Original Study

John Aycock*, Andrew Reinhard, Carl Therrien

A Tale of Two CDs: Archaeological Analysis of Full-Motion Video Formats in Two PC Engine/TurboGrafx-16 Games

<https://doi.org/10.1515/opar-2019-0022>

Received March 18, 2019; accepted July 16, 2019

Abstract: As an example of how the archaeology of modern/contemporary media can be conducted, we examine the technology behind artifacts with cultural relevance in modern society: video games. In particular, we look at two game artifacts from the PC Engine/TurboGrafx-16, a game console produced from 1987–1994. A 1× CD-ROM drive could be added on to the console, with a corresponding increase in the amount of data a game could access, and some games took advantage of this capability to include full-motion video (FMV). This digital excavation report details the FMV formats of two such games along with the methodology used to reverse engineer the formats and verify the correctness of the analysis.

Keywords: video games, CD-ROM, digital media, full-motion video, reverse engineering

1 Introduction

Domestic video game entertainment in the 1980s developed into two distinct digital enclaves. Console gaming was seen as a novel and cheap way to bring the arcade into anyone's living room, while personal computers were typically less competent to translate the frenetic action of popular arcade titles. Part of the attraction exerted by the more expensive personal computers came from the versatility and affordability of the preferred storage medium (magnetic cassette tapes or diskettes, versus the expensive ROM chips used for consoles). Larger programs could be run from multiple disks, and as prices of hard disk drives fell, larger programs could be installed and run from there as well. For game developers, this burgeoning capacity created an incentive to explore data-intensive designs. Players grew fond of the expanded breadth and audiovisual beauty that became typical of adventure games and role-playing games at the time. While some computers were designed specifically to bridge the overly simplistic dichotomy presented above (namely the Atari ST and Commodore Amiga), console players were limited in their ability to explore data-intensive gaming. The advent of the PC Engine CD-ROM² in 1988 can be seen as a deliberate attempt to feed this particular type of technological attraction.¹

The very name of the core console – the *PC Engine* – is a clear indication of the designers' intent to create a bridge between the two digital parks. While Hudson Soft had gained expert knowledge on the type

1 The “2” after “CD-ROM” isn't a footnote; this was how the CD-ROM unit was labeled.

*Corresponding author: John Aycock, Computer Science, University of Calgary, ICT 602, 2500 University Drive NW, Calgary, Alberta, T2N1N4, Canada, E-mail: aycock@ucalgary.ca

Andrew Reinhard, Department of Archaeology, University of York, York, YO1 7EP, UK

Carl Therrien, Département d'histoire de l'art et d'études cinématographiques, Université de Montréal, Montréal, H3T 1N8, Canada

of hardware components needed to bring the arcade home (as one of the prime developers on Nintendo's Famicom), NEC had already established itself as one of the major players on the personal computer front in Japan with its PC-88 line. The CD-ROM² attachment for the PC Engine/TurboGrafx-16 was envisioned as an expansion to the “core” architecture when the two started working together. In December 1988, it became the first CD-ROM device released for domestic video gaming.

PC Engine games were released on tiny cartridges named HuCards, which were physically like bulky credit cards and integrated miniature ROM chips. Nearly all games throughout 1987 and 1988 could fit perfectly on these HuCards in a 250 kibibyte² chip. In this context, the advent of the CD-ROM format could be seen as a great challenge for video game developers; original advertisements for the platform noted that storage capacity increased by more than 2000 times. A way to consume the newfound capacity was for games to integrate full-motion video sequences, including live-action animations and sampled audio. These assets had already been used in adventure games and demos, with obvious limitations. In the case of Newtek's *Demoreel* (1987) or Access Software's *Mean Streets* (1989), they were played directly from a hard disk drive or from RAM. The first generation of CD-ROM drives could transfer data at the relatively limited rate of 150 KiB per second. In this context, the emergence of games using full-motion video (FMV) on early CD platforms represents a historical puzzle. While technological attraction can explain the “why”, we seek to address the difficult question of the “how”. How could such data-intensive assets be stored and then played effectively on the PC Engine's 1X CD-ROM drive?

Here, we analyze two CD-ROM titles in order to expose the mysterious technological expertise buried 30 years in the past. First is *Sherlock Holmes: Consulting Detective* (1991), released roughly midway through the console's life cycle.³ The second is *Kūsō Kagaku Sekai Gulliver Boy*, released towards the end of the platform's commercial exploitation cycle in 1995; it used a novel way to encode FMV, titled “HuVideo”, whose intricacies are presented below. As this paper – this digital excavation report – demonstrates, the puzzling nature of early FMV encoding is not being overstated for rhetorical effect. It highlights the surprisingly alien nature of techniques that are merely 30 years old, engraved on a type of media that is still commonly used today (optical digital discs). The context of accelerated supersession that has been exposed thoroughly by James Newman (2012) has consequences beyond the preservation of video game culture; derelict digital parks, where thousands of artifacts still considered as mere cultural debris pile up and sediment, are becoming archaeological terrains, with very few archaeologists possessing the necessary knowledge to understand the unearthed objects and their usage.

2 Archaeological Framing

It may be useful to step back and consider the initial placement of this work within archaeology; we defer a fuller theoretical discussion, and an examination of the context in which this work arose, to Section 6.

Video games are indisputably part of 20th- and 21st-century culture and, due to their recency, they naturally fall into the realm of contemporary archaeology. In *Archaeologies of the Contemporary Past* (2001), Buchli and Lucas argue that contemporary archaeology's origins lie in the 1960s, making their edited book an interesting waypoint: three decades removed from the 1960s, at a time already replete with digital artifacts, the book's index mentions neither “computers” nor “digital.” Nearly 15 years later, archaeological explorations of digital artifacts were more the exception than the norm: for example, Moshenska (2014) studied a USB flash drive, Perry and Morgan (2015) a single hard drive. For reference, as of this writing, the Internet Archive alone stores over 45 *petabytes* of digital data (Internet Archive, n.d.). The gap between where archaeological method and theory is with respect to digital artifacts and where it needs to be is vast.

