Programming the CD-I Player

Ernest W. Adams

Optical media machines -- CDTV, CD-I, the Multimedia PC and so on -- have been the subject of ever-increasing debate in the computer game industry lately, but most of the emphasis has been on game design and marketing issues. There has been fairly little discussion of the question, "What are these machines like to program?" I want to answer that question for one of the new optical machines on the market, the CD-I player. I've been programming the CD-I player for the last year, working on a multi-player game for adults called *Third Degree.* I'll be describing its anatomy and physiology, and what the development environment for CD-I is like.

The CD-I Standard

CD-I is a publicly available worldwide standard, originally developed jointly by Philips and Sony. Anyone who wants to can get the standards documents and build a CD-I player. Most major electronics companies have plans to build CD-I players.

The standard defines a "base case", which *all* players must meet in order to be allowed to call themselves CD-I or Compact Disc-Interactive. All titles must work in some fashion on the base case, although they may present more features on extended machines -- just as all audio CD's must play in any CD player.

It is important to realize that the CD-I standard is a functional specification and not a hardware or design specification. Consequently there is no requirement on the manufacturers to use any given hardware or software, as long as the demonstrated behavior of the machine meets the functional specification. The standard does not prohibit any extensions, but it does also provide some "recommended extensions" which give a uniform upgrade path.

The CD-I Full Functional Specification, the bible of CD-I, is informally known as the Green Book. (Normal audio CD's, called CD-Digital Audio or CD-DA, are called Red.) The Green Book is an essential document for any programmer's library; at this stage there are very few third-party sources of information about the player. When I talk about the hardware, I will be referring to the Philips industrial model CD-I player, called the 180. The consumer model, the 910, is functionally similar, although it looks quite different.

Anatomy: The Hardware

A CD-I player consists of a 68000 series microprocessor, 1 meg (base case) of RAM, 8K of non-volatile RAM, certain specialized audio and video hardware, a CD drive, and a joystick-like pointing device with two buttons. In the Philips versions, a 68070 is used, running at 8 Mhz.

Standard extensions include: more RAM, SCSI ports, floppy drives, serial I/O ports, a mouse, and a keyboard.

The Compact Disc

CDTV, the Amiga-based optical media product from Commodore, is substantially different from CD-I. It is, essentially, a CD-ROM Amiga with no keyboard or disk drives. With CD-I, the CD can also be treated as a CD-ROM -- basically a large filesystem. However, CD-I also includes a special concept called the Real-Time File

(RTF). These files are intended to be "played" in real time, rather than "read" the way ordinary computer files are. RTF's are guaranteed to occupy sequential sectors on the disc, and to always deliver data at 75 sectors per second. Most importantly of all, they are played asynchronously, so the CD drive can be filling up memory buffers without the CPU doing any work. The play process only interrupts the CPU when a buffer gets full, or when the end-of-play condition is met.

Non-realtime files may be thought of as data files like any other computer file. They are stored in 2048-byte error-corrected sectors and can be read by normal C I/O calls. RTFs contain two kinds of sectors: 2048-byte error-corrected sectors (for critical data like text and program data) and 2324-byte sectors in which errors may be detected but not corrected (for noncritical data like audio and video data -- once detected by the operating system, these errors may be"hidden" by replacing the offending pixel or sound sample with an average of the ones around it). These sectors are larger because they do not include the error-correction codes. (All this emphasis on error-handling is necessary because consumers handle CD's with their bare hands. CD-I titles are required to handle dirty discs gracefully.)

Each sector in the RTF may be assigned a "channel number" (nothing to do with audio channels) for the convenience of the programmer, from 0 to 31. Each sector is also flagged with a type: Audio, Video, or Data. Playing data into memory from the disc requires creating a data structure called a Play Control List (PCL), then making a system call to start the play. The PCL tells the system 1) which channels to look for (sectors which don't have that channel number are ignored), 2) which sector types to look for (sectors of the wrong type, even if they are in the right channel, are ignored), 3) the addresses and sizes of the buffers which will receive the data, 4) which function to call when a buffer gets full, and 5) under what conditions play should terminate.

Audio Hardware

On a normal CD-DA disc, every sector is taken up with 16-bit stereo audio data, recorded at a sampling rate of 44.1 kilohertz. These are known as red sectors, and are stored in sequences called red tracks. This level of audio quality requires that every sector contain audio information.

