Pixels Beyond Colors: Exploring Attributes and Representations of Text-Art Images

João Sebastião de Oliveira Bueno
Independent Researcher
Campinas, Brasil
jsobueno@gmail.com

***Abstract*—Due to historical factors and hardware limitation, printed characters have been around to represent images in digital computers for longer than pixel-based raster images exist. With modern hardware and an enrichment of available characters the expressiveness of possible forms of image representation with characters should've grown as an art form. However, the modern use of character-based images as a digital art-form is comparatively small. This work revisits the history of graphic displays, and points to the absence of a widely accepted way to create, store and transmit character-based images using several text-based image capabilities as one of the factors that may be delimiting this expression. It then moves towards proposing techniques and terminology to make these more widespread.**

***Keywords*—image processing; ASCII Art; Unicode; Pixels;**

I. Introduction

Digital images nowadays are pervasive, and usually separated in two big categories: vector based and raster based images. The former can exist in a variety of formats and languages, and have all kinds of different characteristics, even to the point of being able to embed, or being composed of, turing-complete computer programs, such as SVG and Postscript files. We are here concerned with the later - "raster" images, which denote an image in which all elements are explicitly given a value with a predefined size in the output medium. Raster images represented as a rectangular array of square Pixels (Picture Elements). This has even become the *de facto* standard for keeping photographs in society, since the use of film-based photography fell into disuse. Not only still frames, but also all existing video formats are devised so that each frame is ultimately rendered to a rectangular grid of pixels, each pixel holding a single color value.

Also, most devices created for displaying images, including computer displays, televisions, printers, most augmented reality headsets do render a grid-array of pixels. Even vector images, otherwise free from pixel representation when being created, stored or in transit, are usually displayed in a device that will convert it to a rectangular array of pixels.

On the other hand, there is a computer application category in which another rectangular grid with information in each position is used daily by millions of people: a computer terminal - these days, more commonly, a computer-terminal emulator in which system developers and computer operators type in commands, monitor the health of running systems, trigger the build of complex software and several other activities. In computer terminals, each position in the grid, usually termed as a "character cell", can hold not only a foreground color, like in images, but a glyph representing a character - and this glyph can have a foreground color and a background color. Moreover, most terminal emulators allow for extra text-attributes indicating the character displayed can, for example, be in bold, underlined or blinking.

Though we usually don't think of what is displayed in a terminal emulator as an image, it is so. While it can't represent a photography with the same real-to-life likeness that a program that will use the device-pixels, much smaller than a character block, for each element, it is an image that have an intrinsic aesthetics, allowing not one, but various art categories to exist - as well as being a medium more than appropriate to render graphical information about any numeric chart like we would use to view and compare measurement values in spreadsheets or scientific applications. Moreover, in the 1980 decade, competing computer architectures without so called "high resolution" graphics capabilities, and text-only modes, implemented custom character glyphs in the 128-255 range, not used by the 7-bit ASCII standard ([1]), to enable a finer representation of images, subdividing the character blocks in geometric areas. Character sets with all the combinations of quadrants of a character block and sextants of a character block were in use, and are now part of the Unicode specification. That means one can mimic images in a text rectangular grid with a resolution up to 6 times greater than the character resolution, using these blocks to draw up to six pixels per character cell. Even eight pixels per cell are possible if the braille unicode characters are used for the same purposes, although such use is font-dependent, and not recommended by Unicode itself.

This article is organized in the following way: this Introduction presented the current uses for Digital Images, the main categories in existence and the need to formalize images represented with more attributes than a single color per picture-element.Section II will bring an historic overview of how images evolved in digital form, and how we got to the current state. In Section III we have an analysis of the parameters needed for fully representing a text-based image's picture elements and existing and possible models of Text Art. Section IV explores possible file-formats and structures for archiving and exchanging text-pixel based images, and Section V brings a conclusion.

II. History and the current state of Digital Images

In a typical raster digital image currently we have pixel information conveyed as what got popularly called "true color": meaning each pixel carries its own color information, with components for 3 primary colors (red, green and blue) with 8 bit of information per component: a value ranging from 0 to 255 representing the intensity of that component. Although other pixel formats exist, each having different ways of representing human-perceived colors as an array of numeric components, and different depths than 8 bit, what we care about in this article is that the pixel color is the only information in each pixel. Some image formats and applications allow for transparency, and that is encoded as a 4th channel called "alpha". Its precision can vary, but is usually the same used for the color components - so we have the color and transparency (or opacity) information for each pixel.

