CP SC 4040/6040 Computer Graphics Images

Joshua Levine
levinej@clemson.edu

Lecture 03
File Formats

Aug. 27, 2015

Agenda

- pa01 - Due Tues. 9/8 at 11:59pm
  - More info: http://people.cs.clemson.edu/~levinej/courses/6040
- Reminder: Last Day to drop (without a W) is Tues.

OpenGL / GLUT Structure

```
glutInit();
```

- GLUT Initialization function, must be called before glutMainLoop();

```
glutDisplayFunc(display_func);
```

- Sets the callback function (pointer) to be called in the draw loop.

```
glutKeyboardFunc(keyboard_func);
```

- Sets the keyboard callback — will be triggered if there is a keyboard event. This is also function pointer. Also callbacks for mouse, motion, window resizing, etc..

```
glutMainLoop();
```

- Goes into (infinite) draw loop

Important functions to know for Lab01

```
glRasterPos2i(x,y)
```

- Specifies where to start drawing data.
- Note that bottom left is (0,0), whereas in image data top left is (0,0).
- Should be called in your display function before drawing any pixels.

OpenGL Reference: http://www.glprogramming.com/blue/ch05.html#id40922

Important functions to know for Lab01

```
glDrawPixels(w,h,GL_RGBA,GL_UNSIGNED_BYTE,Pixmap);
```

- Rasterizes `Pixmap` to screen, starting at raster pos and moving up with each scanline.
- `w`, `h` are the width and height of the image.
- `GL_RGBA` specifies the spec of `Pixmap`.
- `GL_UNSIGNED_BYTE` means each channel is a byte.
- `Pixmap` is either an `unsigned int*` of size `w*h` or an unsigned `char*` of size `4*w*h`. Or using the struct we defined in class, with an alpha channel. What is important is how the **memory** is organized.
- Can ignore the "A" channel for now, set it to 255. We'll come back to it.
- Should be called in your display function to do the actual drawing!

OpenGL Reference: http://www.glprogramming.com/blue/ch05.html#id41928

Notes on Bit Packing

```
GL_RGBA + GL_UNSIGNED_BYTE
```

- Expectation is that first four bytes are an R,G,B,A, each using a byte of data.
- Easiest way to do this? A pixel struct with four unsigned char members.
- If you use an unsigned int, you have to pack the data, see House 2.3.2
- HOWEVER, that assumes big endian. Lab machines are little endian. This causes confusion!
- You may need to pack the bits in reverse:

```
(red) | (green << 8) | (blue << 16) | (alpha << 24);
```

- Also, can try using GL_UNSIGNED_INT_8_8_8_8 or GL_UNSIGNED_INT_8_8_8_8_REV instead of GL_UNSIGNED_BYTE.

OpenGL functions for Read/ Write

- OpenGL provides functionality to both write to the display:

```
glDrawPixels(w,h,GL_RGBA,GL_UNSIGNED_BYTE,Pixmap)
```

- And also to read from it:

```
glReadPixels(x,y,w,h,GL_RGBA,GL_UNSIGNED_BYTE,Pixmap)
```

- Where `x,y` is the lower left corner of the block of size `w,h` that you intend to read from memory and store in `Pixmap` (be sure to allocate `Pixmap` first!!)

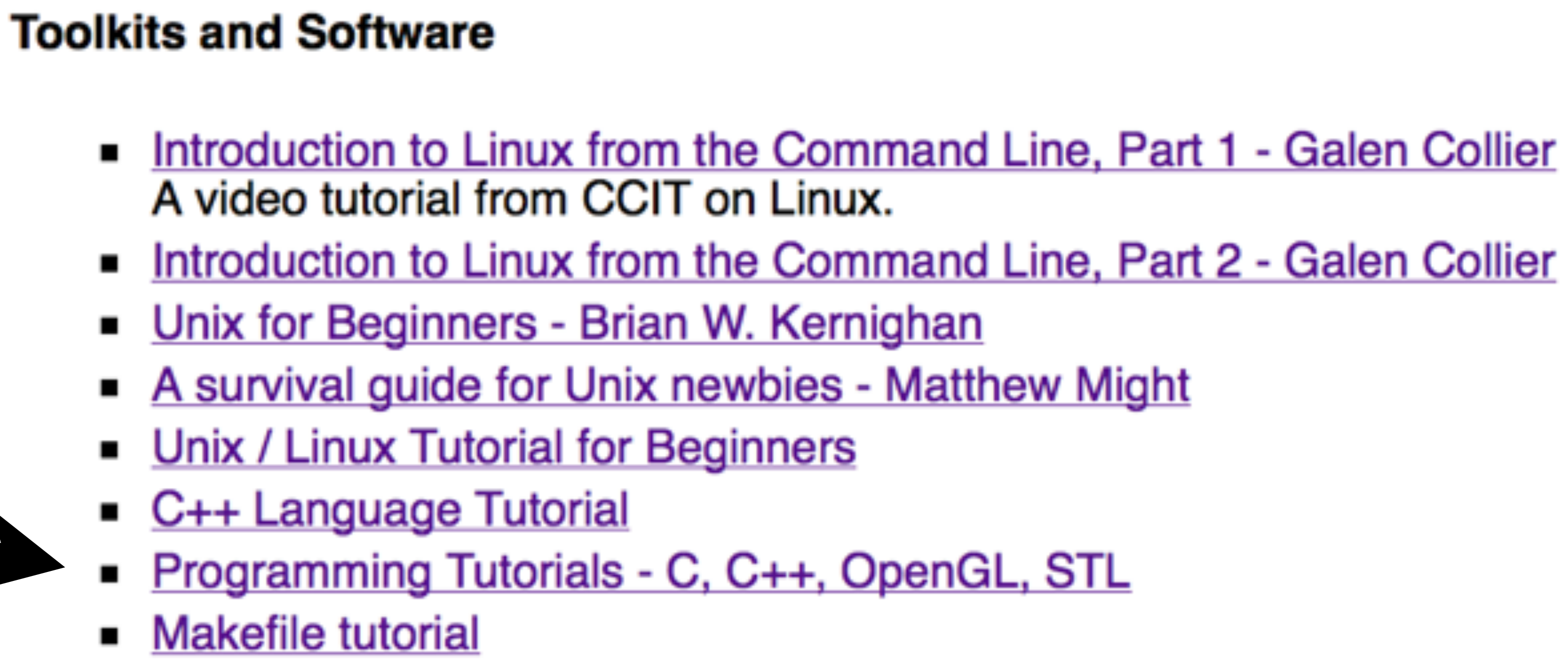
OpenGL Reference: http://www.glprogramming.com/blue/ch05.html

C++ Refreshers

Speaking of allocating memory with new.