CD-I audio takes up less space than normal CD-DA audio. This is accomplished by employing one of six lossy compression levels (actually, 3 levels called A, B, and C, but each can be mono or stereo) which provide increasingly lower quality. The lowest quality, called C mono, is about as good as an AM radio, and provides 16:1 compression (i.e. 15/16ths of disk is free for other things). Normal CD-DA is stored in what is called Pulse-Coded Modulation (PCM); CD-I sound is stored as Adaptive Delta Pulse-Coded Modulation (ADPCM). The details of the compression are too long to go into here.

In a Real-Time File, the compressed audio is not stored all in one place, but is evenly spaced on the disc to arrive under the playback head exactly as often as it is needed. In A stereo, which provides 2:1 compression, every other sector contains audio data. In C mono, every 16th sector is audio. *The intervening sectors are empty and can be filled with other information.* This is the most powerful and important feature of CD-I: the ability to interleave audio and video sectors so that the data is read in exactly when it is needed. In addition, the channel mechanism, which loads only the sectors in the channels you specify, enables you to interleave multiple audio tracks. This provides two benefits. Firstly, you may store far more audio on a CD-I disc than on a CD-DA disc if you are willing to sacrifice quality; up to 16 hours, if stored in C mono. Secondly, you may present multiple versions of basically the same audio, which prevents games from sounding repetitve.

CD-I programs generally present their information by playing Real-Time Files and delivering the audio to the speakers and the video to memory. The audio hardware permits audio to be played directly from the disc to speakers, or from disc to memory and then from memory to speakers. It also permits the volume to be

attenuated from software. There are four attenuators: left-to-left, left-to-right, right-to-left, and right-to-right. Audio in memory can be manipulated mathematically and even mixed before it is played, although this is a CPU-intensive process and may not be able to keep up with the demand.

Video Hardware

The video hardware is complex and powerful, although the native resolution of the screen is fairly poor. NTSC (American 525-line TV's) display a screen of 384 pixels horizontally and 240 pixels vertically); PAL (European 625-line TV's) display 384 pixels horizontally and 280 pixels vertically.

There is also a video mode called "double-resolution" with twice the number of pixels horizontally, and you may turn interlace on and use twice the number of pixels vertically as well. In practical terms, however, this requires more data than is usually managable, and is rarely used. The Green Book also provides for a 768 × 560 non-interlaced mode called "high resolution" in the extended case; again, this is a lot of data and all titles must run in the base case anyway.

The hardware provides two video planes, one in front of the other; a special rear plane which will be used for full-motion video (FMV) when it becomes available; and a cursor plane at the very front which is 16 × 16 and is normally used for displaying the cursor.

There is no video RAM set aside; it is all shared with the program RAM.

The video processor runs a special program called the Display Control Program once per screen refresh, or 30 times a second. The DCP is established by the application which is currently running. It executes a table of instructions once at the beginning of each field (the Field Control Table), and more instructions at the beginning of each line as the beam scans the screen (the Line Control Table). The DCP provides instructions to the video processor on a wide variety of things such as:

- The address of the data for the pixels on the current line (the Line Start Address).

- Mattes, which permit "holes" in the front plane which look through to the rear plane.

- Chroma-key transparency data -- all pixels on the front plane of a given color are transparent, permitting the viewer to see through to the rear plane (this is different from matting).

- Image contribution factor: Each plane can contribute a certain percentage to the final image, permitting fade-up, fade-down, and cross-fade effects.

- Palette information.

- Plane order. The video planes can be swapped from front to back very easily, permitting fast cuts from one to the next.

- Image coding information (described below).

The data used to display any given line must be contiguous in memory, but from line to line data can be anywhere, controlled by the Line Start Addresses mentioned above.

Image data on any given line can be encoded in one of 8 different methods, each of which is handled automatically by the video processor. They are:

- RGB555 (5 bits of Red, Green, and Blue data, or two bytes per pixel). This provides the highest color accuracy, but only one video plane is available in this mode. Very rarely used.

- DYUV (Delta YUV compression, 1 byte per pixel). This is a lossy form of image compression. YUV is a color television term, not a computer term; Y refers to luminance or brightness information and UV refer to chrominance (hue and saturation, or "tint" and "color" on your TV). There is simple mathematical formula for converting YUV to RGB and back again.

  In DYUV mode the Y, U, and V data is delta-compressed into 4-bit values. However, the U and V deltas are only stored for every pixel PAIR, not every pixel, because the human eye is less discriminating of chrominance than it is of luminance. The Y delta is stored for every pixel. DYUV is well suited to continuous-tone pictures like photographs, but because it only stores 4 bits of delta data for Y, U, and V, it cannot show regions of high contrast well.