While raster images are represented by pixels for over 6 decades now, few things have changed in recent decades - among which the following features are built upon the original concept of one numerically represented color per pixel. We list the development and capabilities of pixel based images in order to contrast them to "text based images" later on.

1) Square Pixels: Most raster images have standardized to a square pixel (1:1 ratio). Up to the 1980s, as computer graphics in 8 bit personal computers were limited by hardware resources including memory and video signal generation, it was not uncommon to have screen resolutions that featured non-square pixels, and a distortion that would have to be taken in account when reproducing a given digital image in a different device. Nowadays basically, image pixels are decoupled from device pixels, and the later anyway, just present an API were pixels are square to any relevant drawing software. (e.g. DSLR cameras will use non-square pixels in their "raw" format which directly maps sensor data information, but will usually encode the image into a jpeg file with square pixels). Early pixel based image files were rather a dump of the video-memory representation of an image, and since some of the early raster pixel based display systems would not feature a 1:1 aspect ratio, that ratio was carried on the image. One such non-square system example is what is possible in the IBM CGA ([2]).

2) Transparency: It is now common to have transparent pixels, allowing pixels to be more or less transparent, up to fully invisible. The so-called "alpha channel" is commonplace as a 4th channel along with one channel for each of Red, Green and Blue colors, and contains independent opacity information. Of course, that only makes sense if the image is to be overlaid on a surface with another background - such images are typically used as textures inside computer applications so that the image with transparent pixels merge seamlessly into a larger, "container" image. [3]

3) Colors: over the years pixels went from simply indicating "set" or not "set", known in literature as "bitmap" graphics, to selecting a color from a predefined palette, to use "full color", in which each pixel carries independent color information to deeper bit depths for representing each color. Again, this evolution follows the increased computational capability, mainly available memory, for representing images, as images that are able to display an independent color value for each pixel component. The GIF image format, created in the late 1980s deliberately used a 256 color palette allowing the information for each pixel to fit in a single byte, thus getting to a compromise of memory usage vs quality.

4) HDR: - "High Dynamic Range" in which pixels can depict color values in non-linear ways, allowing scenes as perceived from the "real world" to be represented with more realism: since human eye color perception is not linear, trying to linearize the intensity of colors with a flat number will usually make for less colors than perceivable. A set of techniques to display non linear color values, being able to add detail both to shadow and to high-lighted areas is now used when passing digital images around, usually with information associated in the image meta-data, explaining how the numeric values in the pixels are to be mapped to light or ink in the final output device.

5) Color Profiles: Images have a "color profile" information associated and usually attached to their file, which informs about standard transformation all colors in the image go through compared to a standard, allowing color reproducibility, and thus similar perception, across a wide variety of display or printing devices. These are also used to describe the curve mapping from pixel values to real-world light or color intensity, enabling HDR.

6) Image Metadata: Not strictly associated with Pixels, rather with image files, metadata associating data not directly mapped to the pixels, rather describing information about the image as a whole (like color profile, above), had been in common use. These include authorship and copyright information, geolocation and many other ends not related to the present work.

A. *Text-Based Displays and Image Representation*

It should be noted that text-based images are nothing new. Actually in the early days of digital computers, the first human readable output was printed text, in line printers adapted from the accounting industry ([4]). While it is not easy to trace the first uses of digital computers to represent imagery, drawing with characters, it is almost a natural development. For one, the breakthrough work in mathematics that first provided a visualization of the Mandelbrot Set was published as late as 1978, and would display both a Julia set and a Mandelbrot set for maye the first time, presented both as textual images ([5])

Later on, one of the longer standing terms to refer to text based images was established as "ASCII Art" - referring to drawings made with characters in monospaced text environments. Such images can be simply copy and pasted around, even across the Web and social media. However contemporary text-media on the web and messaging apps will usually be rendered with varying width fonts (as opposed to monospaced fonts) - and most "traditional" ASCII art representations, specially multi-line drawings, created by the thousands in applications that used text with monospaced fonts in the 1990s and prior years won't be displayed correctly.

The "ROFL Copter" (fig. 1), although crude, is a good example of copy-and-pastable ASCII art that was used as today's "memes" in the 1990's internet.