Be sure to delete you memory when your program is done!

Check the course page for some references:



Toolkits and Software

- Introduction to Linux from the Command Line, Part 1 - Galen Collier
  A video tutorial from CCIT on Linux.
- Introduction to Linux from the Command Line, Part 2 - Galen Collier
- Unix for Beginners - Brian W. Kernighan
- A survival guide for Unix newbies - Matthew Might
- Unix / Linux Tutorial for Beginners
- C++ Language Tutorial
- Programming Tutorials - C, C++, OpenGL, STL
- Makefile tutorial

e.g., http://www.cprogramming.com/tutorial/dynamic_memory_allocation.html

Command Line Processing (Review Needed?)

See House, section 2.3.5

Image File Formats

Image File Formats

- How many image file formats can you name?
- Why so many? What do they do differently?
- We will not go over all of them in class.

https://familysearch.org/blog/en/solutions-file-format-challenges-2/

Raw File Format

- This type of file just stores directly what is in memory to disk.
- A “raw” dump
- Potential problems?
- Potential benefits??

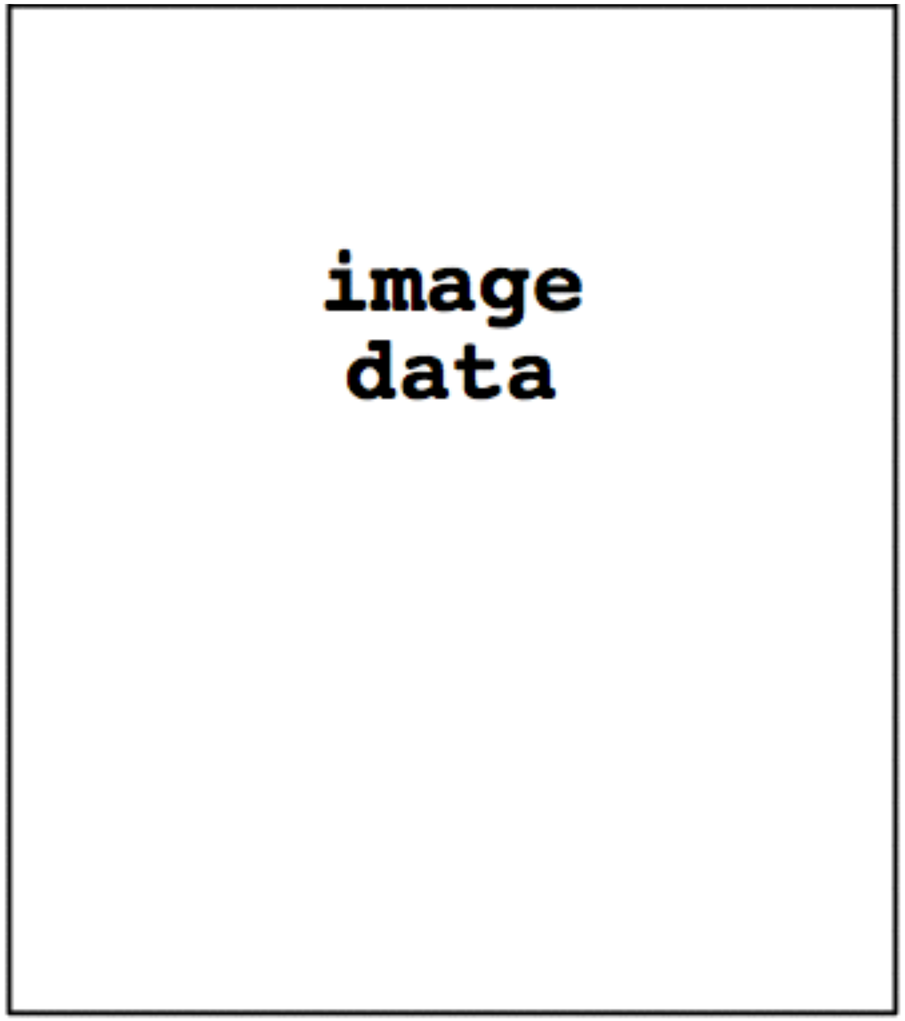
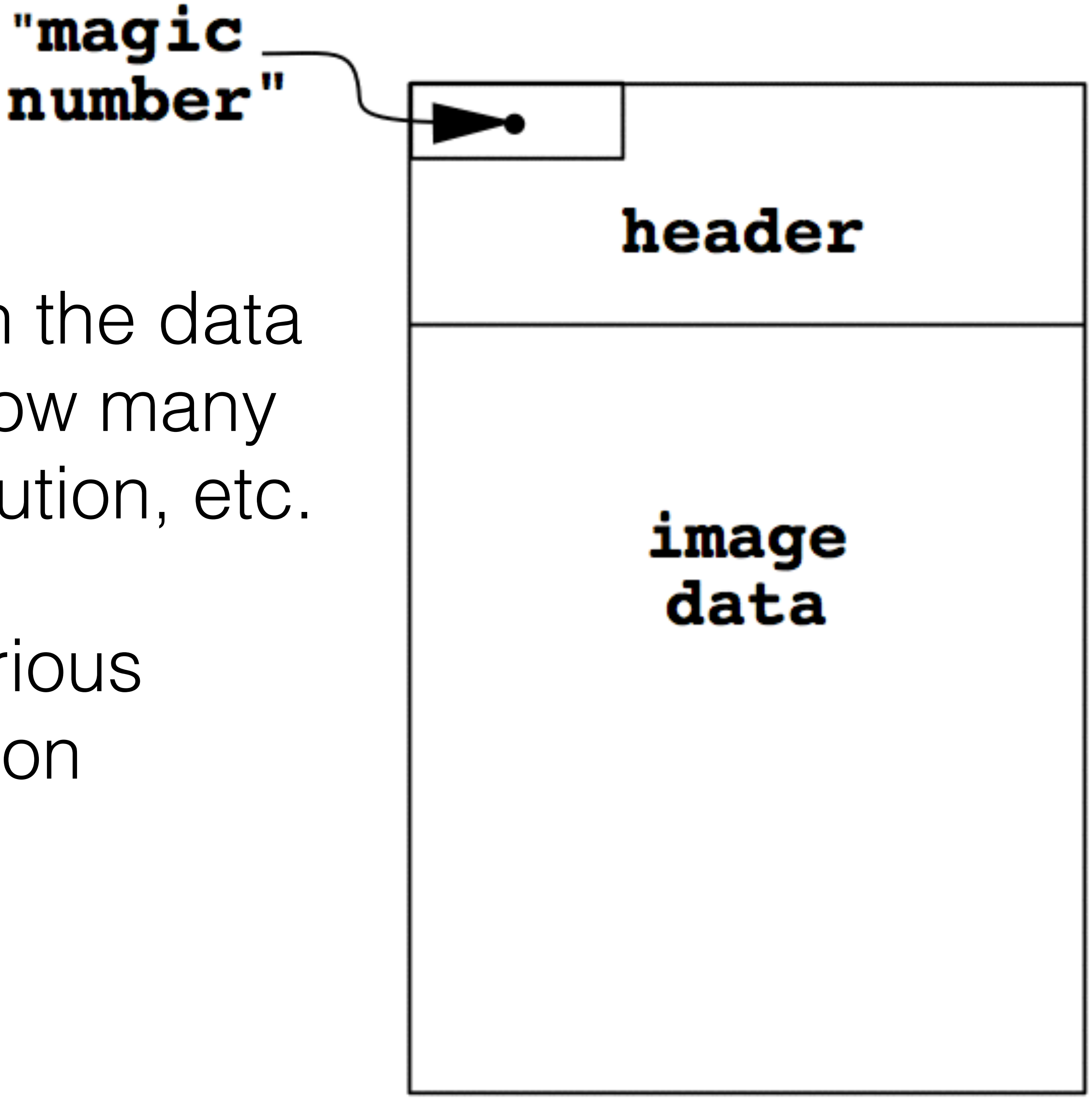


