Proceedings of the First CD-I Software Conference

February 11 - 13, 1990

Laguna Beach, California

Preface

This document contains the proceedings of the first Compact Disc Interactive software conference, which was held February 11-13, 1990, in Laguna Beach, California. It is intended to highlight the topics presented and discussed during the conference and not as complete minutes of the conference.

The presenters and panelists were chosen on relatively short notice by the program chairs and I wish to thank all of them for the time invested in preparing notes and presentations in time for the conference. I also want to thank all the administrative staff of American Interactive Media without whose hard work this conference would never have occurred.

I especially want to thank Lucy Lediaev for her service well above and beyond the call of duty, both in assisting with presentation materials and review of this document. Also, Andy Davidson, Philip van Allen, and Alty van Luijt who showed great diplomacy in refreshing my memory on many points referred to herein.

Finally, my most sincere thanks to all the conference participants without whom the conference would have been just three lovely days in Laguna Beach - especially Larry Lowe who gave us all a vision for the future of CD-I.

I hope these proceedings prove useful to you.

Charles S. Golvin
CD-I SC
Proceedings Editor

Table of Contents

Proceedings of the First CD-I Software Conference
February 11 - 13, 1990
Laguna Beach, California

Sunday, February 11

The conference began with a cocktail party and dinner, followed by a keynote address given by Larry Lowe, CD-I producer and designer. Larry has been involved with CD-I since its inception. His presentation focused on the importance of looking beyond what has been done so far to what he termed "interactive television." Larry related how his vision for CD-I was inspired in part by James Burke, and he showed some clips from his *Connections* series from Public Television. In addition, he showed videotape from an interview he did with Burke concerning CD-I. He stressed that conference participants, as those who are engineering the titles, are the only ones who have the ability to fully realize CD-I's potential.

Monday, February 12

Session: **Performance Issues**

Panelists: **Alty van Luijt, AIM**
Eric Miller, Microware

Alty's main point was that, until recently, engineering has been waging a functionality battle that is now nearing a close. The number of bugs in the player has nearly diminished to a manageable number. Thus, it is time to focus on improving performance. He emphasized that the 1.0 Philips player has a better clock rate and disc access than its predecessor and that the Green Book now provides that all seeks within 20 megabytes in either direction from the current location take no more than one second. He also noted that the current players form a reference, and future players will improve beyond this reference point.

Eric then offered ways that engineers can optimize performance. Real-time play gives the highest sustained data rate, but there are methods for improving the behavior of a non-real time read (which the driver translates into a play instruction). The CD driver allocates one 2K buffer for each path that is opened, and a read of less than a sector's worth of information places the data into this internal buffer. This 2K buffer is returned when the path is closed. A read of an integral number of sectors delivers the information directly into the application's buffer . One interesting point Eric made was that the current driver does not use short or long word moves, only byte moves. The lesson gleaned from this discussion is to always read an integral number of sectors. One technique mentioned for contending with small files that contain information needed simultaneously is to concatenate these pieces into a single file read as one, and then to parse the data in memory. Chuck Hughes of AIM mentioned that he is working on simple applications and is able to achieve the desired result using only non-real time files.

Eric discussed directories and recommended that files in a single directory be opened consecutively to reduce the seek time from the directory to the file. He also cautioned against having too many files in one directory, because that

could cause the directory information to span more than one sector. His maximum was thirty to fifty files per directory, depending on the length of the file names.

The next topic for improved performance was text output. Eric said that, when using UCM text drawing routines, one-bit, two-bit, and four-bit type fonts add an extra level of indirection that is not needed for the CLUT type fonts. UCM speed is best using CLUT type fonts with clipping set to off, and transparency set to on. The Vortex-produced Font Output Library (*FOL*) is recommended as the fastest text drawing available. These libraries are not in ROM, so you pay a memory price for the added speed. While FOL works with UCM font modules, it requires that each glyph begin on a byte boundary. A font to be used with FOL is likely to require modification and to consume more memory than the original font.

Two techniques were mentioned for improved animation: UVLO, like DYUV, but greater compression is achieved essentially through discarding some chrominance information; CARLIE, a scheme devised for greater run length compression, but applicable to any image format.

Eric mentioned that there are two ways to improve the performance of LCT writing. First, condense writes into large, block writes versus small, individual writes wherever possible. Second, use physical writes which are approximately two times faster than logical writes. This figure raised the question of performance figures for other CD-RTOS calls; Eric promised that a complete list of hard performance figures for each system call and combinations of system calls, on a base case validation player, would be available shortly.

