and long-wavelength
  - no rods
- periphery of retina
  - proportion of rods to cones increase toward edge of retina



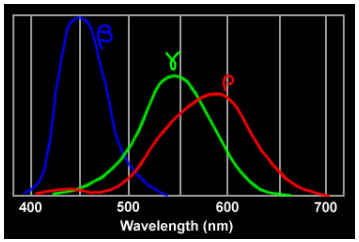


Foveal vision

- fovea has a concentration of three types of cones
- each type is attuned to a different wavelength

Hue

Red –
perceived by long-wavelength cones
Green –
perceived by medium-wavelength cones
Blue –
perceived by short-wavelength cones



Specialized photoreceptors

- peripheral vision
  - contains mostly rods
  - rods are attuned to a broad spectrum of light
    - not specialized to particular wavelengths
    - more sensitive than cones (the threshold is lower)
- fovea
  - specialized for acute detailed vision
- periphery
  - does not provide acuity, but does detect change in scene (e.g., movement)
  - something happened, but not what
  - rods are more sensitive to light than cones

Colour is complicated

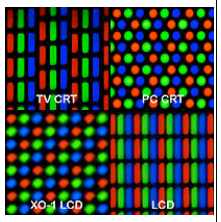
- perception based on 2 types of receptors (hue and intensity)
- our brain does more seeing than our eyes
- what we call colour is more accurately described as hue and brightness

A Key Fact

- the combination of red, blue and green is indistinguishable from **white** to the human eye
- this is exploited by computer displays

Pixels and Subpixels

- Many displays have a cluster of R, G, B sub-pixels for each pixel



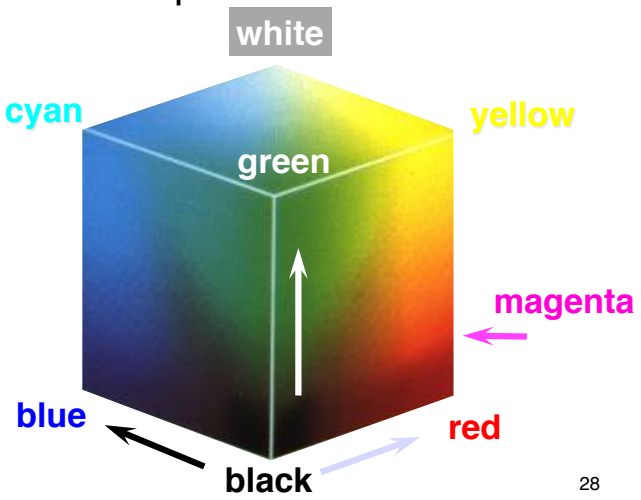
- max *intensity* for R, G, B = seen as white
- min *intensity* for R, G, B = seen as black
- … and other saturated colours…

Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Cyan	0	255	255
Magenta	255	0	255
White	255	255	255
Black	0	0	0

Other cases…

- Intensities are all the same
  - perceived as shade of grey
- Intensities are different
  - perception depends on relative difference between strongest and weakest intensities
- Given a colour, it can be difficult to determine the RGB values without a colour chooser

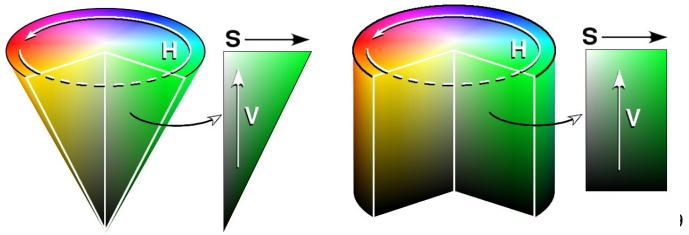
RGB Colour Space



Hue-Saturation-Value (HSV) Model

– Each of hue, saturation, and brightness individually specified

– similarities to the way humans perceive and describe colour



The Picture Explorer

– `L8App4`

What is this *window manager* and why do I care?

– first, a more fundamental question:

- what is the *desktop metaphor*?
  - a set of UI concepts that treat the computer display as if it were the user's real-world desktop
  - desktop items include: documents, folders, desk accessories (calculator, calendar)
  - the purity of metaphor now diluted and now includes things without real-world counterpart
    - » menu bars, task bars, docks, trashcans,

– key feature: desktop items can ***overlap***

What is this *window manager* and why do I care?

– it is system software

- operates computer hardware (the graphics card, in this case)
- provides platform for running apps

– it provides display functionality for apps

- controls placement and appearance of windows
  - open, close, minimize, maximize, move, resize
- implements look and feel of window decorators
  - borders (decorative and functional), titlebar (title and buttons)

The window manager provides services to the VM

- **VM:** *Hi WM, I have this app that wants to draw something graphical on the display…*
- **WM:** *ok VM, here is some screen real estate.*
  - Your app can draw within that region, but not outside it. *(It can try, but I will never permit it to happen)*
  - I will decide what actually gets drawn. *(There may be overlapping windows, so your real estate may be occluded)*
  - I can't guarantee this region. *(The user may move the window, or resize or minimize it)*