8088 Corruption

Motion Video on a 1981 IBM PC with CGA

Introduction

- 8088 Corruption plays video that:
  - Is Full-motion (30fps)
  - Is Full-screen
  - In Color
  - With synchronized audio

…on a 1981 IBM PC with CGA

(and a Sound Blaster for audio)

Introduction – So What?

- 1981 IBM PC w/CGA has:
  - 4.77MHz 16-bit processor
  - 512KB RAM (typically)
  - 16 fixed ugly colors
- Motion video should not be possible given these constraints

8088 Corruption In Action

(demonstration)

History

- Started as a dare
- Collaboration with Sandor Tojzan
- Pilgrimage 2004
  - Won Wild Compo
- Scene Awards 2004
  - Nominated “Most Original Concept”
- 2700+ “diggs”; Diggnation (2006)

The Thought Process

How did you do this?

- Define the problem
  - Write program that displays full-motion video on low-resource hardware (1981 IBM PC)
- Research output device
  - What is technically possible?
- Research input device
  - What looks best?
- Input + output = list of specifications

The Input Device

- Human brain is a pattern recognition engine
- Works better with frequency than amplitude
  - Example:
    16KHz 1-bit speech is intelligible;
    1KHz 16-bit speech is not.
    (even though both take up the same bandwidth)
- Same concept extends to human visual system

Frequency vs. Amplitude

24-bit color	1-bit color
2.5 frames per second	60 frames per second

Both videos use the same bandwidth,
but only one can be considered
“motion-quality” video

What Looks Best?

- Empirical testing
  - Took full-motion video (60 images per second) and created 30, 20, 15, 12, 10, and 6 frames-per-second (FPS) versions of the same video
- Result #1: 30 FPS minimum acceptable motion quality

The Input Device: Audio

- Empirical testing
  - Took source audio at 44KHz sampling rate and created 32, 22, 16, and 11KHz rate versions
- Result #2: 22KHz minimum acceptable quality for music

The Output Device

- CGA displays whatever is stored in its framebuffer (adapter RAM)
- Maximum speed we can update that RAM dictates how fast we can change the display
- Empirical testing
  - Wrote assembly-language routine that measured how fast CPU can copy system RAM to CGA adapter RAM ("**`REP MOVSW`**")
- Result #3: CPU can move 160KB of data to CGA per second

Discovery

- Calculation:
  - Moving data to CGA RAM tops out at 160KB/s
  - 30 FPS minimum quality
  - Audio takes up 22KB/s
  - (160-22) / 30 = 4.6
- Result #4: 4.6KB maximum amount of RAM we can copy each frame to stay within our 30 FPS target

The Output Device

- What are our options?
  - CGA graphics modes use 16KB; 16KB > 4.6KB, so not an option
  - 80x25 text mode uses 4KB; however, 80x25 text mode produces “snow” when writing to adapter RAM (demonstration)
  - 40x25 text mode uses 2KB; no problems writing to adapter RAM
- Final Result: We must use <u>40x25 text mode!</u>