² At the time, this would have been referred to using kilobytes, but in this paper we use the modern unit *kibibyte*, abbreviated KiB, for accuracy. 1 KiB = 1024 bytes.

³ We use the release dates as given by Mobygames (n.d.-a; n.d.-b).

This work is intended as a provocation: we argue that (contemporary) archaeology overlooks digital artifacts at its peril. Much modern culture is rooted in digital artifacts and, importantly, computer software such as video games; to fully engage digital artifacts is to remain relevant. Even the area of “digital archaeology” is more concerned with the application of computers to traditional archaeology rather than the archaeology of digital artifacts, as is the very recent “digital archaeoludology” (Browne et al., 2019). Fortunately, at least in the realm of video games, there is a nascent but growing subarea of “archaeogaming,” the intersection of archaeology and video games (Reinhard, 2018), whose boundaries are inclusive enough to incorporate archaeology on digital video game artifacts. Intuitively this makes sense from an archaeological standpoint, because uncovering unseen implementation details of video games is a nice digital analog for traditional “dirt archaeology”.

Digital artifacts reside on media, and it could be argued that this work would instead fall under the aegis of “media archaeology,” but we see this as problematic. For starters, Huhtamo and Parikka (2011, p. 3) are blunt: ‘Media archaeology should not be confused with archaeology as a discipline.’ A critique by Elsaesser pointedly observes ‘there is no discernable methodology and no common objective to media archaeology’ (2016, p. 182), which is the polar opposite of what an archaeology of digital artifacts should aspire to. Furthermore, Huhtamo and Parikka (2011, p. 3) characterize the range of media archaeology as ‘the humanities and social sciences’ with sojourns into ‘the arts.’ Put another way, media archaeology eschews precisely those fields required for a deep understanding of digital artifacts and the technology that produced them. We side here with Moshenska (2014), who recognized the need for archaeological collaborations with computer scientists and others to address the digital.

This brings us back to an observation Hodder (2001) made, that contemporary archaeology must be an interdisciplinary endeavor. When studying video game implementation, trowels cannot suffice; different methodologies need to be employed. Moshenska (2016) highlighted the connections between archaeology and reverse engineering, and discussed the value of reverse engineering in contemporary archaeology. Reverse engineering is exactly the technique we employ in this work: in computer science, reverse engineering of binary computer code sees heavy use in computer security, but it can be equally leveraged to understand the computer code of a video game.

Our contribution here to both archaeogaming and contemporary archaeology is effectively a digital excavation report. Reading it is similar to reading reports on archaeometry, or those that scientifically analyze the chemical composition of clays, or that conduct XRF analysis of coins. Our proof-of-concept case study demonstrates in detail the level of attention needed to ask and answer archaeological questions about digital artifacts. By carefully documenting our reverse engineering, we aim to reveal what Moshenska called ‘the thought-process of the archaeologist-as-reverse-engineer’ (2016, p. 26). Recall that we began with one research question – how is full-motion video stored and played back in these artifacts? – and our work illustrates the complexity of the inquiry and the amount of work involved to answer what might seem to be a simple question.

We begin by introducing the digital artifacts in the next section, and some properties they have that permit independent reproduction of our work, along with the set of software tools we started our analysis with. Section 4 details the analytical process, and Section 5 summarizes the final results. Then, we bookend our case study with a discussion of how our work fits into the bigger archaeological picture.

3 Artifact Descriptions and Initial Toolset

We began with no information about the artifacts beyond the two binary CD-ROM images, which were used under fair dealing for research purposes. As acquired, they were labeled

```
Sherlock Holmes Consulting Detective [U] [CD] [TGXCD1011] [Sleuth Publications]
[1991] [PCE] [incredible_hark]
Kuusou_Kagaku_Sekai_-_Gulliver_Boy_(NTSC-J)_ [HCD5076]
```

We refer to these as SH and GB, respectively. Besides those and the abbreviations listed at the end of the paper, we note that base 16 (hexadecimal) values are prefixed with a dollar sign or 0x. Base 16 numbers might seem an odd choice, given our usual base 10 system, but in fact base 16 is used frequently in computing because it is a terse representation that is trivially converted to and from the computer's preferred binary form. In base 16, digits larger than 9 are represented using the letters A through F (or equivalently, a through f).

The SH `.bin` file containing the CD-ROM data is 686,254,800 bytes long and its MD5 checksum is \$17384201fd019eaec75f0a7a541a344; GB's file was 775,049,856 bytes with an MD5 of \$49c12d7976d1655ae96768ce9f6c41b6. A checksum may be thought of as a digital summary of data, and MD5 is a particular algorithm for computing that summary. While digital artifacts may not always be directly shareable due to copyright considerations, a suitably strong checksum algorithm helps sidestep this problem and ensure the reproducibility of the work: an independently-acquired CD-ROM image whose MD5 checksum matches ours gives a very high degree of confidence that the two images are the same.

CD-ROMs may have separate tracks containing audio or data. All the GB data relevant for this analysis was located on Track 2 only, whereas the entire SH CD contains data. Without loss of generality, the images were converted from their "raw" format (2352 bytes/sector) to retain only the useful data (2048 bytes/sector) using the `bchunk` program on Linux. A sector, for reference, is fixed-size and is the smallest addressable unit of data on a CD-ROM, in the same way that a house is the smallest addressable unit for postal mail.

For greater certainty about the images' contents, they could be compared to the original CD contents. After we did the analysis, we acquired a physical SH CD, and its image was identical to the one we had analyzed. Doing a similar comparison for GB is possible future work, but is in fact unnecessary, and we validate our GB results using a different technique in Section 5.

