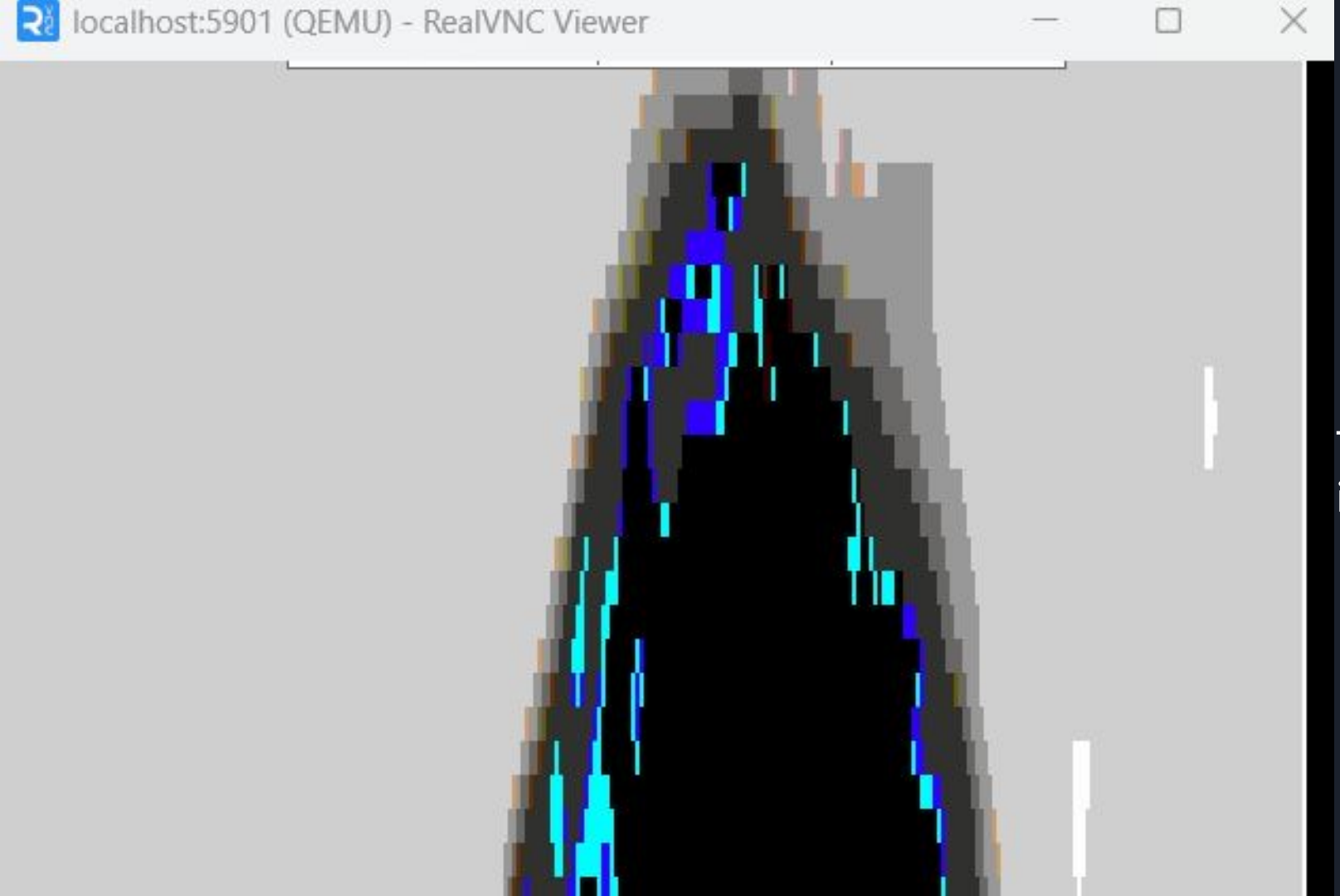
Graphics I/O

- Archit Patil, Brian Zhang, Brian Qu, Rohan Dugad



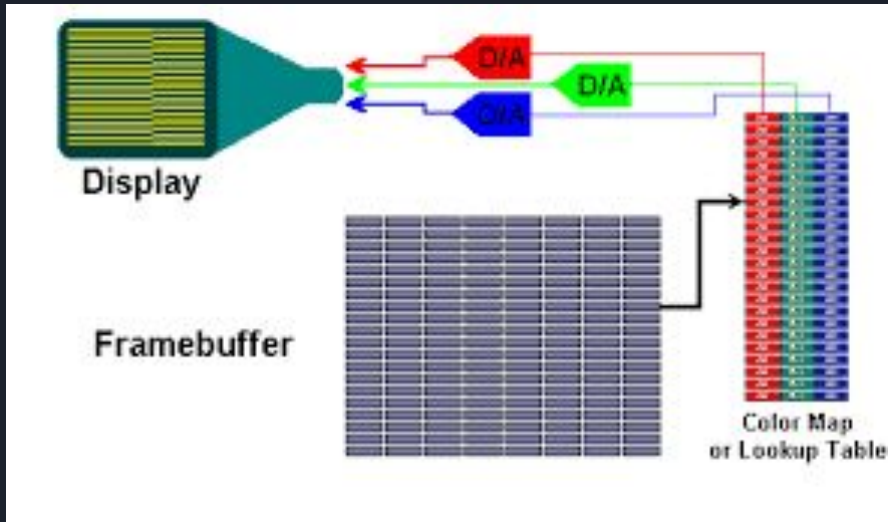
Presenting:
The ultra
instinct lin

Graphics I/O - What we did!