- CLUT8 (8-bit Color Lookup Table, 1 byte per pixel). The video hardware has 256 24-bit palette, or CLUT registers; this mode uses all of them. If one plane is CLUT8, the other plane cannot be in a CLUT mode because all the registers are in use. This most closely corresponds to IBM VGA data. The data is pixel-packed, not planar.

- CLUT7 (7-bit CLUT, 1 byte per pixel). Like CLUT8, but the CLUT is divided into two, half for each video plane. Each plane can show 128 colors.

- CLUT3 and CLUT4 (8 and 16 colors respectively, 1 nibble per pixel).

- RL7 (CLUT7 data which is then run-length encoded). The run-length decoding is done by the video hardware.The extra bit left over in the byte is used to flag whether a run is starting or not.

- RL3 (CLUT3 data which is run-length encoded).

The coding method can change on each line of the screen if there is an instruction in the DCP to change it.

Work is currently in progress on developing a full-motion video chip which will do MPEG decompression of video data and display it on the special, rearmost plane which is currently unused. This will probably be available in players a year from now or so.

Note that unlike the IBM, there is no text mode. Unlike the Mac, there is also no windowing system, etc. Everything must be done with the DCP and RAM buffers.

NVRAM

The non-volatile RAM is backed up with a lithium battery. It is treated as a very small filesystem.

Physiology: The Software

CD-RTOS

The CD-I player runs an operating system called CD-RTOS. CD-RTOS is an extension of the operating system OS-9 68000, which is a port of the OS-9 operating system from the 6809, for which it was designed, to the 68000. Generally, any software written for OS-9 will run under CD-RTOS, and any tools designed for OS-9 development can be used for CD-I development as well.

CD-RTOS has a number of features that OS-9 does not have, having to do with the special CD-I hardware. It has system calls to create and manipulate the Play Control List, the Display Control Program, and the audio processor, for example. The hardware *must* be manipulated through system calls, because the CD-I standard is a functional standard and not a hardware standard.

CD-RTOS is fairly small, requiring only 32K in high RAM and 32K in low RAM. The rest is contained in ROM, which doesn't count against the 1 Meg total RAM. It is also quite featureful: completely mutitasking, with pre-emptive task switching; signals; events; exception processing; interprocess communications, and so on. On the other hand it is fairly slow, especially some of the more complex video drawing routines.

Also, because there is no standard hardware configuration, you cannot directly grab a hardware interrupt -- you have to wait for the CD-RTOS signal, which means there is always a latency of *unknown duration* between getting the interrupt and dealing with it.

CD-RTOS also supports a fairly primitive font methodology, since there are no text modes for the video display. Normal C terminal I/O functions go to the serial port, if one exists -- it is assumed that no consumer title would ever use standard input or standard output.

Real-Time File development

Developing the Real-Time Files for a CD-I product is every bit as important, and nearly as time-consuming, as developing the code. Almost any audio and video to be presented should be delivered through an RTF, and this means thinking carefully about how the data needs to be laid out. The lower quality audio you use, the more intervening sectors you have to store video and other information. Since RTF's are played in real time, you will need to decide exactly when you need certain assets delivered into memory. For example, often some sort of screen change accompanies an audio prompt; therefore you might wish to place the new screen right before the start of the audio. That way the video will arrive in memory and the audio will start playing only one sector (one 75th of a second) apart.

Most video objects read in from an RTF will need to be double-buffered. The sectors arrive sufficiently slowly from the disc that if you write them directly into your displayed screen buffer, you can see the data being loaded as a line moving down the screen. In addition, if the video is interleaved with audio data, there may be sudden pauses while an audio sector is played. It is better to load video into an offscreen buffer, then display the buffer. Since any memory may be treated as video memory, it is not necessary to copy the buffer; it can be linked directly into the Display Control Program once it is full.

Generally it is not possible to have art staff design the Real-Time Files. Because RTF's are played by having the program create a Play Control List, then make a CD-RTOS call, their design is very closely tied to the code. In addition, the requirement that the program run in less than 1 Meg places severe memory constraints on it. You, the programmer, should design the RTF in close cooperation with your producer and art staff.