Generally speaking, we can analyze a game's code using static or dynamic methods; the former involves studying the code without it running, and the latter inspects the code as it runs. Debugging facilities, normally for finding and fixing bugs in programs, can be used for dynamic analysis. For static analysis of the binary game code, we require the ability to disassemble the code: to translate it into low-level (but more human-readable) assembly code. Unfortunately the real hardware does not provide these affordances, and as a result we turn to software-based emulators that both mimic the real hardware and supply the additional functionality we need. The emulators used for analysis were Mednafen 0.9.45.1 and MAME (MESS) 0.187, both built on Linux from their respective source code. The former correctly emulates both games but has limited debugging and disassembling capability, whereas MESS' PC Engine (PCE) emulation is faulty at present – it runs the games but doesn't stream in FMV data from the CD-ROM properly – but it does offer a more full-featured, if generic, debugger. In the end, as described below, we relied more on direct analysis of the CD-ROM data than on game code analysis. Since generic tools were insufficient to ask and answer the questions we had during analysis, we have constructed a number of bespoke scripts written in the Python programming language for this work, which are freely available to allow reproduction of our results and allow others to build upon our work.

4 FMV Archaeology: Analyzing Two CD-ROMs

Considering the novel nature of the analysis conducted in this section, some of the procedures developed are in and of themselves results that should be shared with the archaeological community. Rather than having the results presented as if by magic, it is instead useful to document the overall process that led to our conclusions. Details of fruitless dead ends in the analysis have been omitted.

We began with SH. First we needed to understand where and how SH was accessing its data on the CD-ROM, and we modified Mednafen to enable SCSI logging (SCSI is the interface through which the console and the CD-ROM drive communicate). This was fairly easy, as the logging code was already present in the Mednafen code base. Running SH would cause this data to be printed out when CD-ROM accesses occurred, making it possible to correlate them with in-game activity. For instance, selecting Holmes' Introduction from the main menu (an FMV clip) showed

```

SCSI: Read: start=0x000000e8 (track=1, offs=0x000000e8), cnt=0x00000004
SCSI: Read: start=0x00000f20 (track=1, offs=0x00000f20), cnt=0x00000006
SCSI: Read: start=0x00000f18 (track=1, offs=0x00000f18), cnt=0x00000008
SCSI: Read: start=0x00000f20 (track=1, offs=0x00000f20), cnt=0x000000fc
SCSI: Read: start=0x0000101c (track=1, offs=0x0000101c), cnt=0x000000fc
SCSI: Read: start=0x00001118 (track=1, offs=0x00001118), cnt=0x000000fc
SCSI: Read: start=0x00001214 (track=1, offs=0x00001214), cnt=0x000000fc
...

```

and from Holmes' Introduction menu, selecting the London Library audio (only) clip printed

```

SCSI: Read: start=0x000004a9 (track=1, offs=0x000004a9), cnt=0x00000010
SCSI: Read: start=0x000004b9 (track=1, offs=0x000004b9), cnt=0x00000010

```

This last output tells us, for example, that 16 sectors (the count of 0x10 in base 16) were read from the CD-ROM *starting* at sector 1193 (0x4a9), followed by another 16 sectors starting from the adjoining sector 1209 (0x4b9). From this information alone, we are beginning to glean where the audio data (and, possibly, more code) is located on the game CD-ROM.

We began with the audio-only data, reasoning that it was isolated there as opposed to being mixed in with FMV data. As it happened, this turned out to be the Rosetta Stone that was instrumental in discovering both games' FMV formats. The Mednafen debugger showed the sampled audio's frequency to be 16KHz, and the samples were being played using the Oki MSM5205 chip that was present in the CD-ROM². Using Mednafen's emulator code for this chip as a guide, we wrote a Python script to extract CD-ROM sectors from the SH image, decode the audio samples, and output them as a WAV audio file that we could play independently of the emulator and compare to the original. This worked for the audio-only clips, which were apparent in the SCSI logs as repeated \$10-length accesses (in other words, 32KiB: \$10 sectors × 2048 bytes/sector).

On to SH's FMV. This requires some knowledge of the PCE's graphics system; Figure 1 shows the relevant pieces and how their data interrelates.⁴ Basically, the PCE has two key components in its graphics architecture: the video display controller (VDC) and the video color encoder (VCE). The VDC and VCE work in concert to produce the images shown onscreen by the PCE, and both VDC and VCE have their own private memory (i.e., RAM). As such, the problem of determining the FMV encoding can be viewed as understanding how data from the CD-ROM is placed into the VDC's and VCE's memory.

To get insight into the PCE's VDC memory, we added code to Mednafen that gave a real-time⁵ display visualization using the text-based curses library (Arnold, c. 1980) of the VDC's block attribute table (BAT). The PCE's screen image is subdivided into "tiles" that are 8 pixels across and 8 pixels high, and the VDC uses the BAT to locate information about all the tiles, analogous to using the index in a book. Each 8×8 block of tile data has a 16-bit BAT entry that contains 4 bits of color palette information for that tile, along with 12 bits of pointer information pointing to the 32-byte block of tile data in VDC memory.

Using our added visualization makes it easy to determine certain properties of SH and its FMV behavior. The screen is configured as 64×64 tiles, in which the FMV window is 30×11 tiles, or 240×88 pixels. The FMV window is one region of BAT entries that alternates between pointing to two sets of nonoverlapping VDC memory regions; in other words, SH's FMV is using double buffering, a common graphics technique (Mahday, 2014). The FMV must therefore be continually changing the tile data, as opposed to having a static set of tiles and producing FMV by some clever rearrangement of tile pointers.

⁴ Technical information here on the PCE is not as prevalent as it is for other platforms; we draw primarily from (Archaic Pixels n.d.-a).

⁵ Real-time but not instantaneous. Strictly speaking, it's invoked from Mednafen's emulated VDC vertical sync routine.