- Framebuffer architecture with double buffering
- Multiple graphics modes
  - Text
  - VGA
  - VBE 8/16/32-bit
- Media viewers: PDF with scrolling, GIF, Video
- Color/Greyscale Implementation + Palette Selection
- Graphics IO-Specific Syscall Implementations
- Performance optimizations & testing

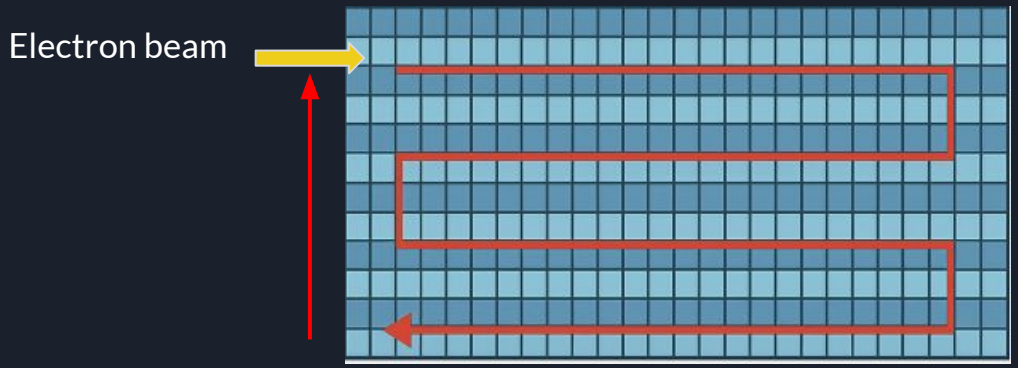
Framebuffer

- Linear memory buffer mapped to display hardware
- Writes to MMIO region -> hardware translates to pixels
- Size determined by: width × height × BPP
- Direct memory access for rendering



How does rendering actually work?

- Electron beam scans horizontally, line by line
- Reads pixel data from framebuffer sequentially
- At frame end: beam returns to top-left (vertical retrace)
- VBlank period: Safe window for framebuffer updates



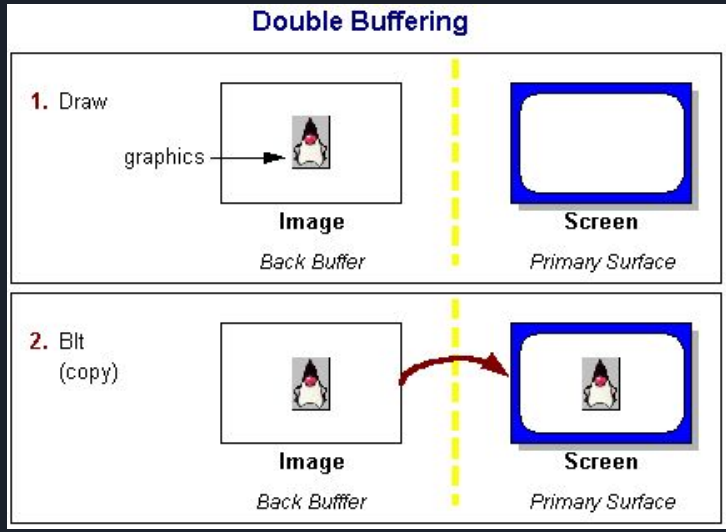
Double Buffering

Single Buffering Problem:

- Direct writes to front buffer during scanout
- Causes tearing

Double Buffering Solution:

- Front buffer: MMIO region
- Back buffer: Regular RAM (we render here)
- Swap during VBlank: pointer swap, not memcpy



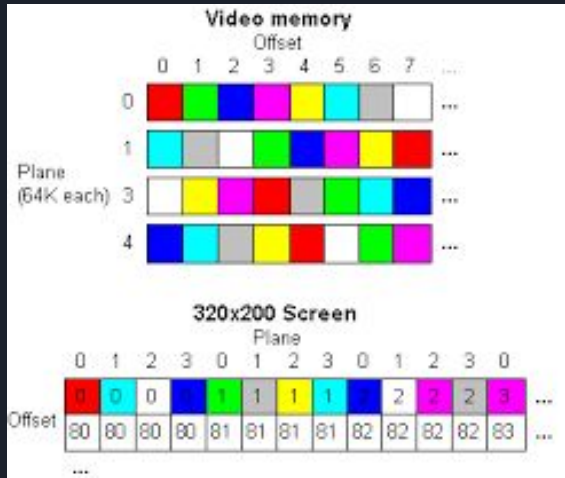
Different Types of Graphics Modes

Mode	Resolution	Color Depth	Use Case
Text	80×25 chars	16 colors	Console/terminal
VGA	320×200	8 bpp (256 colors)	Graphics
VBE	Configurable	8/16/24/32 bpp	Better graphics!