The Converter

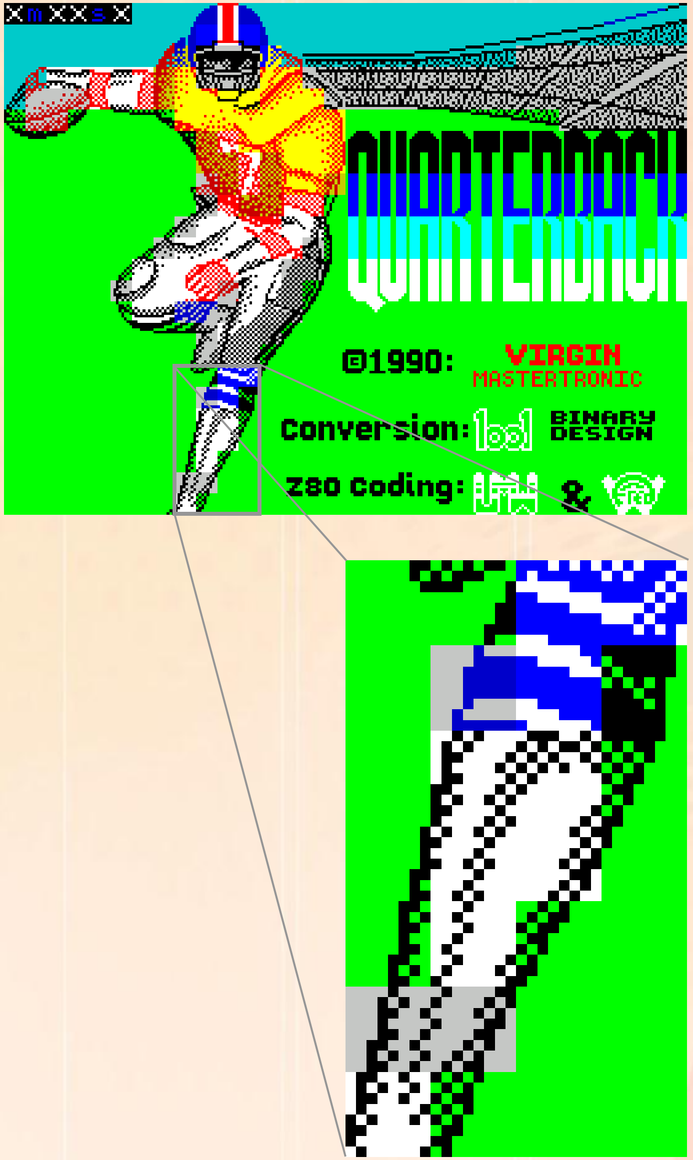
Three iterations:

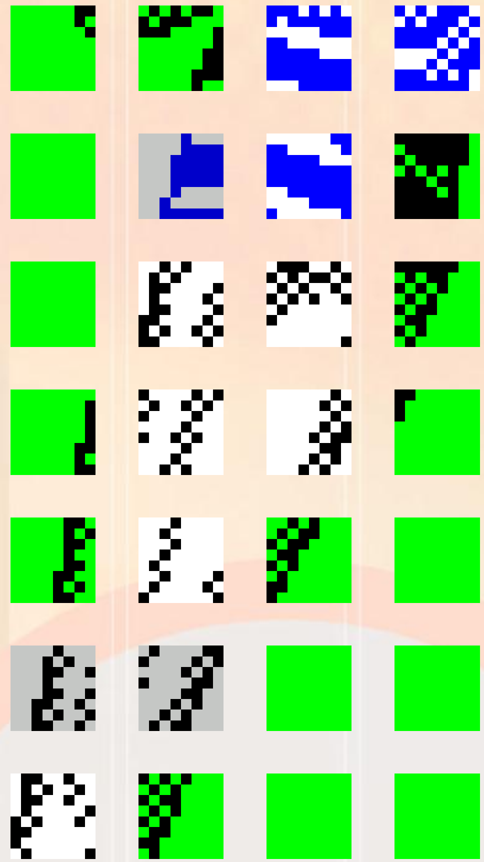
1. Resolution-centric (naïve approach)
2. Color-centric (halftoning)
3. Brute-force resampled compare (final)

1st Converter: Resolution-Centric

- First idea: Emulate “character graphics”
  - 40x25 text mode uses 8x8 character cells
  - Only two colors allowed per 8x8 cell (foreground and background text colors)
  - Effective “graphics” resolution: 320x200
  - Similar to ZX Spectrum graphics, except that each pixel is not individually addressable