Figure 1. Overview of PC Engine graphics architecture and data flow.

We initially assumed that the FMV consisted of a static base image, or “key frame,” to which the animation was encoded in terms of frame-to-frame changes; typically there is relatively little change from one video frame to the next, and substantial space can be saved by only storing the differences. We modified Mednafen to take the VDC memory that the BAT entries pointed to and save it into a file, and then looked for that data in the CD-ROM sectors that we saw accessed in the SCSI log. We found an exact match, revealing where it was located but also meaning that no compression was done on that image’s data.

From static analysis of the SH code (a disassembly captured using MESS), we’d spotted invocations of the audio-playing routine from what appeared to be the FMV-playing loop. It seemed a reasonable working hypothesis that the audio chunks were scattered regularly throughout the FMV data, and that a small portion of audio was played each video frame. (We had some assurance that the audio was not in larger pieces, because we’d tried using our Python script for audio extraction on a whole set of FMV sectors and only heard noise.) As the audio was already well-encoded, we also expected that it would not have been subject to further compression, and would appear as is in the FMV data.

To test these assumptions, we modified Mednafen to save the encoded audio data playing during the game emulation. We then wrote a Python script to search for that data in pieces through a sequence of CD-ROM sectors. The output for the Holmes’ Introduction FMV was:

```

from 0x2dce ... to 0x3000 len 0x232
from 0x5bee ... to 0x6000 len 0x412
from 0x8dce ... to 0x9000 len 0x232
from 0xbbee ... to 0xc000 len 0x412
from 0xebce ... to 0xef6a len 0x37c
from 0x11dce ... to 0x12000 len 0x232
from 0x14bee ... to 0x15000 len 0x412
from 0x17bee ... to 0x17f46 len 0x358
from 0x1abee ... to 0x1af22 len 0x334
from 0x1dbce ... to 0x1df22 len 0x334
from 0x20bee ... to 0x20f22 len 0x334
from 0x23bee ... to 0x23f22 len 0x334
from 0x26bee ... to 0x26f22 len 0x334
...

```

The locations output above are relative to sector \$f20 on the CD-ROM. In other words, the FMV's audio data was located piecemeal in increasing locations on the CD-ROM. What does this mean? First, finding the audio data validated our assumptions. Second, it helped decode part of a header⁶ we found preceding the static base image data. Third, it indicated the structure of the FMV data on the CD-ROM.

While detailed discussion of the audio data encoding is left to Section 5, for now it is sufficient to know that each audio sample takes 4 bits, i.e., each (8-bit) byte contains two samples. The typical \$334 length seen above, 820 in base 10, corresponds to 1640 audio samples which at 16KHz is just slightly over 1/10s. There is one of these situated every \$3000 bytes in the FMV data, or every 12288 bytes, and that divides evenly by 2048 bytes/sector, revealing that 6 sectors need to be read every 1/10s to maintain the audio stream.

This shows us the FMV frame rate, the speed at which the game displays still images to give the illusion of motion, measured in frames per second (FPS). Assuming one video frames' worth of data is contained in each 6-sector segment, 10 video frames/second $\times$ 6 sectors/video frame $\times$ 2352 bytes/raw CD-ROM sector⁷ equals 141,120 bytes, approximately 137.8KiB. If we increase that even by one more video frame per second, we get just over 151KiB. The reason this is important is that, for a 1 $\times$ CD-ROM drive like the PCE had, the maximum sustained transfer rate is 150KiB/s. As this is a maximum rate under ideal conditions, without any CD-ROM optical head movement, some slack needs to be allowed for head movement latency from track to track. 10 FPS is therefore the likeliest rate unless multiple video frames are contained within the 6-sector segments; the answer lies in the FMV video format.⁸ Before examining that, however, we wrote a Python script that extracted the audio data directly from the CD-ROM sectors for an FMV clip and output it as a WAV audio file, allowing us to compare to the in-game FMV audio and confirm our analysis.

Given all this information, and a probable rate of 10FPS, we suspected that the "static base image" was not a critical key frame after all, merely the first image in the FMV sequence. That is, each 6-sector segment might contain data for exactly one uncompressed image to show for that video frame. The problem was how to test that hypothesis, because we had not yet located the color palette information detailing the RGB values of the colors in use, if indeed it was located with the FMV – for all we knew, there was one palette for all the FMV videos that was loaded into the video color encoder (VCE) memory at some point early in the game. Without that, we would be unable to render the FMV video in its full glory, and might not be able to see any coherent images at all.

Instead, we turned to plotting differences. Without color information, we would not know exactly what color each pixel on the screen was, but we *could* know that a pixel's palette index had changed from the previous frame to the current frame, and if we assume that a change in the palette index value means that the color of the pixel changed (not necessarily a correct assumption), then we could at least show a rendition of the FMV video sequence indicating these differences. Our past work (Aycock, 2016) had shown that FMV differences could be clearly correlated with the actual video and, if our hypothesis about the video frame data were incorrect (say that it were encoded or compressed in some different way than we expected) then the result we would see would simply be unintelligible noise.

We wrote a Python script to extract the FMV video data directly from the CD-ROM sectors based on our hypothesis, rendering the differences between pairs of frames as GIF images that we combined into an animated GIF at 10FPS; the animated GIF effectively created a video from the sequence of frame-difference images. If correct, we should be able to play the animated GIF side-by-side with an FMV clip running in-emulator and clearly see the same activity taking place in the two videos. This is in fact what we saw. Figure 2 shows a difference frame from Holmes' Introduction using this process, and a thumbnail of the

⁶ A header is a block of meta-data that precedes the actual data. A header might, for example, store information about the length of the data that follows.

⁷ As mentioned earlier, while the data in each CD-ROM sector is 2048 bytes, it is accompanied by extra information that increases the actual sector size to 2352 bytes. Since this extra information must be read by the CD-ROM drive, we need to use the larger raw sector size in our calculation here.