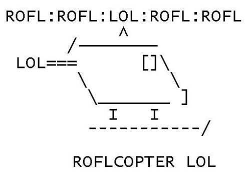

Fig. 1. ROFL-Copter - a famous ASCII Art work distributed as a meme in the 90's

It is interesting to note that the "file type" needed to convey such ASCII Art images is a plain text file. The only special thing about it being the new-line character representation that could change across computer systems - otherwise the same file could be displayed with no special coding across different systems and output devices by simply printing it as text lines. That is in contrast with pixel based images which need at a minimum a predefined protocol of how the image-data is to be arranged as pixels, either hard-coded in a program, or, as became common later on, as metadata prefixed as header information on various image file formats, including PNM, TIFF, GIF, JPEG and PNG among several others.

As mentioned, even before it was usual for computers to feature an interactive screen, representing images through a character grid was a common place. One of the early human-readable output devices for computers being text-line character only printers, which, in contrast with more modern dot-matrix printers, could only print preset characters and had no way to print arbitrary graphics. If one would resort to use the "." (period) character for pixels, it could only be placed aligned with a character cell, with a lot of surrounding empty space: that would result in a low contrast image, and would be characterized as a text image anyway. So, drawing with text characters that was once the only way to create images by using a digital computer: up to the late 1970s there was no way to create raster-arbitrary pixel graphics on computers, even on expensive setups - although the IBM 2250 display, from 1964, could do vector graphics by connecting line-segments in a grid of 1024×1024 coordinates (at a cost of USD 280,000.00) [6], it still would not do pixel based images as we know them today: it would take a list of vectors to draw on that grid, even if a vector would be only drawing from one grid position to a neighbor one, it would still be a line segment, not a single dot as we understand a pixel today. And it would draw text using several such line segments for each glyph, that being the origin of a distinct look and feel for what are instantly recognized as "vintage computer fonts", like 3270font (fig. 2).

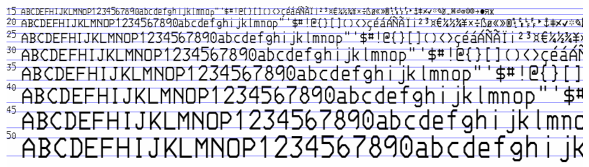

Fig. 2. 2370 font: a modern font targeted at code developers resembling the look of fonts in vector based CRT terminals of the 1970´s ([7])

In the early 1970s graphic displays using raster lines in CRT monitors became somewhat common [4], and by 1975, the Sphere I, the first Personal Computer that could generate a raster video signal with text was launched (soon to be followed by the Apple Computer I). The Sphere I would feature characters in a 32×16 matrix, each cell able to represent 1 of 64 characters from an early ASCII set. [8] pp. 38.

The reason computers at this time could generate text and not pixel-based raster graphics was simply memory, an expensive resource: a 512 byte buffer was able to display a 32×16 character matrix, with the use of auxiliary character generator hardware and a 2KB ROM containing a rasterized font. A single byte representing the character would be placed in a pre-defined memory cell, which would then be rendered to a full character ([8]) . A 256×192 B&W pixel based display, needed for a 32×24 text display of 8×8 pixel characters would require

6KB of RAM ([9] page 3.) - as used by the 1982 model ZX-Spectrum.

From 1975 to the early 1990s, tens of 8 and 16 bit micro-computer architectures were made available - each of them with a unique display configuration able to generate an interactive display with text, with later models able to generate raster-graphics. Several of these models would define their own custom character set for codes 128-255, as the ASCII table would just define characters up to code 127. While some computer families would use the proprietary charset to define regional characters, like cyrillic or Hebrew, others would focus on enabling a richer graphics experience, defining glyphs like combinations of 1/4 block filled characters cells, characters for drawing frames and double-frames of text, with corners and joints, and figures like hearts, spades, clubs and diamonds and others that have later evolved to today's emojis. Several of these custom characters were later incorporated into Unicode ([10]) and are today generally available in the web and other computing environments.

Some of these text display architectures would feature, besides the text itself, character attributes - like reverse text, or underline, and even blinking text. With the popularization of color displays, the possibility of having a foreground and a background color per character cell also became available. One of the most popular text screen formats that allowed such a per-character color representation was the IBM PC - Color Graphics Adapter (CGA) video mode 2, featuring a 80 column x 25 lines text capability, with 2 bytes per cell: one for storing the character code, and another full byte for the color, "bright" and "flash" attributes for each character. This text mode was kept standard for the PC even for later graphic adapters like the EGA and VGA. [2].

As far as storing and retrieving a "full screen art" for this text mode one could simply dump the first 4000 bytes ($80 \times 25 \times 2$) of memory contents at the text area (hardcoded at the 0xb8000 memory address for CGA) to disk, and restore these same bytes to that memory location for re-display. Each "text pixel" in this format will take 2 bytes.

