Graphics = Light = Color = Memory

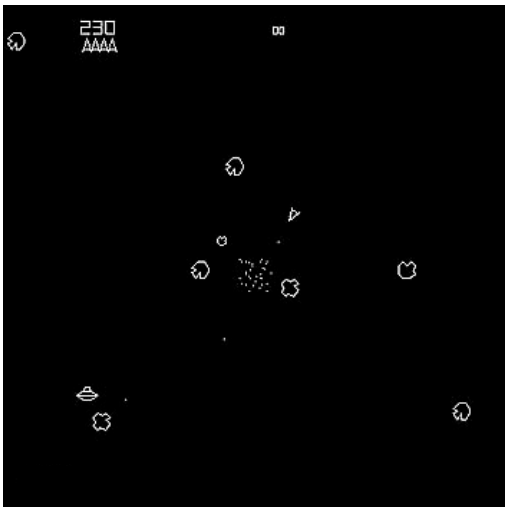
- Light-emitting media
  - *CRT*: cathode ray tube — phosphors excited by electrons
  - *LCD*: liquid crystal display — liquid crystals in a grid; current controls polarization and thus controls what colors can be seen
  - *Projected LCD*: LCD image is magnified before it reaches the viewer
  - *Plasma*: noble gas in between two glass sheets, also affiliated with an electrode grid; current excites the gas to emit light
- Light-emitting media use an additive approach to color: add a color by adding light in that color (i.e., white = light in all colors)

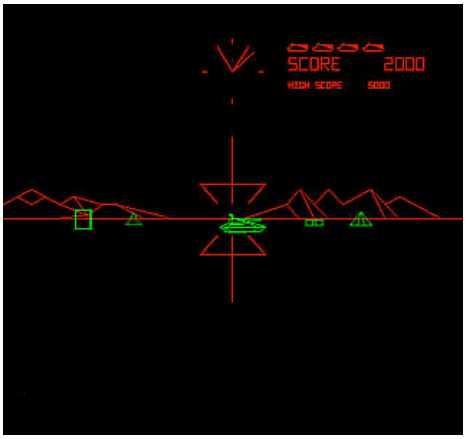
- Light-reflecting media (print)
  - *Ink-jet*: fine spray of pigment onto display medium (most of the time, paper)
  - *Thermal transfer*: heat changes pigment on special paper
  - *Laser*: laser beam "etches" image on a drum coated with toner
- Light-reflecting media use a subtractive approach to color: pigments absorb unwanted colors until only the desired color reflects back (i.e., white = no pigment)

Vector Displays

- *Line-based*: Electron gun traces lines directly from point A to point B
- Based on oscilloscope technology — so pretty much restricted to CRT technology
- Made a lot of sense back when memory was expensive…

  … but what does memory have to do with graphics anyway?





Raster Displays

- *Grid-based*: Display is a two-dimensional raster (array) of individual picture elements (pixels) in memory
  - This memory block goes by many names: *frame buffer*, *graphics memory*, *VRAM*
- Display devices transfer or map this 2D grid onto their specific type of display
- *CRT*: electron gun scans the entire screen horizontally and vertically, exciting the appropriate phosphor that corresponds to its grid location
  - Phosphors fade, so watch out for flicker
- *LCD/projected LCD*: grid of crystals maps to a memory location
- *Plasma*: ditto, but this time the pixels correspond to cells of gas
- Note how LCD and plasma displays are inherently raster-oriented
- So, if pixels are memory, what do they hold?

Mapping Memory to Pixels

- The "value" of a pixel is its color
- The way a pixel represents color determines the amount of memory required by that pixel
- Linear memory is mapped into two dimensions: requires a width and height
  - Different mapping schemes, such as *linear* or *planar*
- *Pixel ratio* is a pixel's aspect ratio — because sometimes pixels aren't square

red	black	green	puce
red	black	blue	green
black	red	lime	tan
cyan	white	white	black

- Example: a 4x4 frame buffer/display
  - Frame buffer size (in units of memory) depends on the size of each pixel
- Non-CRT display hardware devices typically have a *native resolution*, corresponding to the grid size of its physical elements (liquid crystals, cells of gas)

Pixels and Color

The *three-color model* is used to quantify digital light-emitted color: colors are a triple of (*red*, *green*, *blue*)