The session concluded with a question from Kelly Jones of Fathom Pictures as to when the current performance bugs would be fixed, e.g., ss_seek(). Eric responded that the first fixes on floppies containing new drivers would be available within several weeks; new ROMs would be several months away. He also explained that the current player's features vs. bugs would be decided by a panel scheduled to meet within three weeks.[1]

[1] Recommended reading on performance in CD-RTOS: AIM technical note #49.

Session: **Title Architecture**

Panelists: **Armelle Loghmanian, CTI**
Ken Ellinwood, AIM
Tom DiDomenico, OptImage
David Cohn, CD-I Systems

Armelle Loghmanian

Armelle gave an overview of the architecture used to create the Renault disc. This is a European four-disc set, intended for PAL systems. It uses seven languages and contains over twenty-five thousand pieces of sound. Each piece of sound is between thirty and sixty seconds long. Development has spanned over four years at CTI.

The architecture of the Renault disc is a single process with multiple threads; it uses many timers and is event-driven. There are many hierarchically structured managers in the system. Among these are a play/sound manager, a timer manager, a video interrupt manager, a video manager, and a hotspot manager.

The core of the system is a signal manager. CTI signals are associated with OS-9 signals, and different classes of CTI signals may exist. The signal handler translates OS-9 signals into CTI signals and parameters so that the same OS-9 signal may have multiple meanings based on the value of the parameters in the associated CTI signal. CTI signals are divided into two modes: immediate and delayed. Immediate signals are handled as soon as they are received, while delayed signals are placed in a first-in-first-out queue and are handled as the application has time. Each signal has an associated function that is called when the signal is handled, irrespective of mode.[2]

The main problem still to be solved for Renault is that of memory management. A question was raised as to overall code size—Armelle said that the Renault application consists of approximately 100 kilobytes of code.

[2] The essence of this signal handling scheme is incorporated in the Balboa run time project that will be announced later this year.

Ken Ellinwood

Ken gave an overview of the architecture of *Cartoon Jukebox*, a title intended for children, ages two through seven. It consists mostly of run length animation displayed at ten frames per second over level B mono audio. This enforces the build time requirement that all frames be no more than five sectors; each frame is preceded by its own CLUT.

Ken chose a multi-process approach to the title, with inter-process communication handled by AIM's Inter-Task System Mail (ITSM) package. There are four processes: a main state engine, a play manager (DSMS), a cursor process, and a cycler process. These act autonomously, except during screen state changes, and all run at the same priority.

The state engine handles the screen structures and the underlying control structures. It contains the screen handler routines that are instantiated for each object. It also holds information on how to handle incoming data, the sources of which are DSMS and the cursor process. DSMS is responsible for delivering data off the disc. It seeks to assets based on symbolic labels and delivers data into application-specified buffers. The cursor process is responsible for moving the cursor and binding the cursor when appropriate. The cursor in *Cartoon Jukebox* can be either the graphics cursor or a full screen run length image. The cycler process is responsible for any color cycling done during the application.

The memory management scheme Ken uses has several static blocks in plane B for control panel and other static items; the remaining memory for each plane is kept in a large block from which needed pieces are parceled out. These used pieces are returned when no longer needed, and then they are reused.

Ken found that a multi-process approach was an advantage when more than one programmer was working on the title, but that debugging was much more difficult than in a single process architecture. He debugs using Microware's **unibug** package on an MRS, opening a window for each process. This is satisfactory for debugging one of the processes, but not for debugging the inter-task communication. However, Ken stressed that ITSM has been very

reliable for him. Eric Miller said that Microware is working on a source-level version of **sysdbg**, their system state debugger. It was also mentioned that multiple processes require more memory (roughly 5 kilobytes of overhead per process).

Tom DiDomenico

Tom told us that he had been using a multiple process approach for some time. He used six processes: an audio process controlling sound maps and attenuation; an input process handling the pointer and graphics cursor; an I/O process handling non-real time plays and real time file access; a play process responsible for real time play and signal handling; a video process for video memory and effects; and a state process. The state process is the parent process that handles the flow of control of the application. His inter-process communication was by named pipes, one per process, using fixed length messages. He also used signals for quick communication.

He found that debugging and synchronization in this architecture was very difficult. Another drawback was the extra RAM per process; also the CPU load for communication decreased performance in the system. For these reasons, he is now employing a single process architecture driven by signals and using state tables for flow of control. However, he said that he may still use another process for requisite asynchronous operations. He noted the similarity between his approach and that used by CTI, especially the idea that some signals must be handled immediately while others can be placed in a queue for handling when the system allows.