Character Graphics Example



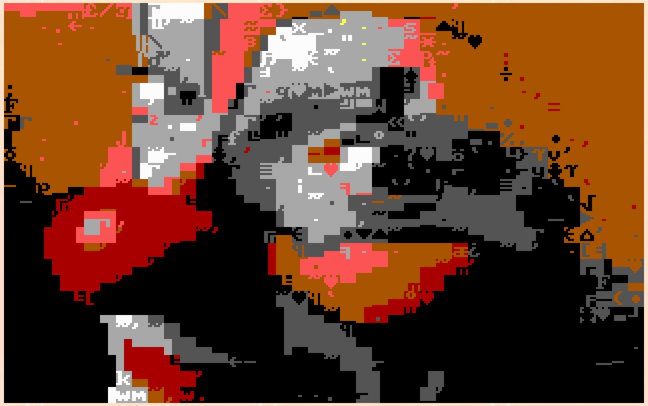


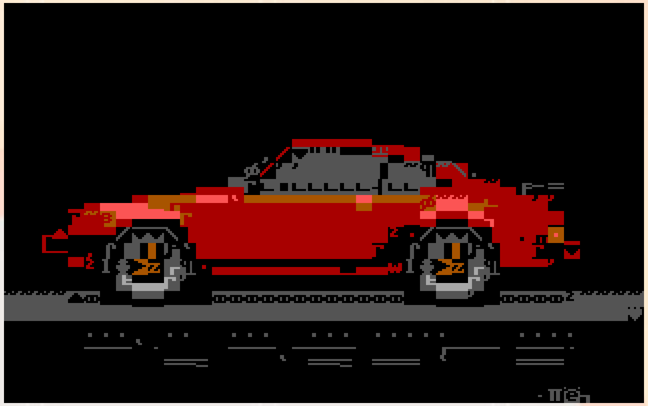
Each 8x8 cell has a foreground and background color, and user-defined font data

1st Converter: Naïve Approach

- 320x200 image broken up into 8x8 “cells”
- For each cell:
  - Remap colors using the CGA 16-color palette
  - Determine two most popular colors
  - Remap cell again using just those two colors
  - Compare to all 512 character/color combinations; best character match used
- IBM character set contains graphics characters – should work, right?

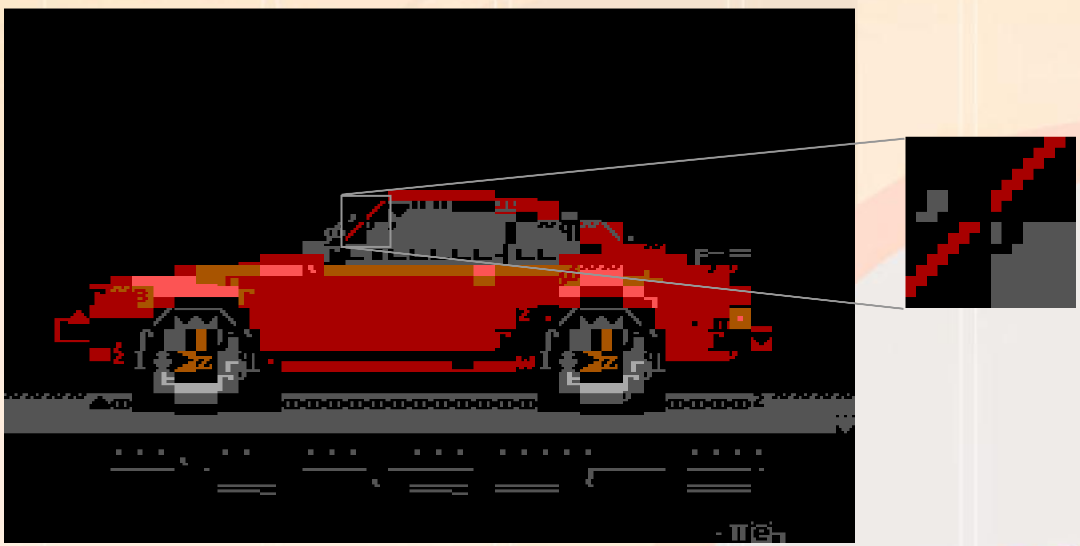
1st Converter Results





1st Converter: Results

- Pros
  - Some details were “perfect” matches
- Cons
  - Largely flat incorrect colors; some picture detail lost