image
data

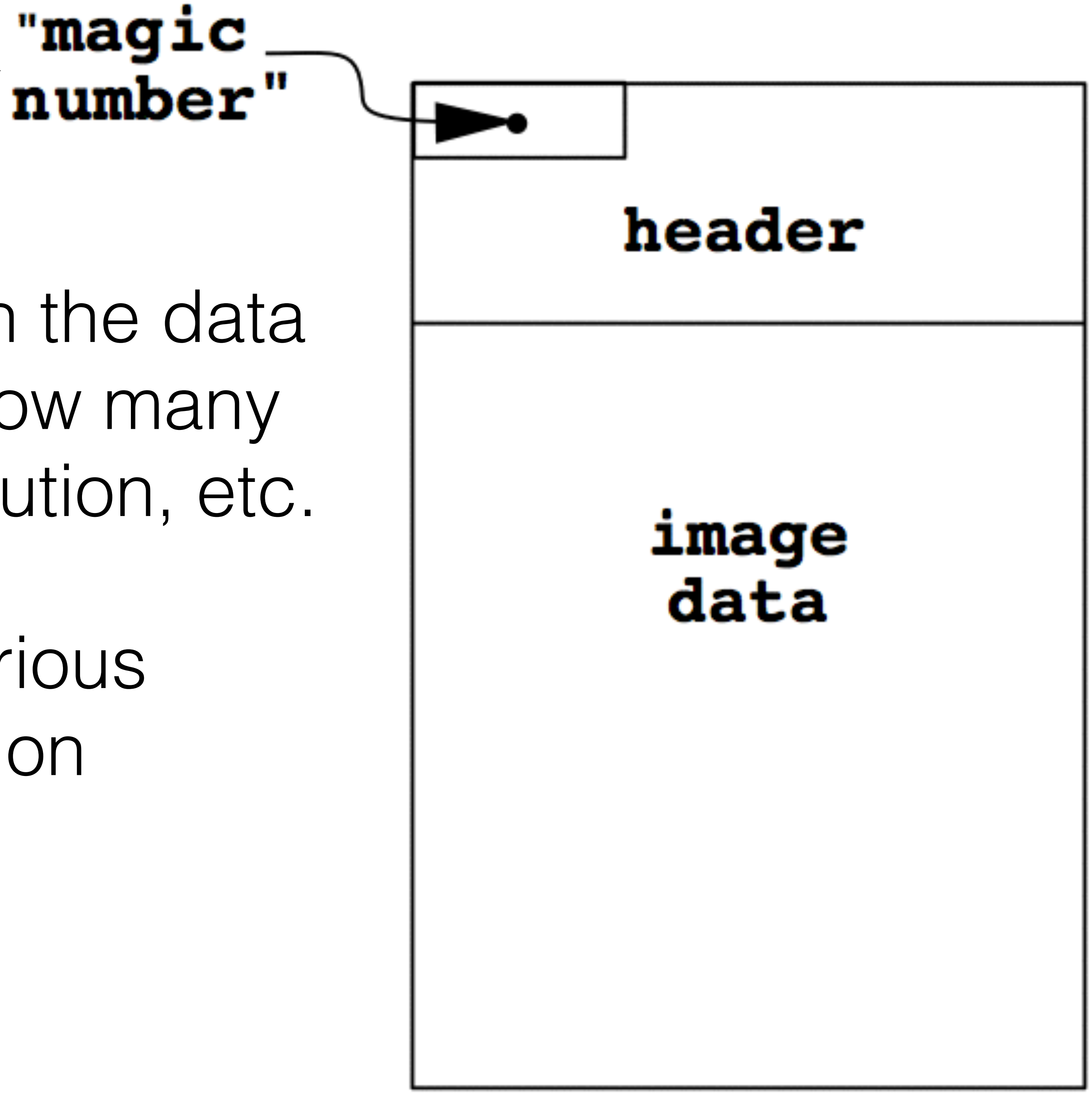
Basic File Format

- Header encodes the mechanism by which the data is stored, how big, how many channels, what resolution, etc.
- Data stored used various schemes, compression techniques, etc.



Basic File Format

AKA “Signature”



- Header encodes the mechanism by which the data is stored, how big, how many channels, what resolution, etc.
- Data stored used various schemes, compression techniques, etc.

PPM Images

Netpbm Project

http://netpbm.sourceforge.net/

- “Portable PixMap Format”
- Designed PPM and its variants to act as an exchange format for images
- Converting between N image formats would require a huge number of converters (how many?)
- Instead, everyone could convert to PPM, as an intermediary, reducing the number to how many?

Netpbm Project

http://netpbm.sourceforge.net/

- “Portable PixMap Format”
- Designed PPM and its variants to act as an exchange format for images
- Converting between N image formats would require a huge number of converters (how many?)
- Instead, everyone could convert to PPM, as an intermediary, reducing the number to how many?

N(N-1)

Netpbm Project

http://netpbm.sourceforge.net/

- “Portable PixMap Format”
- Designed PPM and its variants to act as an exchange format for images
- Converting between N image formats would require a huge number of converters (how many?)

N(N-1)

- Instead, everyone could convert to PPM, as an intermediary, reducing the number to how many?

2N

Netpbm Format

- Simple, but not efficient for most tasks
- Storage is by the simplest means necessary
- Uses the Header/Data format we described
- Six different variants

Netpbm Magic Numbers

- Stored, in ASCII, as the first two bytes of the file.

	ASCII Data	Binary Data
Bitmaps (PBM)	`P1`	`P4`
Greyscale (PGM)	`P2`	`P5`
Pixmaps (PPM)	`P3`	`P6`

PPM Headers

- In ASCII as well, format is:
- P6...[image width]...[image height]...[color depth]
- ... = white space ignored by the reader
    - Note: Except end of line, which is a bit different
- ... = comments
    - Comments are anything that begins with #, and the rest of the line is ignored

PPM Examples

http://en.wikipedia.org/wiki/Netpbm_format#PPM_example

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

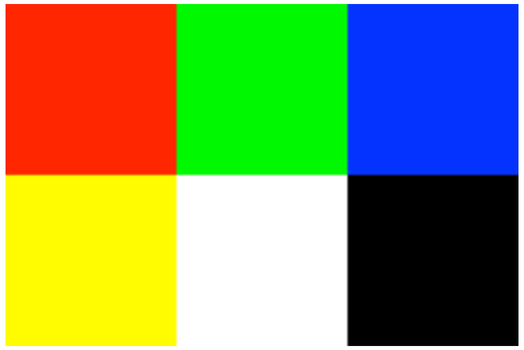
PPM Examples

http://en.wikipedia.org/wiki/Netpbm_format#PPM_example

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

(a tiny image)

PPM Examples

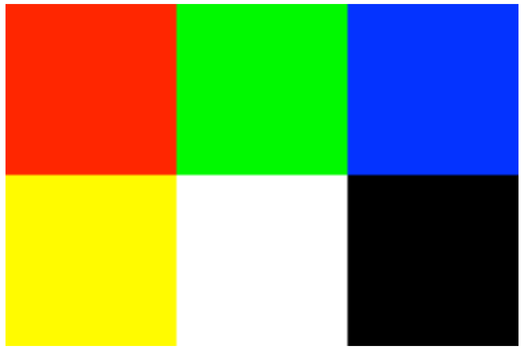


http://en.wikipedia.org/wiki/Netpbm_format#PPM_example

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

(a tiny image)

PPM Examples



http://en.wikipedia.org/wiki/Netpbm_format#PPM_example

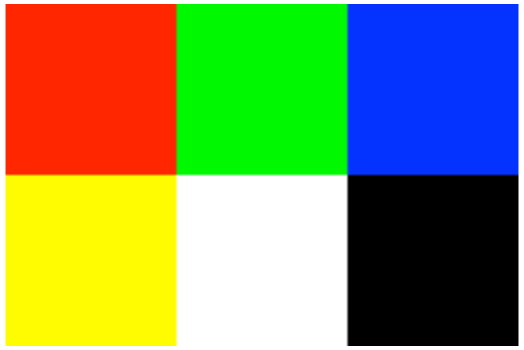
```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

(a tiny image)