One of the absolute, immutable fundamentals of CD-I is that sectors in real-time play are delivered at 75 per second. This enables you to do some math:

> 384 × 280 pixels for full-screen DYUV images is around 110K.
>
> At 2324 bytes per sector, it takes 47 sectors to store a full-screen image.
>
> At 75 sectors per second, it takes around 2/3 of a second to deliver a full-screen image from the disc to memory.

Therefore the fastest that you can deliver full-screen photographic-quality images is 1 1/2 frames per second. As you gain experience creating Real-Time Files, these kinds of calculations become second nature.

The Development Environment

Programming the CD-I player requires three basic pieces of equipment and a variety of specialized software tools. The equipment consists of a programmer's development station (usually a Sun or a Macintosh) with a very large amount of disk space; an emulator to mimic the behavior of the CD drive; and an industrial model CD-I player. The software consists of tools which convert audio and video files into the special formats required by the CD-I player; cross-compilers and assemblers; and Real-Time File and disc image builders. I'll address each of these in turn.

Development station

The CD-I player is an inadequate programmer's workstation. It is really not fast enough or convenient enough to serve as a desktop computer. You will need a separate workstation, the faster the better. At the moment there are software tools for the Sun and Macintosh platforms, with the PC a little farther beind.

Most computer games can be developed in ten to twenty megabytes of disk space. CD-I products will require over a hundred times that much space -- at least two and probably three gigabytes. It is generally better to capture video assets at much higher resolutions than the player needs in order to get the best possible image quality, but this means that your artists will need a huge workspace. A 640 × 480 × 24-bit graphics file, for example, requires 900K; this will typically be reduced to 110K if converted to full-screen DYUV. Audio files recorded as PCM typically undergo 3 or 4:1 compression as well. Consequently you will have to have room for both your original assets and their compressed CD-I form. Eventually, you will have to start backing up and deleting the assets in their original form. In any case you can count on needing three or four 650 Mbyte disc drives for your project. A high-volume tape backup system is also essential.

Emulator hardware

Since it is not possible to press a new compact disc every time you make a change to your code or assets, some means has to be found to emulate the presence of a CD. This is done by disconnecting the CD-I player's motherboard from the CD drive, and connecting it instead to special emulator hardware. This is a box or board attached to your development station which interprets the player's requests for CD data and retrieves the data from your development station's hard drive, at the appropriate delivery rate. You can't do this with the consumer player, which is why it isn't appropriate for development.

Emulators are very expensive, since they are low-demand, highly specialized pieces of hardware. Most cost several thousand dollars. The emulator for the Macintosh consists of a board and a software driver. There are at least two companies now selling emulator hardware; see the list of tool vendors at the end.

The file the emulator reads is called a disc image; it is a byte-for-byte copy of each sector of a CD, including all the error-detection and correction codes, lead-in and lead-out areas, and so on. The disc image is built by a disc building tool which opens your asset files one by one and lays out the sectors according to the way the Real-Time Files have been defined. You will, of course, need to have hard disk space for the disc image *in addition* to all your assets.

The CD-I player

The industrial model CD-I player consists of three stacked boxes hooked together with cables. The disc drive is in the top box; the computer is in the center box; and a pair of floppy drives, serial port, and SCSI interface are in the bottom box. Normally the CD drive is disconnected from the computer unless you are playing an actual CD-I disc.

On the back of the middle box are a VGA-style 14-pin connector delivering analog RGB video; an NTSC composite video RCA jack; and two RCA jacks for audio out. You will need a suitable monitor, amplifier, and speakers to display the video and audio. There is also a switch to switch the player from NTSC to PAL mode; if your monitor is capable of displaying 625 lines you can see how your product will look on a European TV.

Conversion tools

Fortunately, CD-I toolmakers have agreed on a standard for asset files which are going to be incorporated into CD-I discs. This is the IFF standard developed by Commodore and Electronic Arts, with appropriate extensions for the special CD-I audio and video formats. Any CD-I development station will need tools which can read and write these formats. I have used AudioMedia for audio editing and Photoshop for video editing. AudioMedia is capable of writing PCM Audio IFF files, which then require compression to ADPCM. Plug-in modules are available for Photoshop which permit it to read and write CD-I formats directly. There are also a variety of tools which run under the Macintosh Programmer's Workbench to convert audio and video files. Again, see the list of tool vendors for some tips on where to find these tools.

Cross-development tools

Microware sells a C cross-compiler and linker for OS-9 which runs under MPW. The GNU C compiler has also been ported to OS-9, but I have never seen it; I assume it is available from the Free Software Foundation. If there are other cross-development tools available I am not aware of them, nor have I heard of any languages besides C being available.