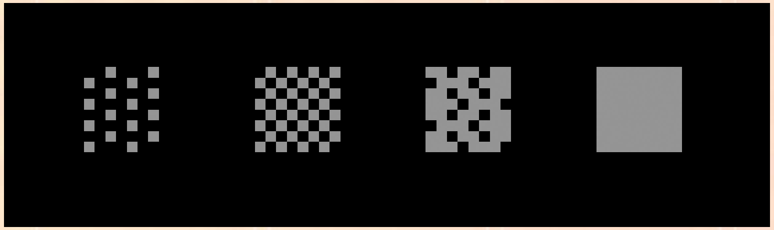


“Perfect” matches to the forward-slash (“/”) character

2nd Converter: Halftoning

- Dithering; trades spatial resolution for color resolution
- IBM character set includes a 50% pattern character, #177 (looks like "checkerboard")
- 50% pattern a crude form of dithering
- 136 unique "colors" possible by mixing colors and using #177

Halftoning in CGA



#176 #177 #178 #219

CGA text font data contains a few shaded graphical characters; #177, a 50% pattern, is what we want

By using all 16 CGA colors (and removing duplicate combinations), we can simulate up to 136 different colors using #177

2nd Converter: Halftoning

- 320x200 picture resampled to 40x25
- For each pixel:
  - Compare to all 136 “color” combinations
  - Use the closest match

2nd Converter Results

2nd Converter: Results

- Pros
  - Colors much better
  - Less memory requirements
    (text character is always #177, so only color data needs to be stored)
  - Conversion process very fast (136-entry lookup table)
- Cons
  - Most detail lost

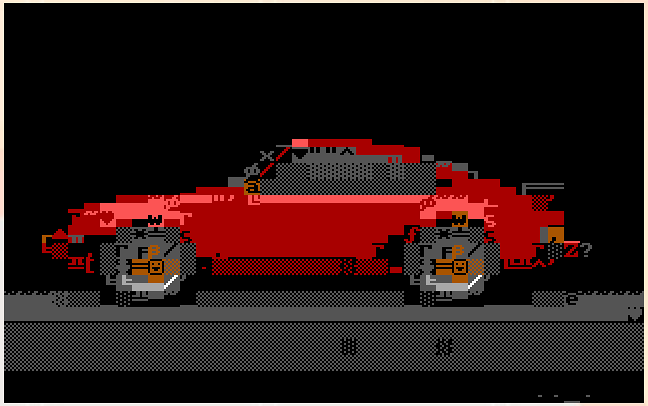
3^{rd} Converter: Resampled Compares

- How to get results that are “halfway” between the first two attempts?
- “Half” led to the idea of resampling both the picture and the character/color combinations smaller and performing comparisons at that level

3rd Converter: Resampled Compares

- 320x200 picture resampled to 160x100 (half vertical/horizontal)
- Divided up into 4x4 “cells”
- Compare each “cell” against every character/color combination *also resampled half vertical/horizontal*
- Use the closest match