```
P6
# The P6 means colors are in binary, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
FF000000FF000000FFFFFF00FFFFFF000000
```

(Hex representation)

PPM Examples



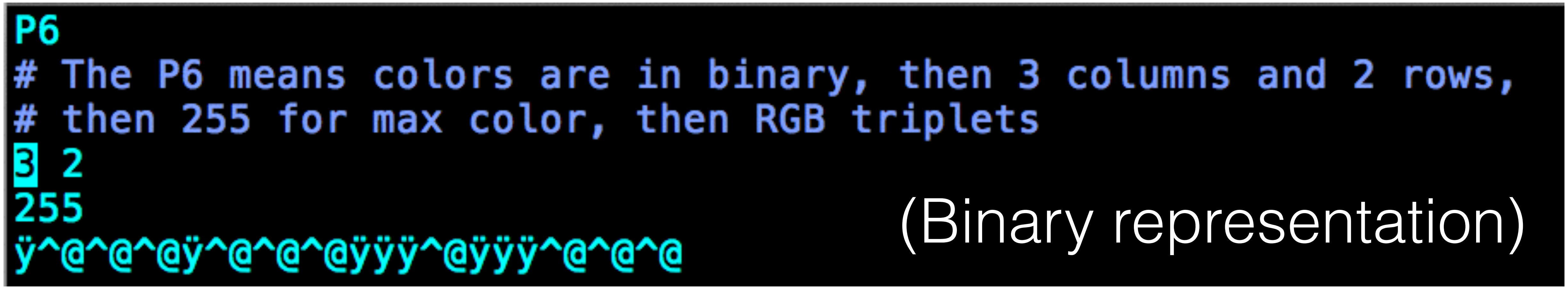
http://en.wikipedia.org/wiki/Netpbm_format#PPM_example

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

(a tiny image)