Default VBE: 640×480 @ 32 bpp

VGA Mode

- Resolution: 320×200 pixels
- Color depth: 8 bpp -> 256-color palette
- Framebuffer: 64 KB (320 × 200 × 1 byte)
- Fixed MMIO address: 0xA0000 - 0xB0000 (QEMU)
- Activation: Hardware registers + QEMU flags



Palette

- Color: 24 bit RGB, 8 bits each for R, G, B
- Lossy Conversion from 24 bit to 8/16BPP Mode.
- Max possible colors:
  - 8 BPP:256,
  - 16 bpp = 2^15 = 32,768 (1 byte is unused)
  - 24&32 bpp = 2^32 = 16.7 million

VBE Mode

- Brian Qu

What is VBE?

Why?

Each graphics card had its own ways of accessing higher resolution modes

What?

Standardized access to higher resolution modes in graphics cards

How?

Introduced Firmware in the BIOS

Linear Frame Buffer

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

```
vector<int> screen;
screen[i] = color;
```

Linear Frame Buffer

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

```
vector<int> screen;
screen[i] = color;
```

Width X Row Allocation

Bank Buffer

Bank 0 Bank 1 Bank 2 Bank 3

Advantages:

CPU Memory < Video RAM

Disadvantages:

Bank Switching is Expensive
More Complicated to Use

Bank Buffer

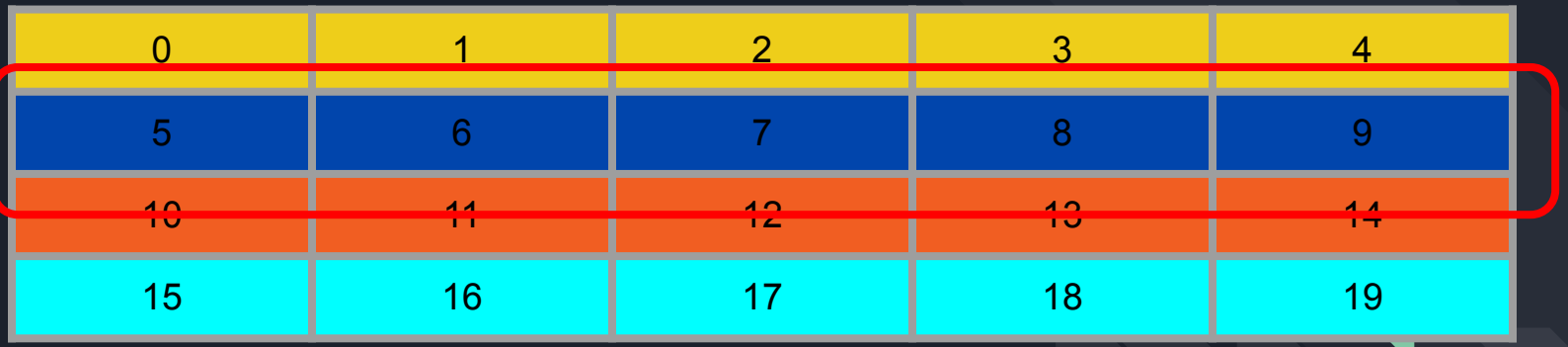
Advantages:

CPU Memory < Video RAM

Disadvantages:

Bank Switching is Expensive
More Complicated to Use

Bank Buffer



Bank 0 Bank 1 Bank 2 Bank 3

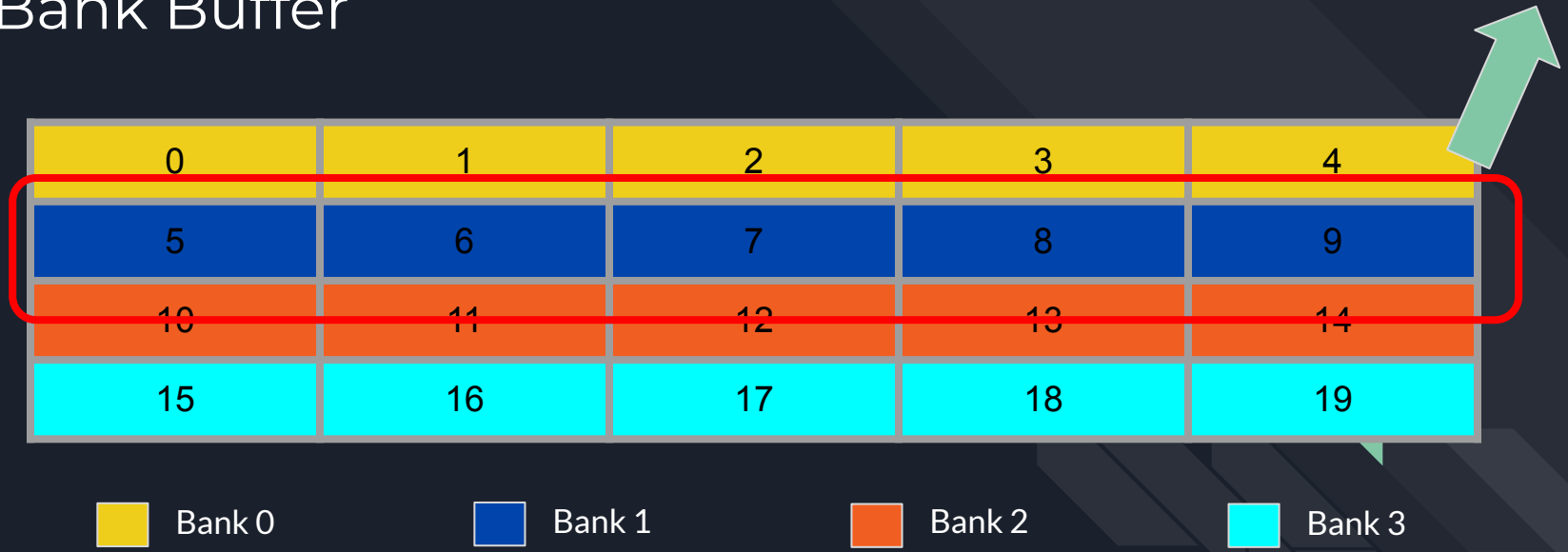
Advantages:

CPU Memory < Video RAM

Disadvantages:

Bank Switching is Expensive
More Complicated to Use

Bank Buffer



Advantages:

CPU Memory < Video RAM

Disadvantages:

Bank Switching is Expensive
More Complicated to Use

Memory Mapped IO Region

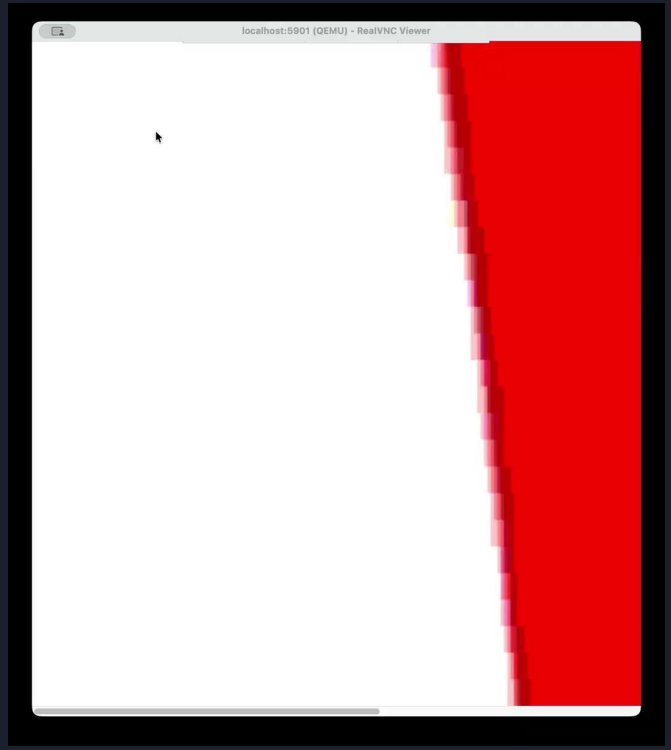
VBE

- Dynamic sized based on:
  - Width
  - Height
  - Bits per Pixel
- Run-time determined address provided by device
  - QEMU: 0xFE000000

VGA

- Fixed Size: 64 Kb
- Address: 0xA0000

VGA
Resolution: 320x200

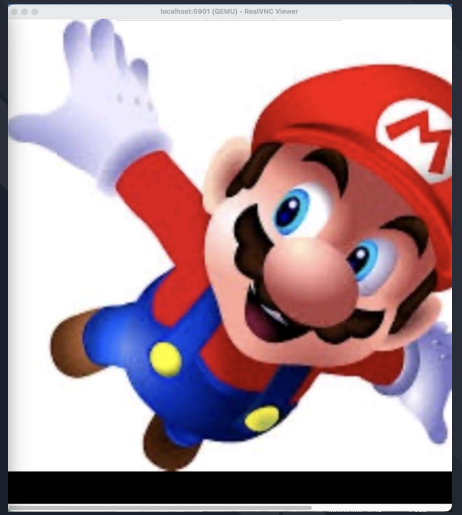


VBE
Resolution: 640x480

Bits Per Pixel: 8
Palette Mode

Pixel = Pre-set Colors

Bits Per Pixel: 16
RGB Mode



Pixel = Red | Green | Blue

Text Mode, Raw, PDF, GIF

- Brian Zhang

What is VGA Text Mode?