3rd Converter Results

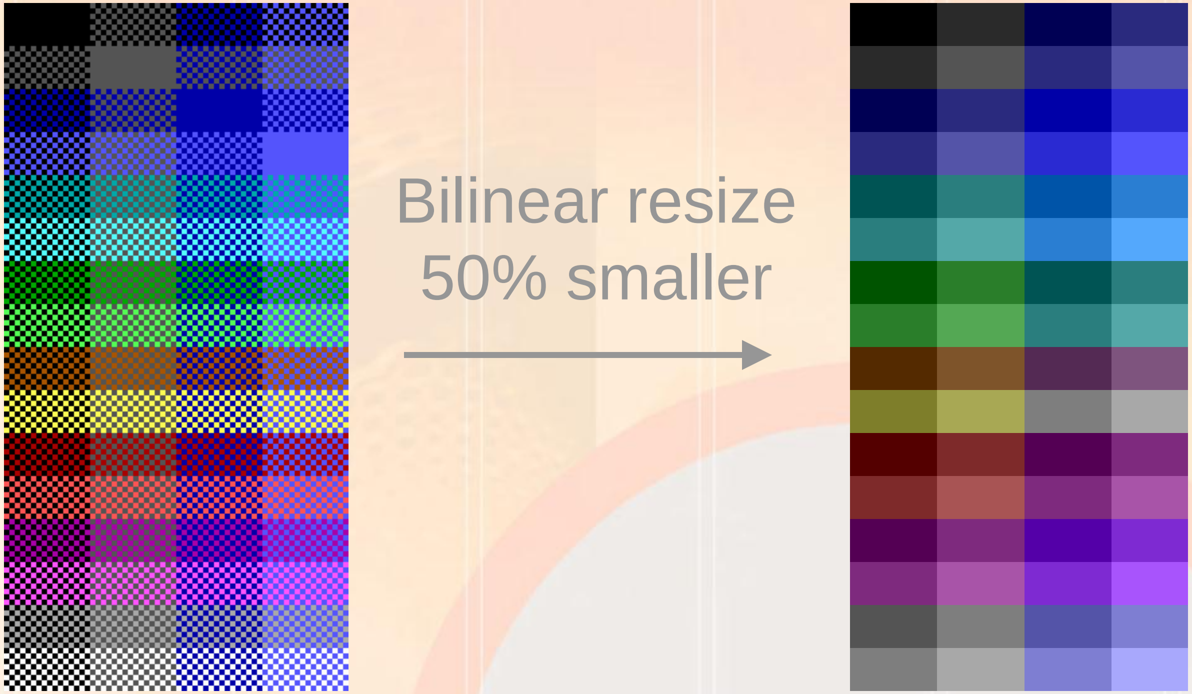


3rd Converter: Results

- Pros
  - Detail, color preserved very well
- Cons
  - Conversion process extremely slow (seconds per frame)
    - **(4*4)(16*16*256)(40*25)=1,048,576,000** comparisons per frame (nearly 2^{30})
    - Actual encoder contains some MMX assembler and algorithm optimizations, but still pretty slow

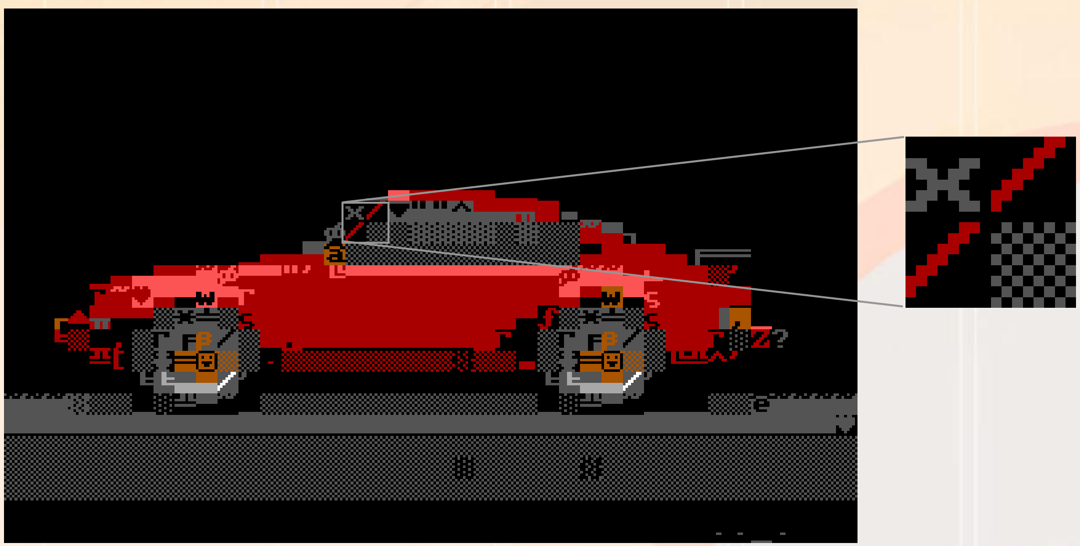
Why did this work best?