```
P6
# The P6 means colors are in binary, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
FF000000FF000000FFFFFF00FFFFFF000000
```

(Hex representation)



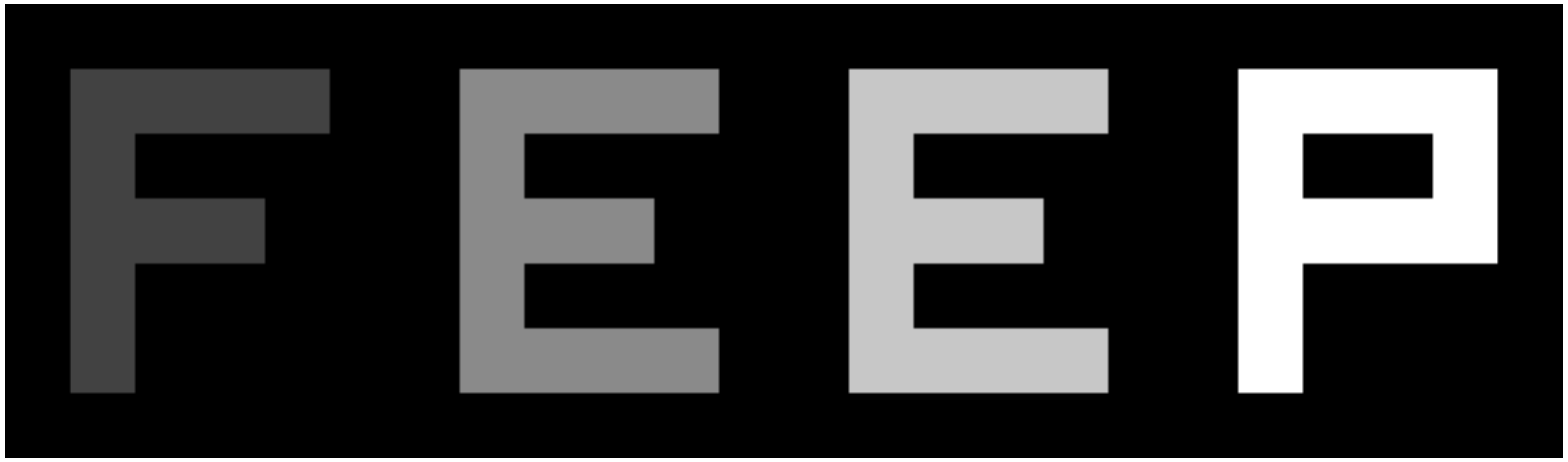
```
P6
# The P6 means colors are in binary, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
ÿ^@^@^@ÿ^@^@^@ÿÿÿ^@ÿÿÿ^@^@^@
```

(Binary representation)

netPBM Data

- The [color depth] controls if one is using 1 byte per channel (255) or 2 bytes (65535)
- Can also be used to scale the max, especially in ASCII formats

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```



RGB Data in PPMs

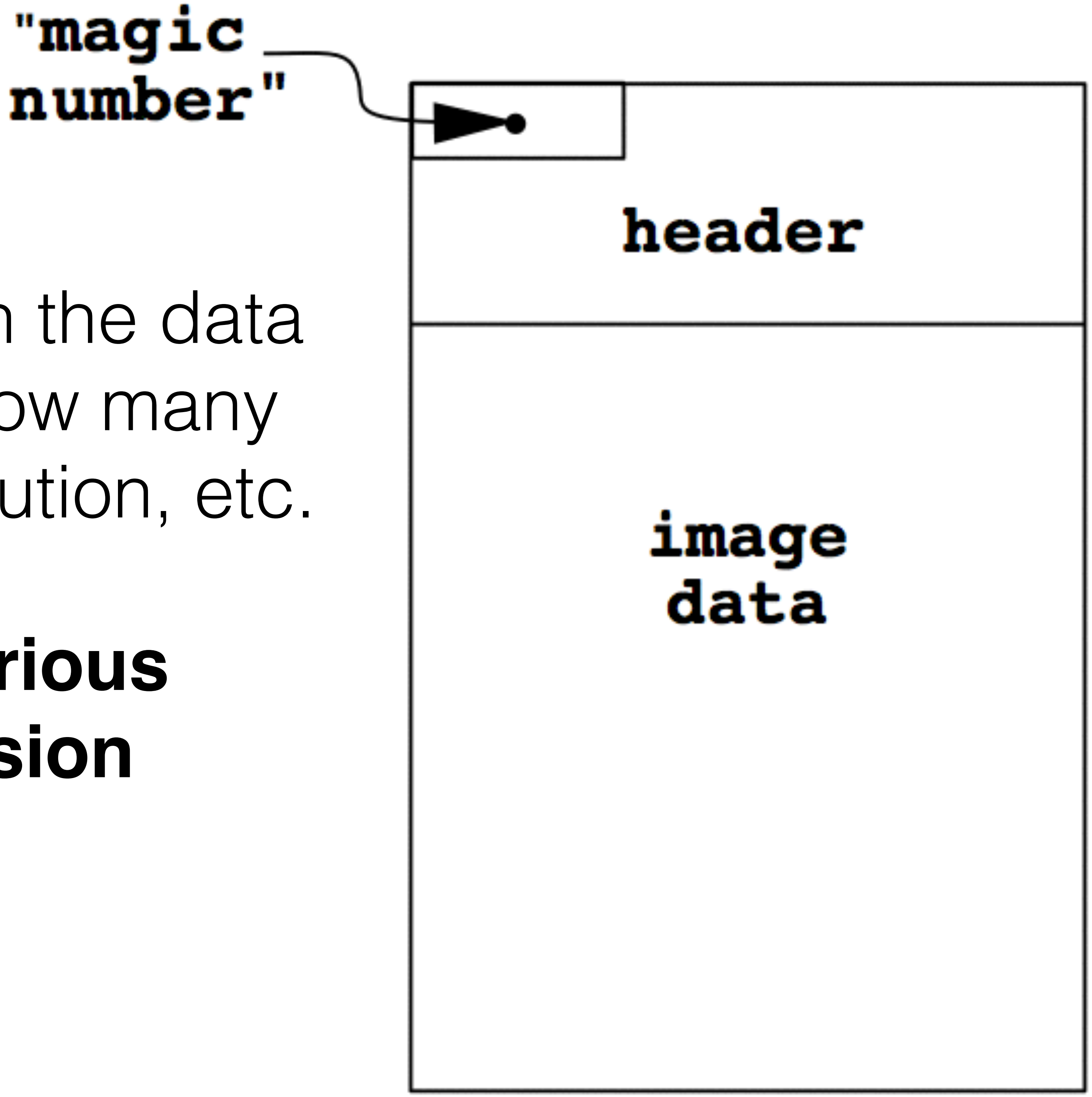
- Note that storage interlaces channels, $R_0G_0B_0R_1G_1B_1$...

- In the binary format that is no separation of scanlines, with ASCII you can add whitespace

Encoding Image Data

Reminder: Basic File Format

- Header encodes the mechanism by which the data is stored, how big, how many channels, what resolution, etc.
- Data stored used **various schemes, compression techniques**, etc.



How much data is in an image?

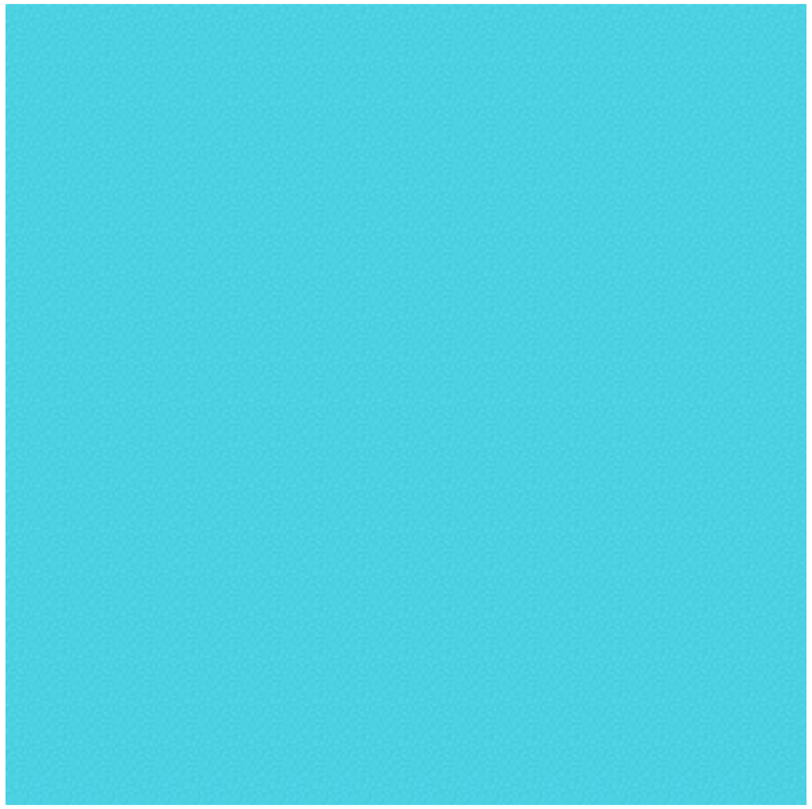
Image Data

- For example, let’s say we have the following:
    - Resolution 400x400
    - 3 channels, RGB
    - Color Depth: 1 byte/channel

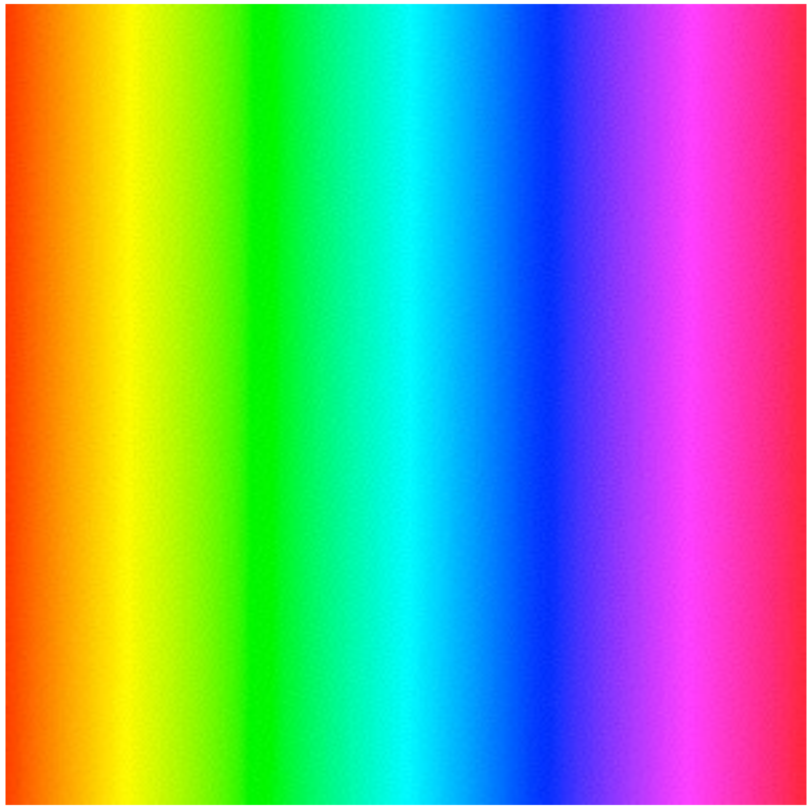
Image Data

- For example, let’s say we have the following:
  - Resolution 400x400
  - 3 channels, RGB
  - Color Depth: 1 byte/channel
- 400 x 400 x 3 channels x 1 byte/channel
  = 480,000 bytes

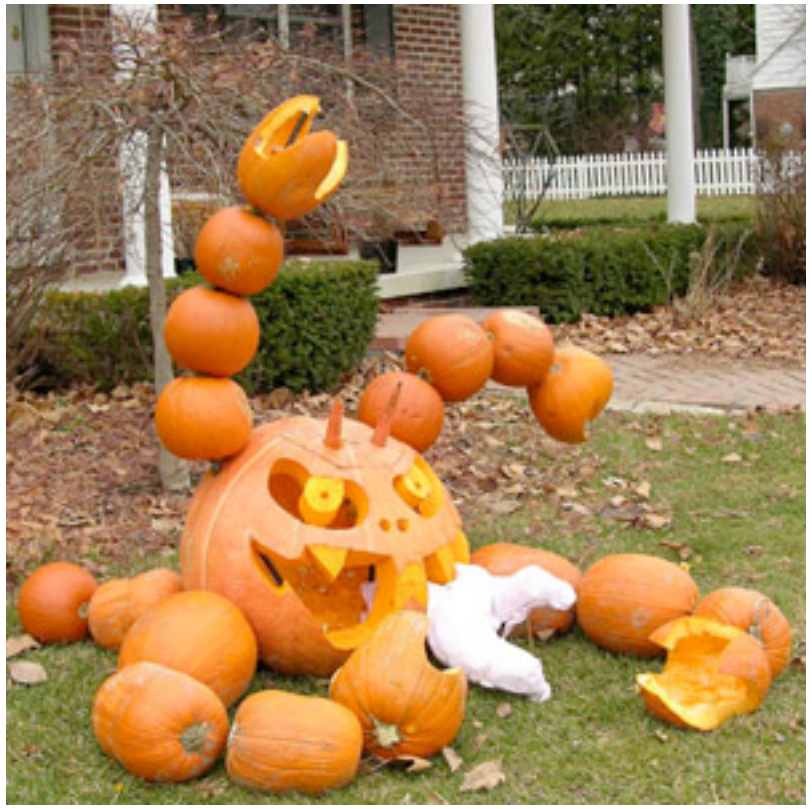
Which image has the most data?



A



B



C

http://www.popularmechanics.com/home/improvement/outdoor-projects/twisted-extreme-halloween-pumpkin-pictures

Two Concepts for Data Encoding

1. **Coherency**: the tendency for one portion of the image to be similar to another.

    - Could be spatial (nearby in (x,y)-space) sequential (nearby in linearized array), temporal (for video)

2. **Redundancy**: the amount of irrelevant or repeated information

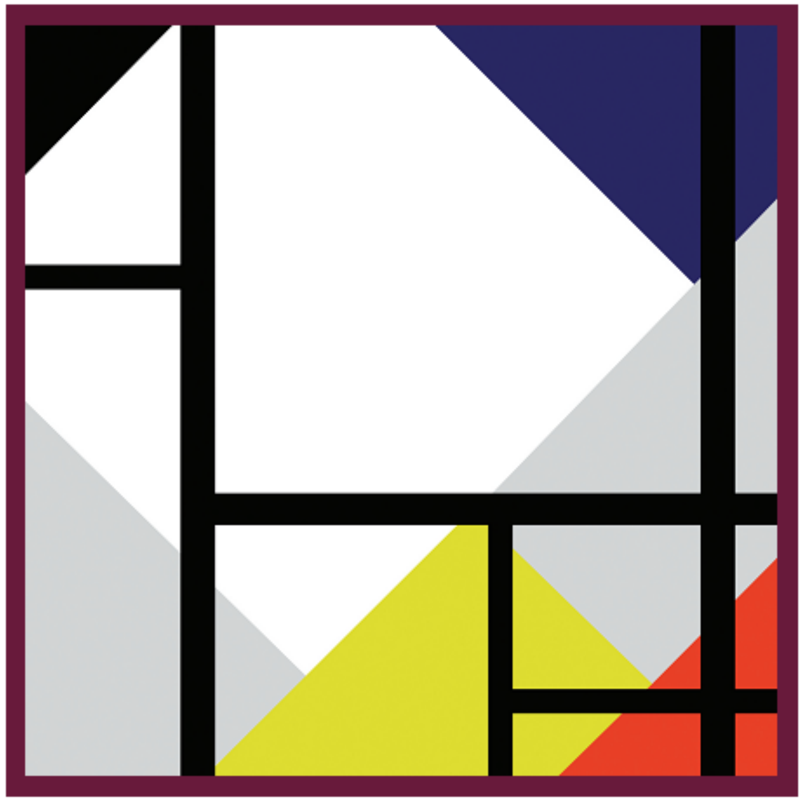
    - Differences that the human eye cannot discern

    - E.g., a far away checkerboard looks grey

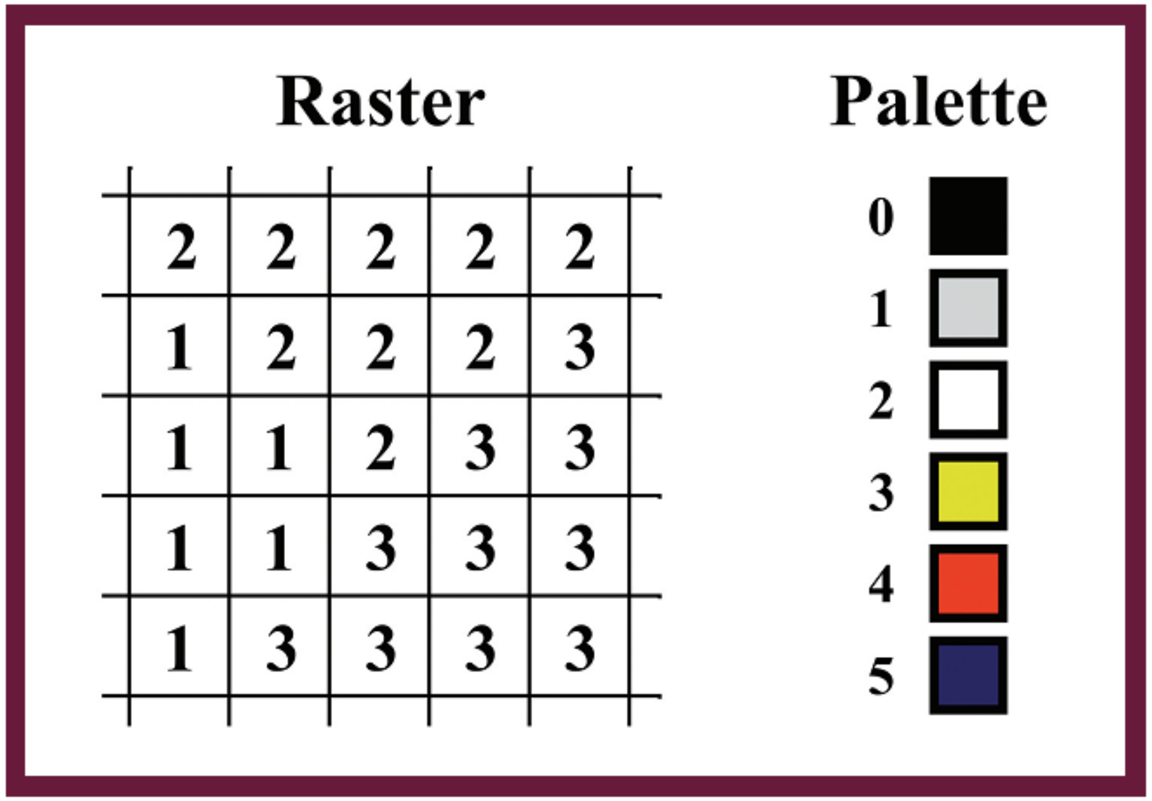
Color Indexing

Color Indexed Images

- One alternative for true-color images having a small number of colors in the image. Use indexing.