Tom has used these approaches in the OptImage Starter Kit and PCEC's audio products training application.

A discussion then ensued on the value of both multi- and single-process architectures. Graham Sharpless mentioned that the multiple process approaches cited were all at the same priority and always active—perhaps tasks of different priorities that slept when not active would suit the bill better.

David Cohn

Dave is working on the Gambling disc which contains five different games. Each game consists of a relatively large amount of code, and it is not feasible to have them all in memory at one time. Thus, he has adopted a multiple thread approach, one per game, and a controlling process that handles the navigation through the main casino. At the time of his presentation, he was considering one of two approaches—use either one subroutine module per game or make each game a subprocess of the main process. In either case, it requires that code be loaded when the game choice is made and then unloaded from memory on return from the game.[3]

The mention of subroutine modules prompted an exchange on this topic among several of the participants. Eric Miller said that subroutine modules may not use initialized data or reference global data. Someone mentioned that Jon Piesing of PRL had created a document describing how to use subroutine modules and still reference global data. Keith Lehman of Spinnaker claimed that there would be problems with signal handling in Jon's method. Jon countered that the subroutine module could install its own signal handler, and the calling routine would reinstall its interrupt handler on return from the subroutine module. Someone raised the question of trap handlers versus subroutine modules—Eric said that subroutine modules are more efficient, because a context switch is not needed. He also mentioned that some C library calls reference global data and, hence, are not candidates for inclusion in a subroutine module.

Following Dave's presentation the question of object-oriented programming (OOP) was raised, in particular the suitability of this approach for CD-I. Armelle said that although OOP was used for the Renault disc, she was still not convinced of the necessity for that approach. David Furlow of CTI said that one problem was the difficulty of linking objects at build time with the object itself. In deciding whether OOP is suitable for an application, look at the re-usability of data structures and methods—if high, this would be a good candidate for an object-oriented approach. The general conclusion was that the nature of the application should dictate the architecture and that object-oriented

[3] Recommended reading on real-time code loading: AIM technical note #47.

design is not necessarily tied to an object-oriented implementation (e.g., C++). It was resolved to arrange a breakout session on OOP for CD-I later.[4]

[4] See the enclosed summary of the OOP breakout session.

Session: **CD-I IFF 1.0**

Panelist: **Andy Davidson, AIM**

Andy's presentation had two parts. The first part was an introduction to CD-I IFF in general: its purpose, its scope, what it does and does not achieve, and how to use it. The second section focused on changes for CD-I IFF version 1.0.

CD-I IFF is the de facto standard for the exchange of CD-I image and sound files. It supports a hierarchical structure within files (e.g., a "chunk" is contained in a "FORM," many of which can be contained in a "CAT "). CD-I IFF is not a database management system—the only information contained in an CD-I IFF file is that necessary to correctly output its contents. The CD-I IFF specification is public domain and is distributed by AIM. Libraries to read and write CD-I IFF files are also available from AIM and run on Unix, OS-9, PC, and Macintosh platforms.

The following features will be added to CD-I IFF version 1.0:

- Environment chunk: provides a standard method of adding non-standardized data
- User-defined image formats
- New image models including multi-component types
- Extended header for images
- Audio data type that excludes the twenty null bytes at the tail end of each sector for sound map output
- Irregular shapes for image data, regions, and hotspots
- Data types to support full motion video

The subject of animation (image sequences) was discussed extensively; the committee decided it more prudent to allow a standard to evolve and add that standard to the next revision of the specification.

Session: **CD-RTOS 1.0**

Panelists: **Philip van Allen, AIM**

The focus of Phil's presentation was an update on all the outstanding bug reports currently in effect for the 1.0 Philips player. Please note that this list has changed since CD-I SC and the most up-to-date bug/fix list is included with these proceedings.