- 50% “checkerboard” character (#177), when resampled 50% smaller, better matches solid color areas in the resampled source image



Why did this work best?

- Individual characters can still "match" because both picture and character set resampled by same amount



"Perfect" matches to the forward-slash ("/") character

Choosing Source Material

- Just as important as the converter!
- Visual cortex works best with familiar patterns
  - Faces; human movement (like walking/dancing)
- Be mindful of converter limitations
  - Avoid complicated backgrounds, tiny details, subtle color gradiations
- Pop culture references
  - A little social engineering never hurts ☺

Choosing Source Material

Example

The Player

- Fills memory queue with A/V data
- Starts playback
- Main code tries to keep queue full
- At desired framerate frequency, interrupt code pulls A/V data and sends it to CGA framebuffer and sound card audio buffer
- If queue can't stay filled (slow hard disk), main code pauses playback until queue is full again

How is A/V sync maintained?

- Data stored in A/V “chunks”; each chunk includes video and audio data for the frame
- To get the right timing, we set the sound card’s audio buffer to the size of one audio chunk (for example, 22050 / 30 = 735 bytes)
- When sound card requests (via IRQ) the next audio chunk, we update CGA RAM at the same time
- *Essentially, the sound card drives the entire system*

Comparisons

- ANSI Animation
- Existing Video CODECs
- aalib and libcaca

ANSI Animation

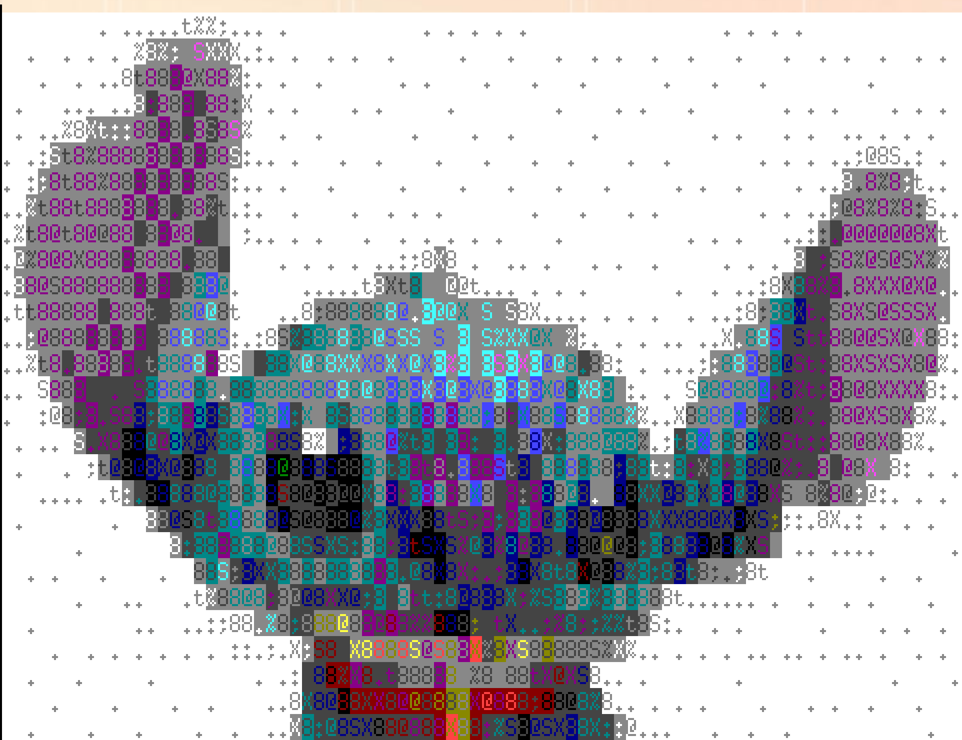
- Nothing in common with ANSI other than the character set and colors
  - Doesn’t use ANSI.SYS in any way
  - ANSI Art is painstakingly crafted by artists; 8088 Corruption is a brute-force conversion
- Artists almost always produce better results