Other architectures would use the highest order character bit, always 0 for the ASCII range, for the "highlighted" or "reversed text" attribute.

Meanwhile, a little before, and in parallel to microcomputers displays, mainframes text-terminals also evolved in their capabilities - the Digital Equipment Corporation iconic VT-100 terminal, from 1978, was one of the first to support what are now called ANSI Escape Sequences - a standard published to support in line control codes to select text attributes and control the text flow. This means advanced text formatting capabilities could be used by a program writing only sequential data to its "stdout" stream, with no direct access to video-memory or need for Operating System calls for scrolling text, positioning the cursor or clearing the screen, as was needed in micro-computers. These ANSI Sequences were designed to be vendor neutral, and after their popular use in hardware terminals for mainframes in the early 80s, were kept alive in modern day Unixes - including Linux and MacOS, whose text-console applications are known as "terminal emulators", as they mimetize the capabilities of those hardware based ones. Microsoft OSes could make use of ANSI control codes through special configurations in their terminal programs, and even in 2023 that is still not the default for Windows systems. The direct impact for that is that rich text applications which run in the terminal have to take special provisions, often providing their own ANSI code to text flow and attribute managing emulation layer, in order to work in Windows, even 4 and a half decades after the introduction of the VT100. ([11])

The "ANSI character codes" themselves were not published all at once, rather, relying on a series of documents built upon the original ASCII (formally specified in the ASA X3.4-1963 document), by the American National Standards Association, culminating in the ANSI X3.64 [12] specification.

III. Text-based Images

As stated in the introduction, once we get to a textual rectangular grid of characters, we can represent images with completely different information than simply a color per image element. One of our proposals is to keep calling such elements "pixels", although a better name can be found for them. For the record of similar words, digital information describing volumetric objects in a given 3D rectangular grid already used the term "Voxel".

A. Application Types

As far as talking about full-images for text terminals may sound out of the conventional usage, and reserved just for niche art-related applications, the truth is that more than one class of applications can make use of text-based images, and not all art-related.

1) CLI Applications: Developer and operator tools like compilers and scripts have evolved using mostly a single serial output stream for all their results - although a lot of modern tools will use seamless cursor-movement commands and emoji to mix things like progress bars and "passed" ticks into the streamed result. CLI stands for "Command Line Interface"

2) Informational Applications: Long time informational applications will use screen refreshing to display a table of values continually updated - one classic such application being the posix "top" command.

3) TUI Applications: Classic business application environments will use a rich screen layout to display fields of information which can be updated in place, and navigated through keys like “tab” or arrow keys. Mainframe COBOL Applications would use this approach from as early as the 1970s. The 80s saw tools like Dbase III and Clipper which would allow for the development of useful and practical businness applications in a matter of minutes. In modern days, there is the Textual [13] framework by Will McGugan that uses unicode characters in innovative ways to draw borders and relief buttons, and popularized the term Text User Interface (TUI) in contrast with Graphical User Interface.

4) Artistic and Entertainment Applications: Although not widespread, there are hundreds of video games, and tens of applications for “painting” in the terminal which make use of text as graphics. One such app is terminedia-paint by João Bueno [14], which among other features, will allow one to paint with arrow keys or even a mouse, type text in any of up, down left or right directions and draw textual subpixels by setting smaller squares inside each character cell, using 1/4 blocks, 1/6 (sextant) and braille unicode characters to effectively create finer images than would be thought imaginable in a terminal.

Terminedia-paint is based on the Terminedia framework [15], a Python project, which although not aimed at non-programmer artists, do pack powerful tools for creating text-art, including a simple drawing API able to do lines, rectangles, bezier curves, the capability of rendering “big text” by using rasterized fonts using $8x8$ pixels per character in a 6 pixels per character cell subpixel mode, using unicode Sextant characters, outputting text in arbitrary directions with a programmable flow, instead of just left-to-right, newline rewinds to one line down, left margin, mixing in emoji characters and unicode-translations to emulate character effects. (fig. 3)