Some points were made in reference to the bugs discussed:

- Televisions with comb filters exacerbate the poor NTSC encoding of the player
- Loss of audio sectors at the beginning of a play is most noticeable with C-level audio
- Fabio Minazzi speculated that the "sound map plays garbage" bug only occurs with level B audio. Dave Lampert of OptImage claims the bug only occurs when a sound map is reused; he works around it by saving the sound map descriptor

Some points were made that will remain valid after the related bugs are fixed:

- Examine PCB_Stat immediately, because it is updated every sector
- It is possible to not have system I/O paths on a player. This means that **printf's** could be sent to UCM or CDFM causing problems
- Set **gc_org(path, 0, 0)** yourself

Session: **Memory Management**

Panelists: **Eric Miller, Microware**
Paul Rother, Independent
Jon Piesing, PRL

Eric began by giving some details of exact memory locations in the Philips 1.0 player, including an overhead showing schematically how colored memory is used. The first $500 bytes in each plane are reserved for the system; $50 bytes at the end of plane A are for the hardware cursor instructions. Eric mentioned that recent discoveries have unearthed approximately twenty-five kilobytes of unused kernel memory that will be returned in the next revision of the operating system. He also explained that all the UCM **gc_** calls are actually implemented as DCP instructions.

He next gave a rundown of the static storage used by the various CD-I file managers. CDFM has space for sound map identifier tables, path table buffers, and play control structures used by I$Read. There are also identifier tables for FCTs, LCTs, and system fonts. Each entry in an identifier table is 16 bytes; there are sixty-four identifiers each for sound maps, regions, and draw maps and sixteen identifiers each for FCTs, LCTs, and system fonts. When a table is too small to accommodate a request to create an item, the current size of the table is doubled, the current table is copied to the new table, and the old table is deleted. This will probably cause fragmentation.

Eric explained that **srqmem(size, DONTCARE)** will choose memory from the plane with the largest amount free. He explained further that the OS-9 system memory allocation functions allocate memory in sixteen byte chunks whereas **malloc()** allocates memory in eight kilobyte blocks from which it then parcels out smaller chunks. Subsequent calls to **malloc()** that require more than that available in the current 8 kilobyte block receive another 8-kilobyte block. Pieces returned with **free()** carry an eight-byte overhead and begin on a longword boundary.

Eric's recommendation for memory allocation is to predeclare as many structures and buffers as possible and to allocate C structures allocated at run

time in blocks or in sixteen-byte entities. The question was raised about the limitation of static allocation to 64 kilobytes; Eric mentioned that the Microware C compiler supports a datatype **remote** that allows preallocation to exceed this limit.

Someone asked what system calls allocate memory - Eric's list was **F$Srqmem**, **F$Srqcmem**, **F$Load**, **F$DatMod**, **F$IRQ**, **I$Open**, **F$Send**, and perhaps **F$$Svc** (he wasn't sure). In regard to **F$Send**, a signal queue entry is created (sixteen bytes each) if the signal is not handled. His recommendation was to not allocate memory in your signal handler and to mask signals as little as possible.

One final note of interest concerned FCTs. The application's FCT, on execution of **dc_exec()**, is copied by the system after filtering of certain opcodes (set display start address, link FCT to LCT). The size of the memory used for the system's FCT is forty-eight bytes greater than the size specified by the application's call to **dc_crfct()**.

Paul Rother

Paul discussed the memory allocation strategy he employs in *Jukebox: Oldies*. The essence of his scheme is to allocate one static block of memory in each plane at the start of the program and to continue to reuse portions of those chunks in a cyclical manner. This eliminates fragmentation, but places a memory management responsibility on the application.

Jon Piesing

Jon presented an overview of the memory manager he developed. He first outlined his reasons for developing a memory manager, and then explained some of the considerations involved in designing his manager. The issue of garbage collection was raised; Jon said that the problem with garbage collection is in attempting to scan-synchronize garbage collection of video data currently being displayed. While the OS-9 memory allocation scheme generally proceeds from high to low memory, the scheme he developed allows the user to specify in which direction memory allocation proceeds. This makes it possible to transition

between memory configurations without fragmenting memory, provided the memory can be returned in the order in which it was allocated.

Jon's scheme essentially sits in front of the OS-9 system memory allocation calls and handles all memory allocation requests, including those of the system itself. Eric Miller pointed out that each memory allocation or deallocation request could result in as many as four system calls; Jon agreed with this analysis.

All the presenters agreed that, at a minimum, applications must execute as few allocation and deallocation calls as possible. This theme was common to all three presentations.

Session: **Video Management**

Panelists: **Christopher Thompson, Vortex Interactive**
Eric Miller, Microware
Jon Piesing, PRL

Christopher Thompson

Chris presented an overview of the *GO* package. The presentation covered his motivation for developing the package, as well as its features. The basic entity in *GO* is a strip, a section of video at least full-screen width. The user does not have to remember all the correct display control program instructions for a particular coding method and is able to specify the location of objects in simple Cartesian coordinates. The concept of an image is essentially an object-oriented one: the user changes characteristics of an object and the accompanying visual change occurs as a result of that change.