Existing Video CODECs

- 8088 Corruption could be considered a Vector Quantization CODEC that uses a fixed codebook (the CGA ROM FONT + all color permutations)
- Not really a CODEC
  - There is no decompression on playback
  - Didn’t start with CODEC and work forwards (traditional porting) but instead started with PC’s limitations and worked backward
- Better term: Transcoding

aalib, libcaca, libggi (monotext)

- C libraries that convert graphics into text
- Real-time conversion
  - Each input pixel -> output character
- Terminal-agnostic; actual text font data not used during the conversion



Improvement

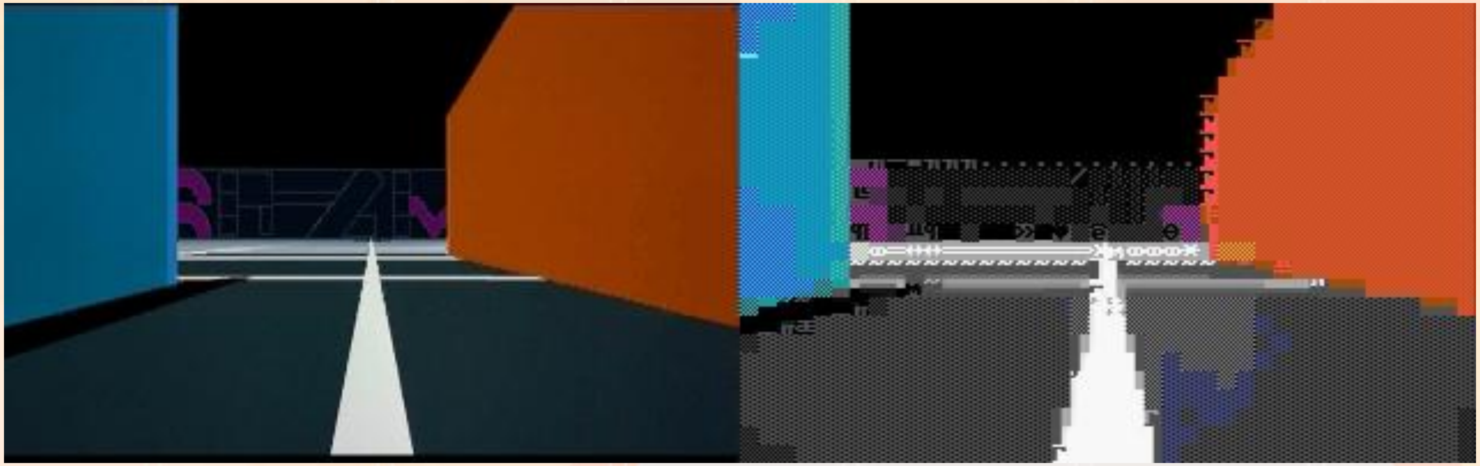
- Everything can be improved!
  - Player
  - Video processing
  - Actual compression
  - Full framerate

Player improvement

- Video updates happen at any time, resulting in shearing. Since system timer tick is free, we can use it to simulate vertical retrace interrupt to implement page flipping.
- EMS support could be added to reduce rebuffering

Video pre-processing

- Subtle variations in the source causes colors to "snap" between two close matches due to CGA's limited color selection
- Pre-processing source video for temporal noise greatly reduces "sparkle"