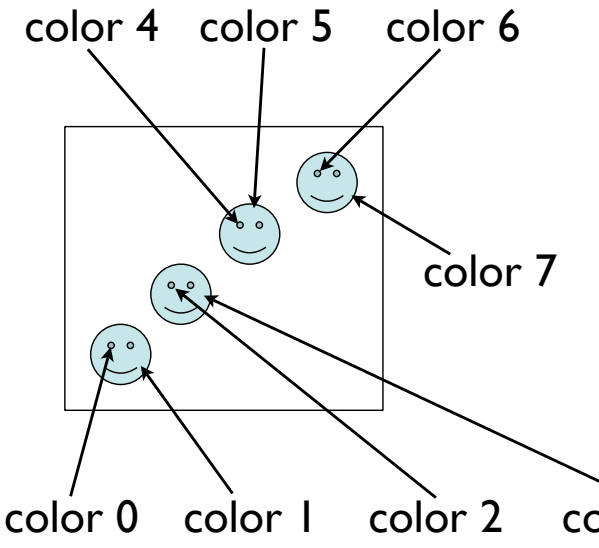
- Light-emitted = additive color, so all colors can be represented by a combination of red, green, and blue
- Individual color values range from "none" to "full blast," such as ranging from 0.0 to 1.0 in floating point, 0 to 255 for 8-bit colors, and so on
- Studies have shown that the human eye can generally perceive no more than 256 levels of a specific hue

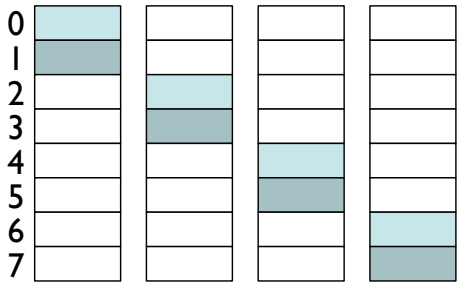
- So if pixels map to some (R, G, B) tuple, how is this tuple stored in memory?
  - *Direct representation*: the pixel is the tuple
    - For 1 byte per color, we need 3 bytes per pixel
    - For monochrome, we need 1 bit per pixel
  - *Indexed representation*: a pixel in memory is an index to a *color lookup table* (a.k.a. LUT or palette)
    - Typically described as "simultaneous *i* out of *n* possible colors."
    - *i* → amount of memory occupied by a pixel
    - *n* → amount of memory occupied by a color
- Catch phrases like "5.0 megapixels" or "128M of graphics memory" ultimately owe their precise technical meaning to how pixels map to memory

Old-School Animation: "Close to the Iron"

- Before full-frame animation became practical, computer graphics animation techniques were very reliant on knowledge of how pixels corresponded to physical memory
- Harder and harder to find real-world examples of these older animation techniques; perhaps the best place at this point would be in emulators of older arcade/computer games

Palette animation: relies on indexed/indirect method of representing computer graphics — image stays the same, and only the palette changes





Rotate among these palettes in turn.

Sample old-school arcade game: Zaxxon

XOR-based animation: Based on the exclusive-or equality ((*a* xor *b*) xor *b*) = *a*

- If you XOR a pixel with another, then XOR-ing that pixel again restores the previous value
- Requires no additional memory to "remember" an animation's background
- Generally works well only with monochrome graphics
- Useful for transient effects like rubberbanding — but these days even that application of XOR animation is fading away
- *Sample old-school arcade game: Berserk*

Sprite animation: Blocks of memory organized into individual animation units called "sprites"

- Copy background to a buffer
- Paint sprite (usually a memory dump with the exception of a designated "background" color)
- To move, paint the background back, then repeat
- Can be used with a single display buffer, or combined with double buffering to reduce flicker
- Basis for a whole generation of video games, such as Arkanoid, the Donkey Kongs, Rastan…the list goes on and on

Basic Image Manipulation

- Since colors are just numbers after all, it stands to reason that manipulating these numbers somehow will result in some recognizable color effects
- Rudimentary image processing is thus a matter of implementing a function from some pixel to another, in some meaningful way
- Simplest form: function that takes a single pixel and produces a new pixel value

- Simple examples:
  - *Filtering* — showing only the red, green, or blue elements of an image
  - *Brightness and contrast* — manipulating all three components in a coordinated fashion
  - *Bit-level effects* — combining two images using bit-oriented operations
- More advanced form: function takes into account a pixel's "neighbors" — the 8 or more pixels surrounding it — to determine its new value
  - Ultimately based on the same principle; just different (more) input