- Earliest VGA mode (1978)
- Modeled the screen as an array of 2-byte cells
  - 1 byte **character code**
  - 1 byte **attribute**
  - Typically size 80 x 25
- Lives at 0xB8000
- Not as expressive, but efficient

Attribute								**Character**							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Blink	Bg Color			Fg Color				ASCII							

What is VGA Text Mode?

- Earliest VGA mode (1978)
- Modeled the screen as an array of 2-byte cells
  - 1 byte **character code**
  - 1 byte **attribute**
  - Typically size 80 x 25
- Lives at 0xB8000
- Not as expressive, but efficient

Attribute								**Character**							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Blink	Bg Color			Fg Color				ASCII							

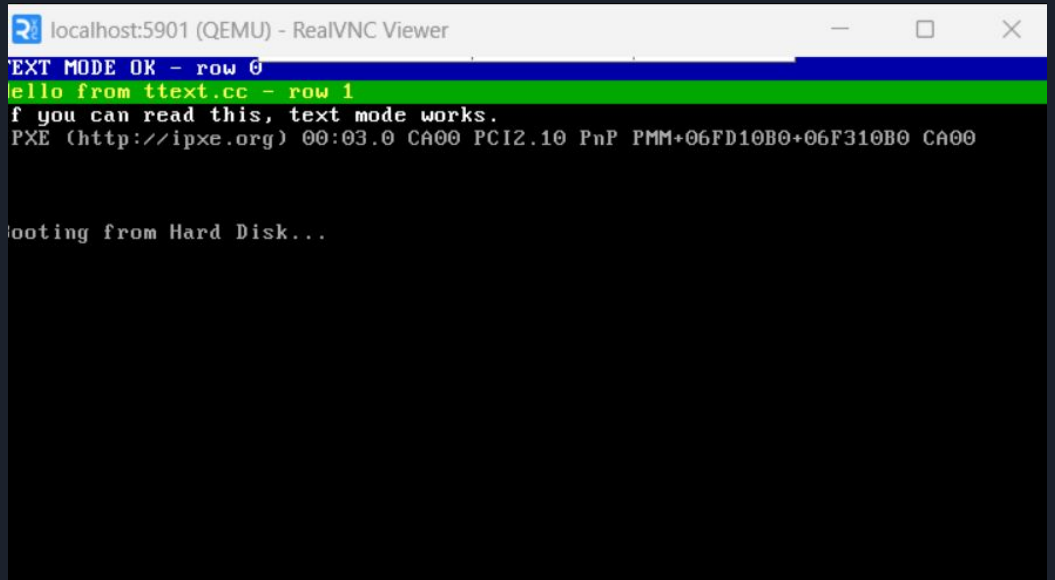
What is VGA Text Mode?

- Earliest VGA mode (1978)
- Modeled the screen as an array of 2-byte cells
  - 1 byte **character code**
  - 1 byte **attribute**
  - Typically size 80 x 25
- Lives at 0xB8000
- Not as expressive, but efficient

Attribute								**Character**							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Blink	Bg Color			Fg Color				ASCII							

Demo

```
write_text_line("TEXT MODE OK - row 0", 0, 0x1F); // blue background, bright white text, row 0
write_text_line("Hello from ttext.cc - row 1", 1, 0x2E); // green background, bright white text, row 1
write_text_line("If you can read this, text mode works.", 2, 0x0F); // black backcgournd, bright white text, row 2
```



From Pixels to Images

256:0:0	256:0:0	256:0:0
0:256:0	0:256:0	0:256:0
0:0:256	0:0:256	0:0:256

216 bits (27 bytes) for 9 Pixels

From Pixels to Images

256:0:0	256:0:0	256:0:0
0:256:0	0:256:0	0:256:0
0:0:256	0:0:256	0:0:256

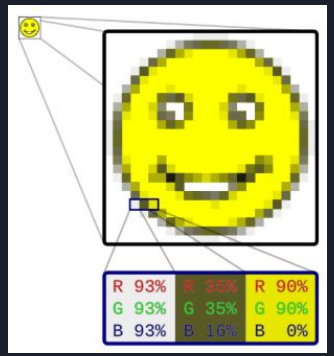
216 bits (27 bytes) for 9 Pixels

NOT SCALABLE (1920 x 1080)

From Pixels to Images (**Raster** Representation)