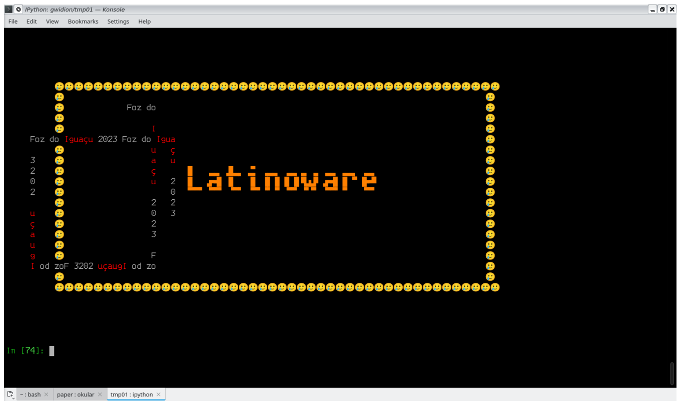
IPython: gwidion/tmp01 — Konsole
File Edit View Bookmarks Settings Help
Foz do
Foz do Iguaçu 2023 Foz do Igua
Latinoware
od zoF 3202 uçaugI od zo
In [74]:
~ : bash
paper : okular
tmp01 : ipython

Fig. 3. Text art demonstrating some of Terminedia’s capabilities by issuing Python statements in an interactive session

B. Text-based image types

A text-based image will vary how it looks like depending a lot not only on the authors intention, but on the capability of the tool used for its creation and also, the tool or media intended for its exhibition.

1) Classic ASCII Art: Early ASCII Art examples would be hand-crafted in a 2 Dimensional ext editor, of the same kind suitable for writing computer programs, and use “typeable” only characters - mainly the subset “\ / + _ |” to draw horizontal, vertical and diagonal lines, and a couple other characters for filling up regions, or draw “eyes”.

In this category, we may possibly add “emoticons” - which would usually fit in a single text line, and could be used inline in text messages, such as smiley faces “ :-) ” or a crying feature “ :’-(. These became so widespread that they eventually found their way as standalone characters as specified by Unicode, in what is now known as the Emoji character category. (fig. 1)

2) “Realistic” ASCII Art: A separate category can be given by images that will emulate light and shadow areas of a real-world photography or portrait by using a set of characters that have a different weight of “ink” due to their visual density. A scale of contrasting characters such as “ .,-OG*” can be used to simply replace pixel-vales from light to dark creating a visually compelling example, which, given a large enough character grid, can be immediately identified as the original image. (fig. 4)

3) EMOJI Loaded text: Some of the needs of integrating graphics into text messages evolved naturally from ASCII art emoticons into special characters that ended up receiving a separate category in Unicode: the Emojis. The most obvious case being the porting of the smiley drawn with “:-)” to Unicode’s “WHITE SMILING FACE” (code=`0x263A`, “”). Emoticons are special enough that they can be drawn as multi-color graphics even in most setups where all other glyphs must be monochromatic. The use of Emoji is nowadays widespread in communication apps and social media, but the lack of specialized tooling combined with the dominance of variable width fonts in these media, restrict their use to a linear fashion, with no possible way for one to draw a more complex shape combining several emoji or other special characters.

4) Colored Character Blocks: An option used by several tools is to take advantage of color capability in terminals and use the Unicode “Full Block” (\u2588) character to represent pixels. This approach emulates pixel-based raster images and doesn’t use any of the special characteristics of text art. (fig. 5).

5) Unicode special shapes: Although Unicode had defined several characters usable for drawing complex forms such as character sub-blocks in all combinations, checkered patterns,

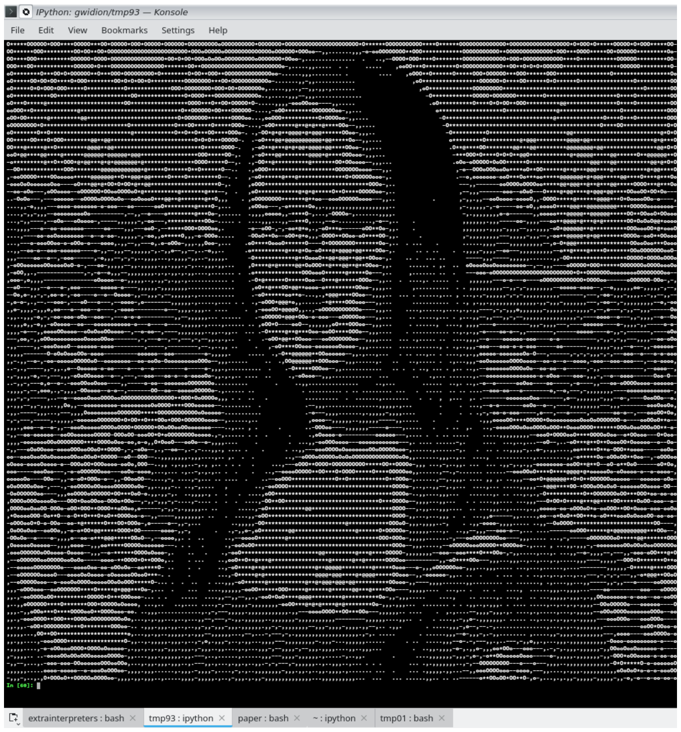

Fig. 4. ASCII Art for the Monalisa painting. Credits: the Author using the Terminedia Python framework