  - Each pixel is a single byte.
  - This byte is an index into a color table (or palette).
  - The color is indirectly given by the pixel value.
- How many possible colors? Relationship between color depth and table size.



(a) Source image.



Raster

2	2	2	2	2
1	2	2	2	3
1	1	2	3	3
1	1	3	3	3
1	3	3	3	3

Palette

0	⬛
1	▨
2	⬜
3	🟨
4	🟥
5	🟦

(b) Indexed representation.

Pros/Cons

- Computational efficiency?
  - Is there a penalty for using these? Almost none.
- Small memory footprint?
  - Very efficient use of memory.
  - Requires W x H x color depth, plus storing the table
- Notes
  - The palette limits the number colors, we could **quantize** the color space if we require more though.
  - Color quantization works OK for natural scenes
  - Artificial scenes will often having banding, where you can see discrete transitions between colors instead of smooth blending

“Quick” Color Quantization

- Given a n-bit color, if we want a m-bit color, where m < n
- We can always discard the (m-n) least significant bits
- Example: 8-bit color = 256 colors, say we only want 4-bit color = 16 colors
    - Every color goes from hexadecimal XX to X
    - E.g. **`FF->F; F1->F; A3->A; 4C->4;`** etc.

Run-**L**ength **E**ncoding

(used, in part, for BMP, JPEG, MPEG, etc.)

RLE Images

5	5	5	5	5
5	5	1	2	4
4	3	3	3	4
6	6	2	2	2
2	2	1	7	3

- Goal: exploit *sequential* coherency in the data
- Consider the a 5x5 image of greyscale bytes. Could represent it as a sequence:

```
5 5 5 5 5 5 5 1 2 4 4 3 3 3 4 6 6 2 2 2 2 2 1 7 3
```

- Instead, encode the **runs** of each value and their **length**
- Store each pair as a two-byte (length of run, value)

```
75 11 12 24 33 14 26 52 11 17 13
```

- Original: 25 bytes. RLE: 22 bytes (11 runs, 2 bytes each)

RLE Images

5	5	5	5	5
5	5	1	2	4
4	3	3	3	4
6	6	2	2	2
2	2	1	7	3

- Can also use a flag to distinguish between runs and sequences
- RLE:

```
75 11 12 24 33 14 26 52 11 17 13
```

- RLE + Sequences:

```
75 -212 24 33 14 26 52 -3173
```

- Here -k means a “sequence of length k”
- Original: 25 bytes. RLE: 22 bytes. RLE+S: 19 bytes.
- Note: we’re doing this at the level of colors, you can also RLE on the bit sequence

RLE Pros/Cons

- Best case?
- Reliance on specific depth for length of runs
- Worst case?

Lec04 Required Reading

- Hunt, Ch.2, 3.4.2
- House, 3.1, 3.2