Among the features Chris cited were:

- Buffers created are always an integral number of sectors
- Simple graphics capabilities: fill routine, no clipping, blitting, simple text output
- CLUT loading and CLUT cycling
- All coordinates specified in pixel coordinates

Eric Miller

Eric's presentation focused on Microware's *RAVE* package. He explained that *InVision*, *RAVE*s progenitor, did not provide the programmer enough control at a low level. The major advantages in *Rave* are in the area of provided hooks to lower levels. Provided dynamic effects are based only on rates. The low level routines that address the FCTs and LCTs are published so that a programmer can access the control tables directly. Also, accompanying documentation specifies which fields of the control tables *RAVE* uses. The communication scheme for accessing the cursor/hotspot controls is also documented.

There is build time support that allows the user to create hotspots for controls and state tables. These tools output include (.h) files that are then consumed by the program.

Someone asked about overall code size: Eric responded that DSM, the communications package that forms the basis of *RAVE* , is between four and six kilobytes, plus a 256 byte data module. The remaining code size depends on what features from the package are actually used by the application.

Jon Piesing

Jon's presentation focused on the PRL Video Manager, which is actually made up of eight separate managers. He emphasized that all of the managers' objects are initialized with "sensible" defaults so that the user need not go through a large number of initialization calls to set up a display. As with *GO* the user need not know about FCTs or LCTs in order to display pictures and create effects. The architecture is open, and data structures are all published so that the application can pick and choose which features of the manager to use and which to discard.

The basic construct used by the system is that of a "picture" which is like a drawmap (and may even contain a drawmap). However, a picture can be directly displayed without any additional information other than what is in the picture itself. In addition to the information contained in a drawmap, a picture also holds transparency control information, transparent color (if appropriate), image contribution factor, viewport, location, and other additional information needed to correctly display the image. All the active pictures are held in an ordered linked list, with each picture displayed after those preceding it in the list. The Display Manager analyzes the picture list, figures out which pictures go where, and queries the CSD to correctly set the compatibility mode. In addition, the Display Manager supports the concept of a display segment. A display segment resembles *GO*'s strips, but includes both planes simultaneously.

Among the other constituent managers and their features are:

- LCT Manager: the user may reserve a section of the LCT to be left untouched by the LCT manager
- Picture Manager: creates and displays pictures
- Matte Manager: transforms logical shapes into DCP instructions
- CLUT Manager: writes CLUT data, constructs antialiasing colors for text
- Text Manager: writes text to pictures, supports kerning (font format deviates from UCM format)
- Video Interrupt Manager: allows user to scan synchronize functions and effects; uses signals not events[5]

[5] The PRL video and accompanying managers are being integrated into the Balboa run time project that will be announced later this year.

Session: **Optical Data Errors**

Panelists: **Alty van Luijt, AIM**
Robert Patton, AIM
Jim Bumgardner, ISG

Alty first presented some figures regarding the frequency of occurrence of bit errors on new discs and on deteriorated discs. His figures said that the bit error rate (BER) on a new disc, after the CIRC, is approximately 10^{-11}. This translates roughly to five out of every one hundred discs that leave the factory containing a single bit error. For comparison, the BER for a magnetic disc is roughly 10^{-9}. The BER at which tracking is lost is 10^{-7}; in this event, there is nothing that an application can do to recover.

Next, Alty spoke about the CIRC and ECC and the ability to recover from errors. The CIRC deals well with smudges and radial problems, but not with concentric problems. Alty drew two graphs: the first was a plot of uncorrected errors as a function of raw errors and showed the behavior of a poor CIRC, a good CIRC, and ECC; the second graph was a plot of error probability as a function of sector location and showed that given an error the probability of an error in adjacent sectors decreases with distance until one revolution further where a slight increase in probability occurs. The final topic on error correction was a brief explanation of the ECC algorithm.

The next discussion focused on what applications can do to protect themselves in the event of an error. Alty estimated that one percent of the data on a CD-I disc is critical. He discussed four possible actions for a title when an error occurs:

1) Correct
2) Conceal (e.g., duplicate scan lines in video data)
3) Abort and have dialog with the user
4) Ignore the error.

For correction, he gave three alternatives: retry—this is only effective if the player was bumped or moved; duplicate data—in building the disc, decide which data is critical and place two copies of that data physically separated on the disc; ECC—place critical data in non real-time form 1 sectors so that they are error corrected.