256:0:0	256:0:0	256:0:0
0:256:0	0:256:0	0:256:0
0:0:256	0:0:256	0:0:256

But we use it anyway:
- **PNG**
- **JPG**
- **GIF**



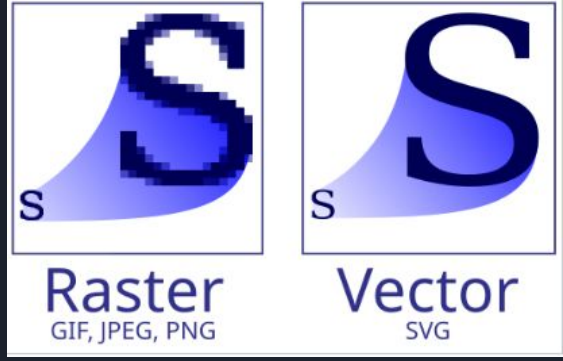
216 bits (27 bytes) for 9 Pixels

NOT SCALABLE (1920 x 1080)

Vector Representation

Draw 256:0:0 Rectangle
Draw 0:256:0 Rectangle
Draw 0:0:256 Rectangle

Vector Representation



Draw 256:0:0 Rectangle
Draw 0:256:0 Rectangle
Draw 0:0:256 Rectangle

Rich representation at the cost of compute

Raw, PDF, PNG, GIF, ...

- Files need to be compressed
- **Lossy** - throw away “less important” detail to save more space
- **Lossless** - no information lost (exact data recoverable)

PNG
Lossless/Raster

JPG
Lossy/Raster

GIF
Lossless/Raster

PDF
Mixed/Vector

PPM (Portable Pixel Map)

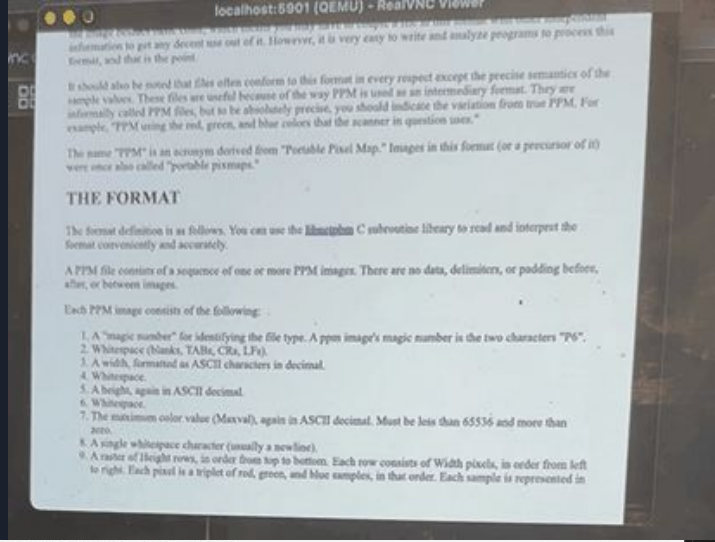
```
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

- The Least Common Denominator Image File Format
- Consists of:
  - Magic Number
  - Width, Height, Max colour value
  - Raw RGB triples in row order

- Images can first be preprocessed (`pdftoppm`)
- Incurs heavy storage cost (1 MB per file)