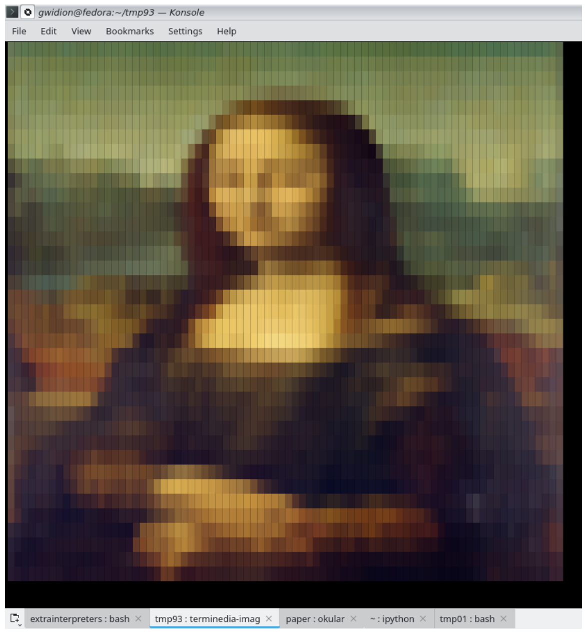

Fig. 5. Monalisa painting rendered on color terminal with Full Block characters. Credits: the Author using the Terminedia Python framework

triangle parts, specialized corners and frame-drawing characters, the lack of proper tools to draw (and type) these characters make art using these characters, a lot of which introduced in Unicode in 2019 [10], virtually non-existent.

6) Special character effects: Beyond colors, text terminal emulators will usually add upon capabilities present in the VT-100 terminals - which means special character effects such as reversed text, underline, strike-through and even blinking text are a possibility in several terminal emulator programs - but none of these exist as part of a text art corpus, again for lacking of a convention to replicate such effects.

7) Colored Text combining Words and Images: Digital artists can sometimes go the extra mile and work on their own toolchain to be able to create art featuring letters combined in words, colors, making up shapes, enabling a unique authorial style. One such artist goes by the artistic name 1mpo$ter on social media, creating not only text-based stills, but looping animations that, although rendered to rasterized videos, display character-cell based images. (fig. 6). It is interesting to note that one of the World's most recognized text art, the character cascades used to represent entering into the "cyberworld" in the Matrix ([16]) movie series, falls in this category,although that art is slightly enhanced at the pixel level, after the characters rasterized, with each character featuring a palette of similar shades instead of being monochromatic to mimic a vintage phosphor display. This implies that the movie effect cannot be fully replicated with text only graphics.

IV. Storing and Transmitting Text Art

As mentioned above, one way to store and transmit text based images is to simply write a plain-text file. An universal text-encoding, such as UTF-8 can even allow for any Unicode defined character to be part of the result.

Moreover as raster lines are defined by the newline indicator (in posix systems a \x0a, ASCII newline character), no meta-data except the text encoding is needed to allow the file to be interpreted and re-rendered.

However text files will hold only character information, and no color, positioning or special effects - except that, as the ANSI Codes [12] were designed for inband use, if one just store the corresponding code for each character attributes in the file, it will just work: one can create a "plain" .txt file with ANSI sequences to reposition the cursor, if needed, and change the printing foreground and background color for each text-pixel.

In fact, the Terminedia [15] framework for text art uses that output format. (it can't at the time of this writing, read and interpret embedded ANSI codes, though). The figure (fig. 5) above, for example, can be encoded in a single file with the