⁸ This can also be examined experimentally. Enabling Mednafen's FPS display, we see it shows the emulator running at 60FPS. Stepping through one of SH's FMV sequences frame by frame in Mednafen, we would therefore expect to see the FMV images change every 6 emulated frames if the FMV were running at 10FPS, and this is in fact what we see.

Figure 2. Plotting differences in pixel palette indices.

original frame for reference; pixels with differences are shown in red. The differences shown are actually using the lower 4 bits of the palette index only, because the palette bits from BAT entries were not taken into account. Also, the SH video was very noisy even watching it in the emulator, so this script is computing pixel differences over a sliding window of the last 20 video frames to help reduce the noise to some degree. At this point, we knew the FMV video and audio format, *sans* color information, and we moved on to analyze GB.

For GB, we began the same way, by logging the SCSI accesses during FMV, immediately noticing the familiar repeated \$fc-length requests endemic to SH's FMV:

```
SCSI: Read: start=0x0002ef4b(track=2, offs=0x0002d85c), cnt=0x000000fc
SCSI: Read: start=0x0002f047(track=2, offs=0x0002d958), cnt=0x000000fc
SCSI: Read: start=0x0002f143(track=2, offs=0x0002da54), cnt=0x000000fc
SCSI: Read: start=0x0002f23f(track=2, offs=0x0002db50), cnt=0x000000fc
SCSI: Read: start=0x0002f33b(track=2, offs=0x0002dc4c), cnt=0x000000fc
...
```

From Mednafen's debugger, we could see that the FMV audio was, like SH, using the built-in Oki chip with 16KHz sampling frequency. Assuming we would see the same audio encoding, we took the same approach. We re-enabled our audio data-dumping code in Mednafen and re-used the script we wrote to find SH's audio to find *incredibly* similar results:

```
from 0x2cd0 ... to 0x2fc8 len 0x2f8
from 0x5ca8 ... to 0x5fc8 len 0x320
from 0x8ca8 ... to 0x8fc8 len 0x320
from 0xbca8 ... to 0xbfc8 len 0x320
from 0xeca8 ... to 0xefc8 len 0x320
...
```

This implies that GB's HuVideo also has 6-sector chunks for FMV along with the same frame rate (\$320 is 800 in base 10, or 1600 samples, making those audio portions exactly 1/10s long). With very minor changes to our Python script to extract FMV audio data for SH, we were able to perform the identical feat for GB. Furthermore, on a whim, we made minor modifications to the script we had written to plot FMV video differences for SH (compensating for HuVideo's apparent lack of header data), and it worked for GB as well.

To discover how truly similar the two games' FMV formats were, we needed to find and decode the final piece: the color information. We returned to the real-time BAT visualization we added to Mednafen before.

First, we note the comparative sizes and FMV behavior. GB's screen is also 64×64 tiles, with an FMV window size of 24×14 tiles, or 192×112 pixels. GB also uses double buffering, although in a different manner. It has, effectively, two FMV windows in its BAT whose tile pointer values never change; each FMV window's pointers point to tile data in distinct VDC memory regions. The two FMV windows are flipped by toggling the VDC's X scroll register between two different values, which can be thought of as instantaneously changing the origin point of the display.

Pausing the emulator during FMV, we recorded the (4-bit) palette values in the BAT entries, and manually searched for likely encodings of that data throughout the GB CD-ROM image. Finding its location within a 6-sector FMV segment, we now had three of the four puzzle pieces: tile data, audio data, and BAT palette data. Doing some arithmetic on their sizes and locations within a 6-sector segment, we identified a 512-byte gap that was large enough to store 256 colors' worth of palette data and, moreover, was the only space remaining in the segment large enough to do so. We wrote a Python script to extract FMV video directly from the CD-ROM image and render it in full color as an animated GIF, confirming our ideas about the structure of HuVideo-format data on the CD-ROM.

Additionally, since we now had full color information, we wrote a script to try plotting frame-to-frame differences again, but this time using a different, more precise method that would also show the magnitude of the differences. Now, we treated each pixel's RGB value as an (x,y,z) coordinate in 3D space, and calculated the frame-to-frame difference of a pixel as the Euclidean distance between a pixel's 3D points in each frame. The intuition is that colors that are closer together will have shorter distances between them, and identical colors will be zero distance apart. Our script mapped those differences into a grayscale image (with up to 256 shades of gray), where a pixel became lighter the more it changed by this metric. This technique produced excellent results for GB's anime, as Figure 3 shows along with thumbnails of the original two frames.

Figure 3. Plotting differences in pixel palette indices, as greyscale.

The only remaining task was to find and decode color information for SH. Taking the same tack, we gathered a set of BAT palette values from a paused Mednafen and our real-time BAT display, and manually searched for the data in probable ways that it might have been encoded. When found, we were then able to locate the palette data, and draw from the existing Python scripts to write a new one that extracted the FMV video from SH’s CD-ROM image in full color as an animated GIF. We did modify the GB grayscale script to try grayscale difference plotting for SH, but the video noise there was again prevalent and the results not nearly as good.

5 Results

The SH FMV format is shown in Table 1. All the sectors are laid out consecutively, which minimizes track-to-track movement. The audio-only data is slightly over 2s of leading audio for the FMV clip (8 sectors × 2048 bytes/sector × 2 samples/byte ÷ 16,000 samples/second). There is a curious CD-ROM access pattern at the start of an SH FMV clip: the first 6-sector FMV segment is read to load the first FMV image that the curtain-esque wipe opens into; the 8 sectors of audio data are read; the 6-sector FMV segments are read in sequence, re-reading the first 6-sector FMV segment. This can be seen, for instance, in the SCSI log excerpt for Holmes’ Introduction given earlier in this report. We did not discover an obvious value that indicated how many 6-sector segments there were or, alternatively, an indicator at the end of the FMV data – we may either have overlooked it, or the lengths are embedded internally in the game code.