Actual Compression

- If pre-processing video for temporal noise, 50% or less of the image changes between frames with typical material
- If tests show that partial frame updates can be done as fast or faster than `REP MOVSW`, actual compression can be achieved
- Benefit: Less demand on hard disk subsystem

The Holy Grail: 60Hz Screen Updates

- Updating the screen at CGA's full 60Hz opens up new possibilities
- Technically possible, but with caveats:
  - CPU spends more time updating A/V buffers than it has time to load A/V data from disk
  - End result = rebuffering impossible to avoid ☹

The Holy Grail: 60Hz Screen Updates

- Possible ways to avoid rebuffering
  - Pre-process video to get actual compression (less demand on hard disk)
  - Cache entire video to EMS ("cheating")
  - Read multiple A/V chunks at a time (in theory)
  - Use BIOS sector reads to bypass DOS' double-buffering
  - Drive HD controller directly to use DMA (not portable!)

60Hz Motion

- 60Hz close to the limit of brain's ability to discern individual events in time
- End result: Eerily realistic motion

60Hz Motion

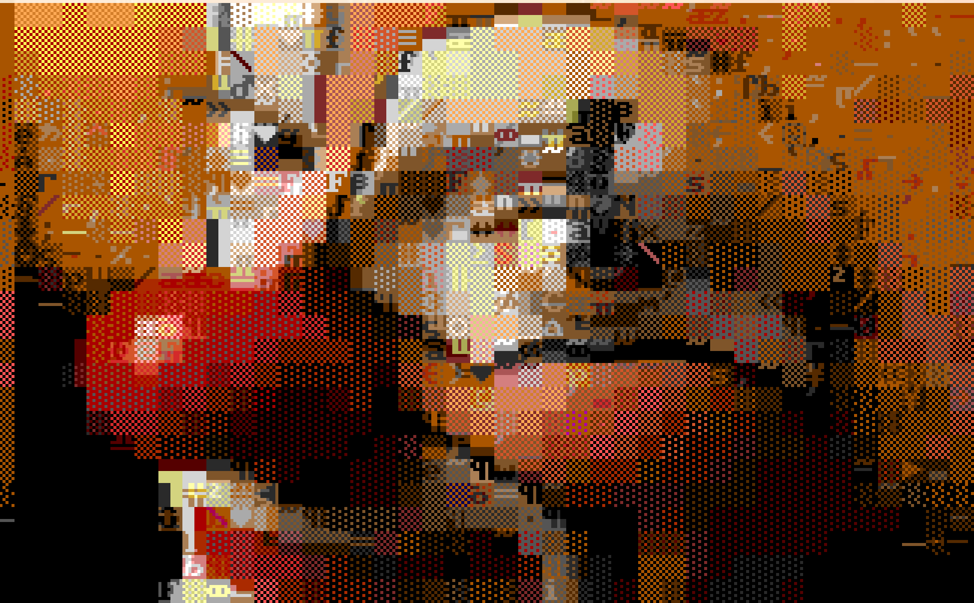
Demonstration

30Hz Dither Across Time

- If we keep track of match errors, we can distribute errors forward in time to the next frame
- 60Hz display rate is fast enough that frames “blend” together, producing additional colors to the human eye

Existing converter

Sample of “time-dithered” converter



30Hz Enhanced Vertical Resolution

- CGA can be tweaked to provide 40x50 mode
- Each tweaked “character” is only the top 8x4 pixels of a character
- No additional resolution, but gain somewhat more flexibility in choosing best matches

Lessons Learned

- Know your output device
  - Having a goal makes it interesting, engaging
  - For low-resource platforms, focus on specs to work backward from, over ideas to work forward to
- Know your input device
  - Know your audience
  - Experiment
- Challenge yourself
  - Stupid ideas are the most fun to work on ☺

Additional Examples

Demonstration

Additional Resources and Q&A

- trixter@oldskool.org
- http://www.oldskool.org/pc/8088_Corruption
  - Downloads, source code
- Any questions?