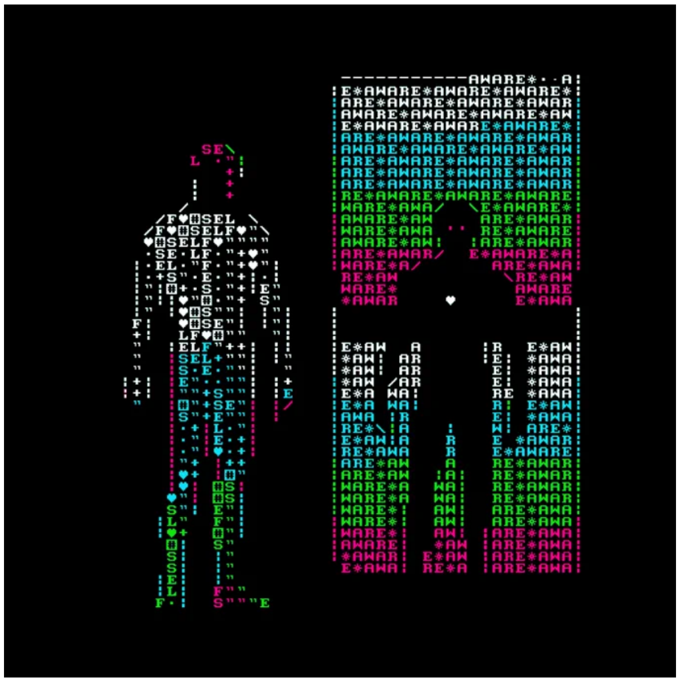

Fig. 6. Still frame from the animation "Self Awareness" by _1mpo$ter on twitter ([17])

Fig. 7. Detail of frame from the film Matrix [16] showing shaded characters in the iconic cascade.

"terminedia-image" script that comes along with the framework, and can be redisplayed in any terminal with the common posix "cat" program. The same framework, however, uses internally a 4-tuple for each character cell, in order to have all possible characteristics for a text image: character, foreground and background colors, and character effect, to which it adds some unicode character translations (like subscript) to the well known terminal effects of reverse text or blinking.

The major disadvantage here is the potential verbosity of such a file - a singe color in ANSI Code is represented by a sequence like "`<ESC>[38;2;255;255;255m`" that is, up to 17 bytes for one color, and then 17 other bytes for the background color - if we add effects, that is aout 40 bytes per character, except that characters that will continuously be set to the same attributes in the same row (like what is common for written text, but not for text representing a picture), some space can be saved as the same attributes are re-used.

1) Needed Resources for a Binary-Text-File: In contrast, if an application would save text art in a binary format derived from the idea used y the original IBM CGA [2], this would take 4 bytes for the character (using UTF-32 encoding), 3 bytes for each color, and 1 byte for text effects, if there are less than 8 possible effects, for a total of 11 bytes per character cell. The resource usage for this is actually negligible for today's memory and CPU capabilities, even for text terminals of a rather large size. A terminal using full screen on a 4k device $(3840 \times 2160px)$ would allow for a 480 column $\times$ 270 lines of text with an 8×8 character glyph grid - and that would use 1,425,600 bytes: less memory than needed to hold a 1024×768 full color image (2.3MB), a typical "one megapixel" image from around the year 2003.

Also, it is important to note that any rich-text file format can be used to convey most information available at text terminals. Usually we don't think of them like that because all default to use a variable-length font, though a monospaced font can be explicitly selected. These files include, but are not restricted to: HTML Files with embedded styling information, Open Document text files .ODT Microsoft Office Open file format for text (DOCX), and Rich Text Format - but these files can only be rendered back to display formatted text inside the respective supporting WYSIWYG style applications (including a Web browser in the HTML case). There is currently a lack of tools able to render a subset of some of these files into a text terminal environment. As for the size, these files require a lot of extra metadata on one hand, but support compression on the other, and generally are stored in a compressed form (but for HTML). Also, for all of these formats the default "way of writing", even if configurable by adjusting the styles, is with a variable width font, and automatic line wrapping at the document width, with an explicit line break introducing a new paragraph, instead of just proceeding to next physical line on column 0: not sensible settings for creating drawings with text.

And least, but not least, any application could render text-art with all the attributes we are considering into a conventional pixel-based raster file, like a .PNG file - just by picking a font and rendering it. We actually even use this strategy in this article for the example images, and the visual impact is preserved. Even the "blinking" effect could be represented in an animated raster file like what is possible using the .GIF format.

Wrapping up, while there are several alternatives for conveying text art, including colors, only a txt file with embedded ANSI codes is currently supported due to inexisting tooling, and there is an unknown demand for a format that could allow for easy-to-edit text art. But that would not be carried straightforward to a printer, for example: one would have to render the art in a terminal program, take a pixel based screen snapshot, and print that instead.

V. Conclusion

In this brief overview on the state of computer graphics and the possibilities for text-art, we could say the main take out is that there are many possibilities in using text for digital images with artistic purposes. In particular when one takes into account that each picture element will include not only a color attribute, but a foreground color, a background one, a character and a possible character effect, and may, as well represent any of the tens of characters available in the Unicode set.