The OS-9 cross-compiler produces assembly code in one of its passes, and therefore includes a cross-assembler. Compilation speed is fairly good; I have heard mixed, anecdotal evidence about the quality of the code it produces. CD-RTOS (and OS-9) come provided with a shell, a small set of utility programs, and a text-only debugger. The emulator driver software usually also functions as a terminal, permitting you to attach your workstation to the player's serial port for debugging operations. You may use standard C I/O calls to communicate with the terminal, but you must be sure they are all removed before final release; otherwise the customer's player will hang trying to communicate with a serial port that isn't there.

Real-Time File and disc image development tools

RTF's are designed in one of two ways. You may either use an RTF description language, in which a text file tells the disc builder where each audio and video asset should go on the disc; or an interactive tool which permits you to "edit" RTFs by "cutting" and "pasting" the assets in. The description language is generally more powerful and versatile for complex RTFs, because it permits iterative asset insertion, branching, and other programming-language features. It also permits relative insertion, so that one asset can be inserted relative to the position of another; this is extremely valuable if you discover you need to insert a new asset and don't want to move all the others around by hand.

In addition, you may view an RTF graphically, via tools which display it on your development station. Each sector of the RTF is shown in its appropriate position, and identified with the channel number to which it has been assigned, and the type of data it contains (audio, video, or error-corrected data).

Building a disc image consists of specifying which Real-Time Files, and non-realtime files (such as your program's executable), should be copied from your development station's hard disk into the disc image. This specification takes the form of a text file called a "map" file, and is read in and processed by a disc building tool. The disc building tool reads the map file and begins creating the huge file on your workstation that is the disc image. When the map file calls for a Real-Time File, the RTF specification is read to see which asset files are included, and what sectors they belong in. One by one, your IFF asset files are opened and their data inserted at the positions specified. For a complete disc image, this process can take several hours, although in the early stages of your project it will probably only require a few minutes. When the disc image is complete, you can start the emulator, and run it on the CD-I player. When the whole project is done, the disc image can be pressed into a real CD.

Conclusion

This article was not intended to be a tutorial on programming the CD-I player, but a general introduction to the subject, and I hope that in that it has succeeded. There are many people far more experienced than I at programming the player, and OptImage regularly conducts classes. The CD-I player presents numerous challenges and fascinating possbilities. Whether or not it will live up to its promise remains to be seen.

Tool vendors

The following companies provide CD-I tools. This is by no means an exhaustive list, and I apologize to anyone who has been left out. Please keep in mind that I was one programmer using what I had to make an actual product; I did not do a lot of shopping around and I have not bought any tools lately, so I can't claim accuracy for any of this information. Inclusion here does not constitute an endorsement, nor does exclusion constitute a criticism.

OptImage, LP.

OptImage is a partnership of Philips and Microware. They produce development tools for the Macintosh and Sun. In particular, they produced the image-conversion and audio-conversion tools that I used. They also make emulator boxes, and a powerful library of routines called Balboa, which makes CD-I development much easier at some cost in control over the machine.

OptImage Interactive Services, LP
1900 N.W. 114th St.
Des Moines, Iowa 50322
1-800-CDI-5484

11050 Santa Monica Blvd.
Los Angeles, California 90025
1-213-445-5706
fax: 1-213-477-4953

Interactive Support Group, Inc.

ISG makes disc-building tools and emulator boards. I used their Real-Time File description langauge, CDL, and their disc image builder and emulator.

21032 Devonshire St. #209
Chatsworth, California 91311
1-818-709-7387
fax: 1-818-709-8160

Microware, Inc.

Microware makes CD-RTOS and OS-9; they also make the cross-compiler for OS-9 that runs on the Macintosh under MPW. They can provide documentation on CD-RTOS; similar documentation is also in the Green Book.

1900 N.W. 114th St.
Des Moines, Iowa 50322
1-515-224-1929

Equilibrium

Equilibrium makes a product called the DeBabelizer, which permits conversion of a wide variety of image formats on the Macintosh, as well as a number of other palette editing and image-manipulation features. They recently added CD-I IFF files to their list of supported formats.

475 Gate Five Row, Suite 225
Sausalito, California 94965
1-415-332-4343

Philips Interactive Media of America

PIMA does not actually sell tools, but they are the largest publisher of CD-I software in the United States, and have the most up-to-date technical information about the CD-I player.

11111 Santa Monica Blvd.
Los Angeles, California 90025
1-213-473-4050