Table 1. SH FMV format.

8 sectors audio-only data
6-sector FMV segment
6-sector FMV segment
6-sector FMV segment
...
6-sector FMV segment

Each 6-sector FMV segment for SH has the structure given in Table 2. Depending on the amount of data in each segment, there can be a few hundred extra bytes left at the end (the amount of audio data in each segment, in particular, is not necessarily the same for each segment).

Table 2. SH FMV segment format.

26-byte header
10,560 bytes tile data (32 bytes per tile, 330 tiles)
660 bytes BAT entries (16 bits each, 330 tiles)
Optional: 480 bytes palette data (240 colors, 16 bits each)
Audio data

The presence or absence of the palette data can be determined by examining where the audio data starts, as given in the segment header. The tile data, BAT entries, and palette data are uncompressed and are in the format needed by the VDC and VCE (Archaic Pixels, n.d.-a; Archaic Pixels, n.d.-b). This includes, rather pointlessly, the BAT pointers too.

The known fields in the 26-byte FMV segment header are in Table 3, with the location of each being an offset from the start of the 6-sector segment. There are some nonzero values in the header whose purpose is unknown, which is not an unusual occurrence when reverse engineering, and the exact length and

endianness of fields marked with a dagger is unclear because there are no example FMV clips exceeding one-byte values for them. Endianness refers to the order in which a number is stored in consecutive bytes; for instance, the value \$1234 may be stored as the two bytes \$34 \$12 (little-endian) or as the two bytes \$12 \$34 (big-endian).

The invariant fields (or, at least, invariant for SH) can be used as a signature to locate FMV clips throughout the CD-ROM. We wrote a Python script to do that, and the 87 clips it identified are stored in a data repository (Aycok, 2018).

For GB, the FMV format is given in Table 4. Again, all the sectors are laid out consecutively to minimize track-to-track movement. The header information does not come close to occupying an entire sector, and the rest of the sector is zeros. There may be some feature of HuVideo that makes use of that space and the three all-zero sectors that follow it, but we did not find any examples in GB.

A 6-sector FMV segment in GB's HuVideo format is arranged as shown in Table 5. There are 56 extra, unknown, possibly unused bytes at the end of each segment.

Table 3. SH FMV segment header fields.

Location	Data type	Purpose
\$00–\$01	16-bit unsigned integer, big-endian	Length of audio data in this segment, in bytes
\$04–\$05	16-bit unsigned integer, big-endian	Offset to audio data in this segment, minus \$2ae
\$06–\$07	16-bit unsigned integer, big-endian†	FMV horizontal width, in pixels
\$08–\$09	16-bit unsigned integer, big-endian†	FMV vertical height, in pixels
\$0a–\$0b	16-bit unsigned integer, big-endian†	Tile height or width?
\$0c–\$0d	16-bit unsigned integer, big-endian†	Tile height or width?
\$0e–\$0f	16-bit unsigned integer, big-endian†	FMV horizontal width, in tiles
\$10–\$11	16-bit unsigned integer, big-endian†	FMV vertical height, in tiles

Table 4. GB (HuVideo) FMV format.

1 sector HuVideo header
3 all-zero sectors
6-sector FMV segment
6-sector FMV segment
6-sector FMV segment
...
6-sector FMV segment

Table 5. GB (HuVideo) FMV segment format.

10,752 bytes tile data (32 bytes per tile, 336 tiles)
512 bytes palette data (256 colors, 16 bits each)
168 bytes BAT palette data (4 bits each, 336 tiles)
800 bytes audio data (1/10s)

The BAT palette data (recall this is 4 bits per BAT entry) is packed with two BAT entries' palette data per byte. As before, tile data and palette data are in the format the VDC and VCE require.

The known fields in the HuVideo header are listed in Table 6. There are some nonzero values in the header whose purpose is unknown. We wrote a Python script to scan the CD-ROM for the distinctive HuVideo signature and found 38 FMV sequences in total, detailed in our data (Aycok, 2018).

Table 6. GB (HuVideo) FMV header fields.

Location	Data type	Purpose
\$00-\$06	ASCII characters	Identifying string <code>HuVideo</code>
\$10-\$11	16-bit unsigned integer, little-endian	FMV length in segments
\$12-\$13	16-bit unsigned integer, little-endian	FMV horizontal width, in pixels
\$14-\$15	16-bit unsigned integer, little-endian	FMV vertical height, in pixels

Since we are interested in the FMV format, and GB uses HuVideo format, one way to validate our findings for GB is to attempt FMV audio/video extraction for another HuVideo game. We did this using the game *Ginga Ojōsama Densetsu Yuna* (1992) that also advertises HuVideo (Mobygames, n.d.-c).⁹ We were able to find HuVideo clips using our GB signature-scanning script, and the only difference was that they were 160×128 pixels. With a minor adjustment to compensate for the different FMV size (the amount of tile data and BAT palette data changes accordingly), the FMV audio and video extraction scripts for GB worked unchanged.

The audio format used by both SH and GB is adaptive differential pulse code modulation (ADPCM), which is directly supported by the Oki MSM5205 chip in the CDRom² (Archaic Pixels, n.d.-c; Dialogic, 1988; Oki, n.d.). A 4-bit encoded ADPCM sample (hence two per byte) is decoded to a 12-bit pulse code modulation (PCM) sample, 10 bits of which are used to generate audio. Conceptually, the steps from an original analog audio signal to ADPCM are as follows.¹⁰ An analog audio signal, whose amplitude is sampled at regular intervals, is PCM data; this is ultimately what needs to be reconstructed by the Oki chip. An audio waveform is relatively well behaved in the sense that the value of sample $n + 1$ is not typically that different from the value of sample n . Therefore, space can be saved by storing audio data as the differences between samples, yielding differential PCM (DPCM). ADPCM goes one slight step further by shifting between different ranges of deltas that can be applied. In effect, the four ADPCM bits of a sample encode both the delta amount to be applied to get sample $n + 1$ from sample n , as well as the set of deltas to be used for decoding sample $n + 2$.