However, there is a current lack of tools that would enable the full use of those capabilities, combined with special shaped characters that exist since the 1980's. Also, it is not trivial to select a file format to store, transmit and replay such works of art.

There could be a significant repressed demand for such tools, and making then available, or more known, could possibly "make ASCII Art great again", along with other digital midia.

Acknowledgments

Special thanks to Marcos Siriaco Martins for so many years making Latinoware taking place and to Christiane Borges Santos for been always ready to help me put this in written form.

References

[1] ITI, "2007 five-year maintenance review of incits/l2 standards: Incits 4 :1986 [r2002] information processing - coded character sets - - unicode," 2007. [Online]. Available: http://unicode.org/L2/L2006/06388-review-incits4.pdf
[2] IBM , *IBM Color/Graphics Monitor Adapter*. [Online]. Available: https://minuszerodegrees.net/oa/OA%20-%20IBM%20Color%20Graphics%20Monitor%20Adapter%20(CGA).pdf
[3] *Compositing digital images.* SIGGRAPH, January 1984. [Online]. Available: https://doi.org/10.1145/800031.808606
[4] A. F. Mayadas, R. C. Durbeck, W. D. Hinsberg, and J. M. McCrossin, "The evolution of printers and displays," *IBM Systems Journal*, vol. 25, no. 3.4, pp. 399–416, 1986. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5387686
[5] R. Brooks and P. Matelski, "The dynamics of 2-generator subgroups of PSL(2, c)," in *Riemann Surfaces and Related Topics: Proceedings of the 1978 Stony Brook Conference*, I. Kra, Ed., 1978. [Online]. Available: https://abel.math.harvard.edu/archive/118r_spring_05/docs/brooksmatelski.pdf
[6] Keydata Corp., "Computer display review," pp. V.1980–V.1964, March 1970. [Online]. Available: http://bitsavers.informatik.uni-stuttgart.de/topic/graphics/ComputerDisplayReview_Mar70.pdf
[7] R. Bánffy, "3270font," 2018. [Online]. Available: https://github.com/rbanffy/3270font
[8] Sphere 300 Corporation, *Operator and Reference Manual*, North Salt Lake, UTAH, USA, October 1976. [Online]. Available: http://bitsavers.trailing-edge.com/pdf/sphere/300-76-0001-1_Sphere_Operator_and_Reference_Manual_Oct76.pdf
[9] S. Collins, "Computer graphics during the 8-bit computer game era," *Jupiter*, vol. 80, no. 3K, p. 8K, 1998. [Online]. Available: https://publications.scss.tcd.ie/tech-reports/reports.98/TCD-CS-1998-15.pdf
[10] Terminals Working Group and Doug Ewell and Rebecca Bettencourt and Ricardo Bánffy and Michael Everson and Eduardo Marín Silva and Elias Mårtenson and Mark Shoulson and Shawn Steele and Rebecca Turner, "Proposal to add characters from legacy computers and teletext to the ucs," International Organization for Standardization, Working Group Document, January 2019, proposal to add characters from legacy computers and teletext to the UCS. For consideration by JTC1/SC2/WG2 and UTC. [Online]. Available: https://www.unicode.org/L2/L2019/19025-terminals-prop-no-attachments.pdf
[11] J. Hartley, "Colorama," 2023. [Online]. Available: https://pypi.org/project/colorama/
[12] American National Standards Institute, "X3.64 additional controls for use with american national standard code for information interchange," 1979. [Online]. Available: https://www.govinfo.gov/content/pkg/GOVPUB-C13-3316fefd2127df2830d92e7a7411541c/pdf/GOVPUB-C13-3316fefd2127df2830d92e7a7411541c.pdf
[13] W. McGugan, "Textual: A rapid application development framework for python," 2023. [Online]. Available: https://github.com/Textualize/textual
[14] J. S. O. Bueno, "terminedia-paint: Interactive app to create ascii-art and unicode art in an interactive way directly from the terminal," 2021. [Online]. Available: https://github.com/jsbueno/terminedia-paint
[15] ——, "terminedia: Python3 library for multimedia functions at the command terminal," 2023. [Online]. Available: https://github.com/jsbueno/terminedia
[16] L. Wachowski and L. Wachowski, "The matrix," 1999.
[17] Imposter, 2023. [Online]. Available: https://twitter.com/_1mposter/status/1701964733295841479