```
1   # mario.ppm: 490x448, maxval=25
2   255:255:255
3   255:255:255
4   255:255:255
5   255:255:255
6   255:255:255
7   255:253:255
8   255:253:255
9   255:253:255
10  255:253:255
11  253:252:251
12  251:250:248
13  252:252:248
14  254:254:248
15  255:255:246
16  255:255:245
17  254:254:247
18  254:254:250
19  253:253:252
20  253:253:254
21  253:253:255
22  253:253:255
23  250:252:255
```

Demo


localhost:5901 (QEMU) - RealVNC Viewer
information to get any decent use out of it. However, it is very easy to write and analyze programs to process this format, and that is the point.
It should also be noted that files often conform to this format in every respect except the precise semantics of the sample values. These files are useful because of the way PPM is used as an intermediary format. They are informally called PPM files, but to be absolutely precise, you should indicate the variation from true PPM. For example, "PPM using the red, green, and blue colors that the scanner in question uses."
The name "PPM" is an acronym derived from "Portable Pixel Map." Images in this format (or a precursor of it) were once also called "portable pixmaps."
THE FORMAT
The format definition is as follows. You can use the libnetpbm C subroutine library to read and interpret the format conveniently and accurately.
A PPM file consists of a sequence of one or more PPM images. There are no data, delimiters, or padding before, after, or between images.
Each PPM image consists of the following:
1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in


localhost:5901 (QEMU) - RealVNC Viewer

What is VGA Text Mode?

- Earliest VGA mode (1978)
- Modeled the screen as an array of 2-byte cells
  - 1 byte **character code**
  - 1 byte **attribute**
  - Typically size 80 x 25
- Lives at 0xB8000
- Not as expressive, but efficient

Attribute								**Character**							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Blink	Bg Color			Fg Color				ASCII							

Overview

https://netpbm.sourceforge.net/doc/ppm.html

https://giflib.sourceforge.net/whatsinagif/lzw_image_data.html