Finally, Alty mentioned several Green Book modifications in the area of error detection and correction. First, in the case of real-time form 1 sectors, the real time takes precedence over the error correction—if the ECC decoder is not capable of correcting errors in real time, then these form 1 sectors will be delivered with errors. In the case of form 1 sectors with errors corrected, the play control error bits might still be set (by the CIRC error detector).[6] Finally, there is a CSD entry specifying whether the player has a decoder capable of doing real-time ECC.

Robert Patton

Robert presented a strategy for dealing with errors occurring during real-time play. The main point of this strategy is to duplicate all critical data on the disc. He pointed out that current disc builders do not automatically support this replication, so the title developer must do it by hand. In deciding placement of copies, he recommended a distance more than one revolution (roughly ten sectors per revolution), but within the seek time tolerance of the application. It was pointed out that the error detection capabilities are identical for form 1 and form 2 sectors. His final recommendations regarded error concealment. With regard to images, unless the picture is considered critical, do not bother to conceal the error. In audio, the effect of an error diminishes over time; replacing the data containing the error with zeroes will reduce the effective error.

Jim Bumgardner

Jim talked about the error injection capability of the Mac CD-I tool set. This program overwrites data in a disc image and records the original data and the location of the changed data so that the intact disc image can be

[6] Author's note: in the meantime this has been disproven. The driver resets the play control error bits while performing the error correction.

reconstituted. This does not simulate real-world errors like smudges and does not introduce errors into the Q-channel.

There was some discussion and lack of clarity regarding whether this error injection scheme is valid only for testing ECC capability or for CIRC error detection. Regis Bridon said he would investigate and respond.

Session: **Animation Techniques and Image Compression**

Panelists: **Alty van Luijt, AIM**
Kirk Rader, AIM
Mark Thomas, Capitol Disc
Michael Edge, Spinnaker
Doug Corarito, Independent

Alty began by talking about the need for image compression and the advantages and disadvantages of using image coding other than those specified by the Green Book. The main disadvantage is using the CPU to decompress image data.

He then introduced the Compressed AIM Run Length Image Encoding (CARLIE) scheme. This scheme takes advantage of high vertical correlation between adjacent scan lines to compress image data. Its use is not restricted to run length data. The current version of the decoder code is roughly 1.9 kilobytes and is capable of keeping up with cartoon style animation. The encoder is still being completed. Finally, Alty pointed out that further compression could be achieved by adding temporal interpolation.

Kirk Rader

Kirk presented an overview of UVLO, a compression method developed at PRL and based on DYUV. The compression derives from some chrominance information being discarded and optionally from vertical interpolation. UVLO is capable of animating roughly seventeen percent of the screen, although the CPU utilization is high (Kirk estimates between 67 and 95 percent). Kirk also discussed the production pathway needed to create UVLO sequences. These presently rely on a Sun with an Androx board, and there are some video peculiarities in the Androx board that lead to problems with the UVLO images. Finally, Kirk gave demonstrations showing UVLO animation both at fifteen and thirty frames per second with accompanying frame reduction.

Mark Thomas

Mark described the animation technique in use in the *Children's Musical Theatre* disc. This disc contains five songs with three sets of lyrics per song and three modes in which the songs may be played. The animation is done in the foreground plane with color key transparency showing a fixed background plane. The audio is B level stereo, and the animation is played at eight frames per second. The overall data rate is roughly one hundred twenty kilobytes per second.

Mark described the calculations that were needed to devise the best performance scheme based on the range of seek times different players provide. A question was raised as to the production pathway used to generate the animation. Mark said that the animation was all created using MacroMind Director software on the Mac.

Michael Edge

Mike described the DYUV animation technique he devised for use in the *Star Tribes* title. These animated sequences are in a DYUV plane, part of which is obscured by a CLUT plane, and are accompanied by C stereo audio. The DYUV images are 248 X 144 pixels, and animate at ten frames per second. Because this is too much data to deliver in the bandwidth available, he devised some build time tools that look for differences between adjacent full DYUV frames and encode the difference information into a format that describes the horizontal offset(s) and length of data for each scan line. These differences are delivered at run time and then decoded to give irregular partial screen updates of the DYUV frame. The build time software enforces a seven sector requirement on each frame's tokenized information to ensure that the desired frame rate can be achieved. Mike noted that the CPU overhead of this technique is quite high. He also mentioned that the source images need to have any background noise filtered out; otherwise the encoder software interprets the noise as changes in the animation frames. He also said that he had investigated other additional compression schemes such as Huffman coding, but these had not borne fruit. Dave Lampert of OptImage noted that his statistical analysis had predicted a 1.2 to 1.3 compression rate from Huffman coding of natural images.