In terms of the limitations of our analysis, we did not compare all FMV clips in the two games to the audio and video that our scripts extracted directly from the CD-ROM images. Having said that, we have no reason to believe that format variations exist within the games that would differ from those documented here.

Despite the years separating the two games, the FMV format did not substantially differ between them. The most serious data compression only occurred for sampled audio, and even this was more a side effect of built-in hardware support; the FMV formats were instead designed more for ease of cramming data into the PCE's graphics memory. This was possible given a willingness to use a relatively low frame rate and a limited onscreen video size, but increasing beyond that in either respect would almost certainly have necessitated a change in format (cf. 1993's *The 7th Guest* for the PC; Aycok, 2016). This might be feasible for the drawn images of GB, although the noisy video present in SH yields many frame-to-frame changes that could present challenges for some compression schemes. It seems safe to say, given encoding and compression opportunities that were *not* leveraged in SH and GB, that CD-ROM capacity in these games was not at a premium compared to other constraints.

6 Discussion and Conclusion

We have presented here a scientific approach to the understanding of a small part of artifactual examples of late 20th-century digital material culture, an approach applied with archaeological rigor. While our exact methods are drawn from computer science and are novel to archaeology, archaeology has a longstanding history of incorporating and embracing analytical methods from other disciplines.

⁹ The image was labeled `Ginga_Ojousama_Densetsu_Yuna_-_HuVideo_CD_(NTSC-J)_[HCD5078].iso` length 478,370,928 bytes, MD5 `$623d029444f14d6c71eb2863c5609c76`.

¹⁰ Some liberties are being taken in this description for clarity.

It would be both brilliant and foresightful to be able to claim that this work was undertaken in the context of an established archaeological framework. We cannot; the reverse engineering was performed strictly from the computer science perspective, and it was only during the course of investigation that parallels to archaeological inquiry were identified. It seemed logical to integrate our findings within the nascent archaeology of digital things. We argue that the potential for this work – and work like it – to assist in creating a new framework, or updating an existing one, should be explored further. Digital artifacts such as the ones investigated in this paper are examples of material culture, created by developers and then used (even cherished) by many people. The creation process utilized a combination of hardware and software technologies that were also used for the creation and manufacture of other digital artifacts. These CD-ROMs and the files and file formats are themselves archaeological artifacts, and are also archaeological residues of the earliest days of graphical interactive digital entertainment.

Unlike the archaeology of ancient things, archaeologists of the digital have the opportunity to conduct their investigations closer to the time in which the source material was created. This temporal proximity is undoubtedly an advantage, that the computer science aspects could be approached with some understanding of the materials being studied and the suitable tools to use, something that might be close to impossible for someone without years of experience in a given technical environment. The archaeologist author of this paper was admittedly lost even though he grew up with computers and games in the 1970s and 1980s, and can write computer programs. We re-emphasize Moshenska (2014): for the archaeology of digital things, computer scientists will prove to be indispensable for current and future investigations, yet archaeology must provide a framework to surround the research questions being asked.

This new framework can draw from existing post-processual and object-oriented frameworks, merging those with methods and tools deployed by computer scientists. With its genesis in computer science, the source material (software programs and the media to house and play them) and the questions raised at the beginning of this article would seem to lend themselves to a processualist (postivist) approach. It is not difficult to imagine that because we are dealing with computers that the data produced in this study provides empirical evidence, and that evidence drives our conclusions. However, viewing this through an archaeological lens, it was immediately clear that the computer science came bundled with a disciplinary bias. Recognizing this bias implies that reflexivity in digital archaeology can be present in the computer science-based analysis of digital artifacts, but the question remains of what framework to use.

We could follow the post-processual approach of Ian Hodder, Michael Shanks, Christopher Tilley, and others, realizing that the archaeological evidence being analyzed was not just borne of material evidence, but rather was a product of a culture of development driving decision-making by game-creators and manufacturers of hardware and software. In effect, the cause of the differences and adoption of one digital standard over another was driven and resolved by conflict, something which Foucault in *The Archaeology of Knowledge* (1969/2012) thought was the driver of any innovation.

Moving away from human agency through both software developers and through the people conducting this project, one could potentially couch future archaeology of digital things within the framework of object-oriented ontology (not to be confused with object-oriented programming). Object-oriented ontology decentralizes the human role in object creation and use and attempts to understand things as they are, entities separate from one another until activated by an outside force.

We suggest that a future framework for the archaeology of digital things must be an inclusive one, taking parts of positivism, post-processualism, and object-oriented ontology paired with existing computer science tools and methods for reverse-engineering artifacts. This needs to be tempered by mindfulness of the investigators towards a diversity of digital subjects while evaluating the best way to approach the research questions (Huvila & Huggett, 2018). Add to that the necessary critique of digital tools and their resulting output (Huggett, 2015; Smith, 2018), as well as the ethics behind an investigation of the intellectual property of others. This will result in a more robust theoretical setting in which digital archaeologists – which include computer scientists – can conduct their fieldwork. The media in which the archaeologists of the digital operate lends itself to archaeological questions that contribute to the existing knowledge of human and non-human interaction with things and the material culture these things help produce. An understanding of archaeological theory paired with expert knowledge in how computer technologies work

can yield detailed, repeatable results that not only document a history of human-digital creation, but also in the ways that the digital influenced (and continues to influence) human creators and communities of users.

Acknowledgments: The first author's research is supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). NSERC has otherwise had no role in this work or its publication.

This analysis problem was originally suggested in discussions with Martin Picard. Thanks to Jeff Boyd for helpful discussion on noise filtering, and Ryan Henry for the postal mail analogy. The editor's and anonymous reviewers' comments greatly helped improve this paper.