`pdftoppm`

Raster format vs Vector format

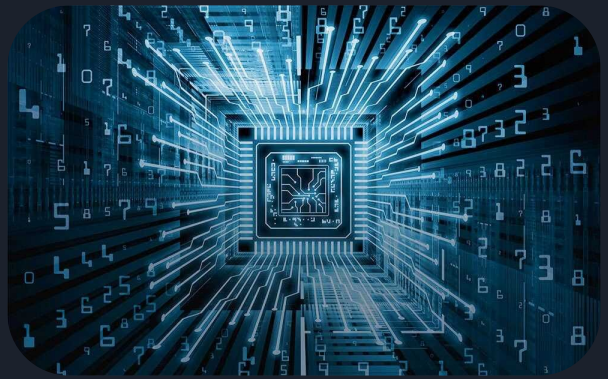
Gifs (What's a gif)

Realtime Graphics, Video Playback

- Rohan Dugad

Why Realtime Graphics & Video Matter?

Usage



Performance

User Experience

Core Requirements

- Maintain framerate
  - 24/30/60 fps → deliver each frame ~16-41 ms
- Keep latency low
  - For interactive or synchronized playback
- Handle resource limits
  - Disk I/O, Display, Memory, CPU bandwidth

Primary Optimizations

- Load in each frame on the fly
  - Discard previous frames
  - Minimize memory burden
- Minimize Disk I/O
  - Frame compression

Custom Video Encoding Algorithms

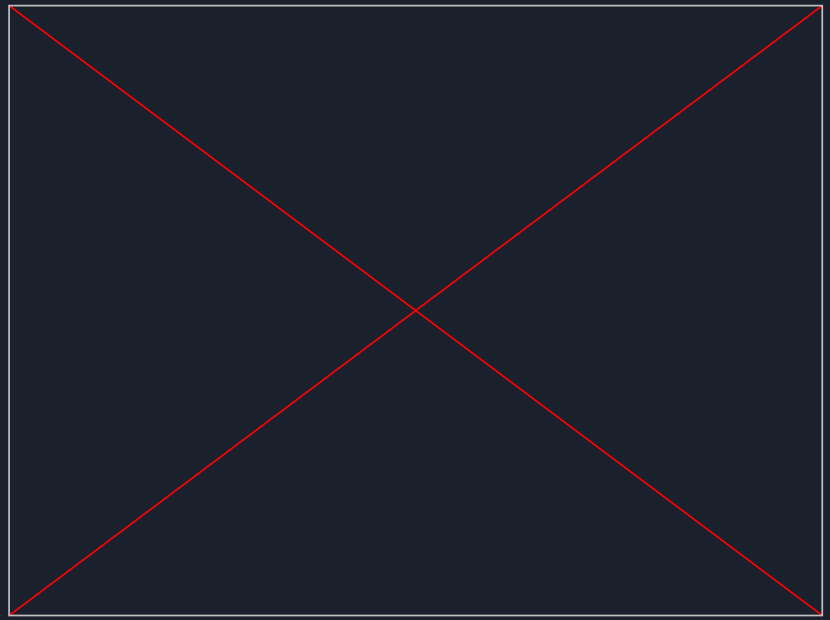
- RAWV
  - Raw Video
- PALV
  - Palettized Video
- CCCV
  - Color Cell Compression Video

Reference Video

220 x 220 RGB at 50 FPS

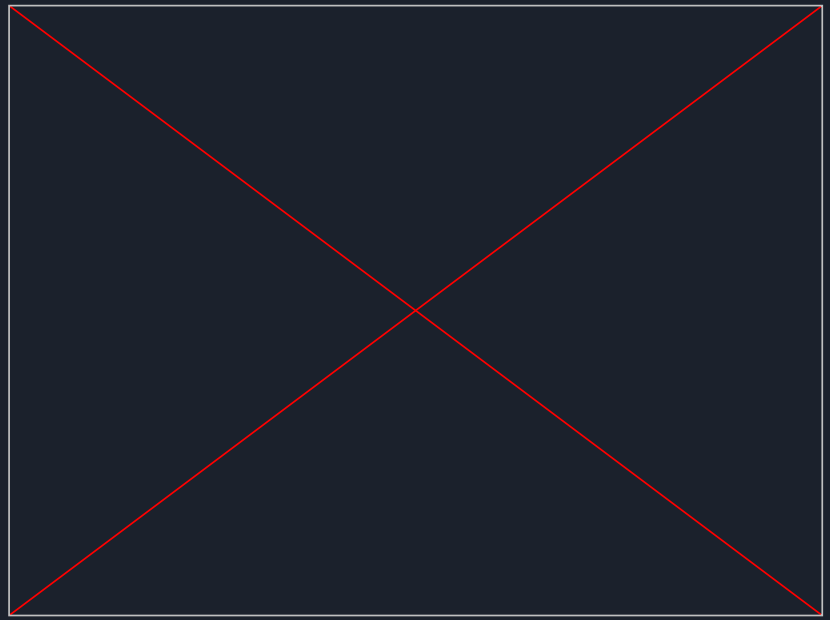
RAWV Encoding

- Store each frame as is
- For RGB 220x220
  - ~145KB per frame
- Zero compression
- ~28MB Video size



PALV Encoding

- Convert each frame to best 256 RGB colors
- For RGB 220x220
  - ~49KB per frame
- 3X compression
- ~9.8MB Video Size



CCCV Encoding

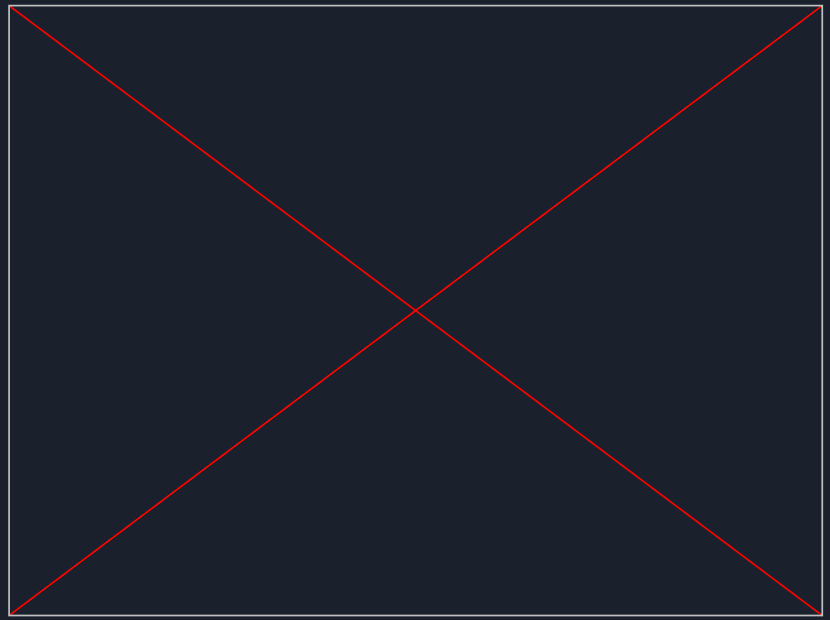
- Convert each frame to best 64 RGB colors
- For 2x2 block
  - Compute mean RGB
  - Mean RGB of above average pixels
  - Mean RGB of below average pixels
  - Correspond two means to two palette colors
  - Hi (6 bits) | Lo (6 bits) | Mask (4 bits)

Mean: 50
Lower Mean: 30
Upper Mean: 70
Lo: 30 →29 (Idx 1)
Hi: 70 → 72 (Idx 7)

20	40
60	80

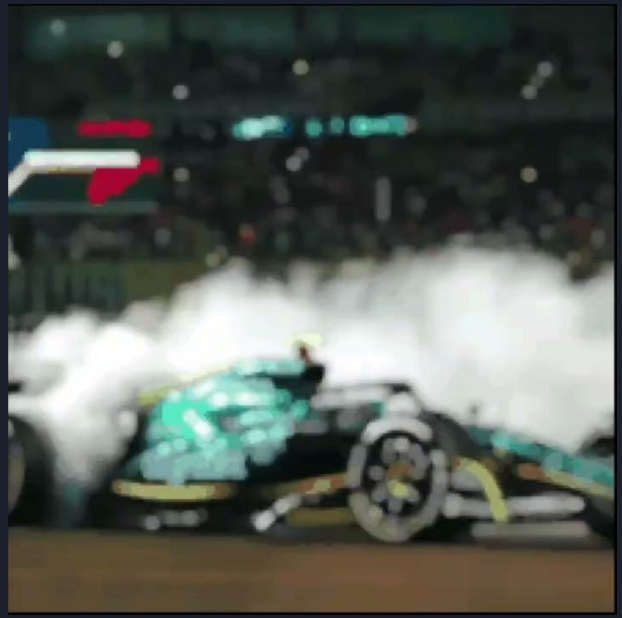
CCCV Encoding

- For RGB 220x220
  - ~24KB per frame
- 6X compression
- ~4.7MB Video Size



Other Optimizations

- Minimize frame writing
  - Render directly on back buffer
- Upsample video frames
  - 320x240 → 640x480
- Dropping frames to keep up
- Buffering frames ahead



Future Work

- Advanced video compression (e.g. MJPEG)
- GPU Support
  - 2D Acceleration
  - 3D Acceleration
- Directly decompressing vector representations

Live Demo

- Johnny
- F1
- Scrollable Pages