The demonstration of this DYUV animation technique, showing a Claymation talking head with a spaceship console overlay, was extremely well received by the conference attendees.

Doug Corarito

Doug discussed the blitting scheme originally devised for the Dark Castle title, then adapted for general application. He first outlined some of the considerations that needed to be weighed in designing the blitter, which was to be used explicitly for CLUT data:

- Is transparency required?
- Is clipping required?
- Is masking required?
- Need the background be preserved?
- Speed of decoding
- Encoded data size
- Number of significant bits per sprite

The blitter can be tuned to optimize for one of these factors over another, and there are different versions of the blitter depending on the number of significant bits per sprite. For Dark Castle, Doug uses sixteen colors in the foreground, and the blitter is optimized for transparency. Doug closed with an emulated demonstration of the Dark Castle title.

Session: **Audio**

Panelists: **Howard Soroka, Independent**
Fabio Minazzi, Philips APG
Doug Corarito, Independent

Howard's presentation focused on the production side of audio for CD-I. His first message was for the engineer to stay involved with the design team in design choices regarding audio. His next focus was on processes; he stressed that audio must be created using professional recording and mixing equipment and techniques. Audio should be recorded to hard disk as PCM data.

He next outlined the requirements for a digital audio workstation in order to produce CD-I audio files. First, the files must not reside in some kind of 'black box;" that is, one must be able to perform operating system commands on the sound files. Second, you must have the ability to manually edit waveforms. Third, there must be a provision for automated editing, the ability to create edit decision lists (EDLs) using SMPTE time code for batch processing. Fourth, you must be able to encode from PCM to ADPCM. Finally, the workstation must have essentially unlimited disc space; that is, the ability to expand local storage without restriction.

Howard strongly recommended *SoundTools* by DigiDesign as a system that satisfies all these requirements and more; it can be used for high quality recording. He also mentioned another DigiDesign product, *Audiomedia*, which is essentially a subset of *SoundTools* but lacks digital I/O. *Audiomedia* could be used for scratch recording, but not for high fidelity recording. The question was raised as to the usefulness of *MacRecorder*, a lower end system that creates Macintosh sound resources. Howard was adamant in his statement that the output of *MacRecorder* is not acceptable for CD-I audio production. The only drawback he mentioned regarding *SoundTools* was the lack of real-time ADPCM encoding, which DigiDesign expects to release "real soon."

Fabio Minazzi

Fabio first discussed testing of ADPCM encoding algorithms. He explained that ADPCM encoding is highly non-linear and that normal tests, based on sine waves, are not valid for ADPCM testing. In fact, sine wave inputs give the worst results when encoded to ADPCM.

He next gave guidelines on how best to test the quality of encoding. He first recommends using a one kilohertz sine wave with 0.8 X 0x7fff amplitude. Perfect playback of this test pattern in levels A and B proves that the data format is correct and that the filter structure representation is correct. The quantizer structure and predictor and gain choices are also validated by this test. The next test uses a one kilohertz faded in sinewave of maximum amplitude 0.8 X 0x7fff, with a fade duration of seven seconds. When encoded to level A this should play back perfectly. This test validates the implementation of the adaptive quantizer and the noise shaper. His final test employs a ramp of sinewaves, spaced by 100 Hz, beginning at 500 Hz and ending at 7000 Hz. Each wave has a duration of roughly 0.25 seconds with amplitude 0.8 X 0x7fff. Perfect reproduction of these signals after encoding to level A verifies the correctness of filters 0, 2, and 3.

Fabio next showed a series of graphs of waveforms illustrating the encoding of various types of instruments, including piano, woodwind, and percussion. He emphasized that encoding to levels B and C will actually increase any noise in the source material. Thus it is imperative that audio that is to be encoded to these levels be well recorded with a minimum of noise.

Finally he gave some results of tests he had done. He found that level A encodes very well and was, in fact, preferred over CD-DA in blind tests. He said that pop music is the most robust for encoding (especially to levels B and C), and the most sensitive signals are fast-attack harmonic signals (he cited Paganini as an example). He emphasized that different signal sources cause different degradation. Thus, the source material must be considered in choosing the encoding level for CD-I audio.