Abbreviations

ADPCM	Adaptive differential pulse code modulation
ASCII	An encoding for characters
BAT	Block attribute table in VDC
CPU	Central processing unit (HuC6280)
FMV	Full-motion video
FPS	(Video) Frames per second
GB	<i>Kūsō Kagaku Sekai Gulliver Boy</i>
GIF	Graphics Interchange Format, widely used for bitmap images
KiB	Kibibyte, or 1024 bytes
MD5	Algorithm for computing checksums
NEC	Japanese company (originally Nippon Electric Company)
PCE	PC Engine/TurboGrafx-16
PCM	Pulse code modulation
pixel	Picture element, the smallest display unit on a screen
RAM	Random-access memory
RGB	Color representation based on combination of red, green, and blue
ROM	Read-only memory
SCSI	An interface between computers and peripherals (e.g., CD-ROM drive)
SH	<i>Sherlock Holmes: Consulting Detective</i>
VCE	Video color encoder (HuC6260)
VDC	Video display controller (HuC6270)
WAV	A common audio file format

References

- Archaic Pixels (n.d.-a). *HuC6270*. Available at <http://archaicpixels.com/index.php?title=HuC6270&oldid=13653> [Last accessed 23 November 2018].
- Archaic Pixels (n.d.-b). *HuC6260*. Available at <http://archaicpixels.com/index.php?title=HuC6260&oldid=13654> [Last accessed 23 November 2018].
- Archaic Pixels (n.d.-c). *MSM5205*. Available at <http://archaicpixels.com/index.php?title=MSM5205&oldid=13221> [Last accessed 23 November 2018].
- Arnold, K.C.R.C. (c. 1980). *Screen updating and cursor motion optimization: A library package*. 4.2BSD documentation.
- Aycock, J. (2016). *Retrogame Archeology: Exploring Old Computer Games*. New York: Springer.
- Aycock, J. (2018). "Automatically identified full-motion video clips", <https://doi.org/10.5683/SP2/OX8LI8>, Scholars Portal Dataverse, V1, UNF:6:eKbcVW6S0K0qM43HaRfUqg==
- Browne, C., Soemers, D. J. N. J., Piette, É., Stephenson, M., Conrad, M., Crist, W., ... Winands, M. H. M. (2019). Foundations of Digital Archaeoludology. ArXiv:1905.13516 [Cs]. Retrieved from <http://arxiv.org/abs/1905.13516>
- Buchli, V. & Lucas, G. (2001). The absent present: Archaeologies of the contemporary past. In V. Buchli & G. Lucas (Eds.), *Archaeologies of the Contemporary Past* (pp. 3–18). London: Routledge.

- Dialogic Corporation. (1988). *Dialogic ADPCM Algorithm. Application note*.
- Elsaesser, T. (2016). Media archaeology as symptom. *New Review of Film and Television Studies*, 14(2), 181–215.
- Foucault, M. (1969/2012). *The Archaeology of Knowledge*. A. M. Sheridan Smith (trans.). New York: Vintage.
- Hodder, I. (2001). Epilogue. In V. Buchli & G. Lucas (Eds.), *Archaeologies of the Contemporary Past* (pp. 189–191). London: Routledge.
- Huggett, J. (2015). A Manifesto for an Introspective Digital Archaeology. *Open Archaeology*, 1(1), 86–95. DOI: <https://doi.org/10.1515/opar-2015-0002>
- Huhtamo, E. & Parikka, J. (2011). Introduction: An archaeology of media archaeology. In E. Huhtamo & J. Parikka (Eds.), *Media Archaeology: Approaches, Applications, and Implications* (pp. 1–21). Berkeley: University of California Press.
- Huvila, I. & Huggett, J. (2018). Archaeological Practices, Knowledge Work and Digitalisation. *Journal of Computer Applications in Archaeology*, 1(1): 88–100. DOI: <http://doi.org/10.5334/jcaa.6>
- Internet Archive (n.d.). *About the Internet Archive*. Retrieved June 8, 2019, from <https://archive.org/about>
- Madhav, S. (2014). *Game Programming Algorithms and Techniques*. Boston: Addison Wesley.
- MobyGames (n.d.-a). *Kūsō Kagaku Sekai Gulliver Boy releases*. Available at http://www.mobygames.com/game/turbografx-cd/ks-kagaku-sekai-gulliver-boy_/release-info [Last accessed 23 November 2018].
- MobyGames (n.d.-b). *Sherlock Holmes: Consulting detective releases*. Available at <http://www.mobygames.com/game/sherlock-holmes-consulting-detective/release-info> [Last accessed 23 November 2018].
- MobyGames (n.d.-c). *Ginga Ojōsama Densetsu Yuna*. Available at <http://www.mobygames.com/game/ginga-ojsama-densetsu-yuna> [Last accessed 23 November 2018].
- Moshenska, G. (2014). The Archaeology of (Flash) Memory. *Post-Medieval Archaeology*, 48(1), 255–259.
- Moshenska, G. (2016). Reverse engineering and the archaeology of the modern world. *Forum Kritische Archäologie*, 5, 16–28.
- Newman, J. (2012). *Best Before: Videogames, Supersession, and Obsolescence*. New York: Routledge.
- Oki Semiconductor (n.d.). *MSM5205 ADPCM speech synthesis LSI. Data sheet*.
- Perry, S. & Morgan, C. (2015). Materializing Media Archaeologies: The MAD-P hard Drive Excavation. *Journal of Contemporary Archaeology*, 2(1), 94–104.
- Reinhard, A. (2018). *Archaeogaming: An Introduction to the Archaeology in and of Video Games*. New York: Berghahn.
- Smith, G. J. D. (2018). Data Doxa: The Affective Consequences of Data Practices. *Big Data & Society*, 2018.1, 1–15.