Doug Corarito

Doug presented an overview of *AMP*, the audio package he created initially for use on the Dark Castle title. This package supports sound combining, both between pairs of in-memory sound maps and sound maps with audio played from disc. Sounds are combined at the sound group level. The package uses the alarm feature of AIM's Inter-Task System Mail package to control the playback of combined sounds and uses a single two-sector sound map as memory overhead. The package is also intended to support buffered channel switching of real-time audio playback.

AMP contains three separate queues for sounds—one for the right channel, one for the left channel, and a neutral queue whose entries play in the next available channel. Sounds entered into each queue have both a priority and a loop count associated with them. *AMP* also supports an infinite loop count that plays the associated sound until cancelled. The question of system overhead was raised; Doug estimated twenty percent overhead. He also pointed out that when the package is paused, it continues to mix silence. Thus, for minimal system overhead, it is better to turn *AMP* off.[7] The presentation ended with a rousing demonstration of *AMP*'s functionality.

A brief discussion ensued regarding naming conventions. Eric Miller pointed out that it is impossible to predict the names of modules that manufacturers will put in the system. However, if a title application begins with the characters **cdi_** then compatibility is assured.

[7] This functionality has since changed: when paused AMP consumes no CPU resources.

Session: **Disc Building & Play Management**

Panelists: **Drew Topel, AIM**
Jeff Kesselman, AIM
Ken Ellinwood, AIM
Chris Angelli, IPA
Armelle Loghmanian, CTI

Drew Topel

Drew discussed AIM's Disc Building Language (DBL). He first presented the reasons for developing DBL and the problems it attempts to address. The first goal was to allow symbolic access to data on the disc. The next goal was the retention of data contained in the source IFF files from which real-time files were assembled. The final goal was to package the signal sequence data for each real-time record in a form understandable to the application. Drew next outlined some of DBL's features: connection to existing disc building tools; output of a low level sector map; and a script file that is consumed by *rtrb*, the OptImage real-time record builder; and the ability to specify forward or back fill of sectors by assets.

He next discussed some of the problems he had run into in first developing a generic play manager. Among these, he noted the difficulty in debugging the system, the importance of synchronizing with the application, the treatment of errors in the datastream, and the differentiation between a logical real-time record and a physical real-time record.

Jeff Kesselman

Jeff presented an overview of the *Play Manager* he is developing for the Balboa project, a generalized run-time system. He first outlined the requirements he had specified for the functionality of the system and then explained in detail how he had implemented, or planned to implement, the system to address these requirements. One of the most notable features of this Play manager is that it provides an overall map held in memory that specifies the location of other map information used for portions of the application in a real-

time file, instead of requiring that all map information be held in memory at all times. This data is replicated to contend with errors in playback.

Ken Ellinwood

Ken discussed the functionality of the *Data Stream Management System (DSMS)* he developed for the *Cartoon Jukebox* title. One of the main requirements Ken had in developing DSMS was to reduce the memory overhead of the map information used to control real-time play. Basically, he uses a run length compression scheme to take advantage of the fact that many of the assets use buffers of the same size. He uses DBL to generate the real- time record scripts, then he filters the map output of DBL into his own format. He stated that his map files are well under one sector's worth of information; they are simply concatenated with his application and loaded at initialization.

Chris Angelli

Chris gave an overview of the process used to develop the real-time records for the *Tell Me Why* discs. Like Ken's scheme, she has a program that parses and filters the map output of DBL into a format of her own devising. However, one addition that the team at IPA made was to add timing and effect information into the DBL scripts themselves, then distill this information into symbolic information that is consumed at run time. The advantage of this approach is that effect information can be changed without having to rebuild real-time records. This gives the designers more flexibility in tuning the application, and relieves the engineer of much of the script generation responsibility.

Session: **Full Motion Video**

Panelists: **Andy Davidson, AIM**
Phil van Allen, AIM

Andy Davidson

Andy gave an overview of the full screen, full motion video (FMV) that is being developed for CD-I. He discussed the features of FMV, the additional hardware requirements placed on the player, the bandwidth consumption, encoding of data into FMV format, and some of the software engineering issues that will need to be addressed in order to incorporate FMV into titles. He also briefly discussed the various elements of the algorithm and its relation to possible ISO standardization for motion video.

Phil van Allen

Phil gave further insights into the software engineering issues with which engineers will have to contend in order to integrate FMV into future titles. Among the issues he discussed were memory considerations, use of FMV circuitry to display still images, techniques for switching between FMV "channels", and the granularity at which full motion video sequences could be addressed.