AMERICAN INTERACTIVE MEDIA
A Philips/PolyGram Corporation

AIM Technical Note #49

PERFORMANCE CONSIDERATIONS IN CD-RTOS

Alty van Luijt
Vice President of Engineering

December 1, 1989

Although recent changes in the 1.0 Philips JNMS player and in CD-RTOS have resulted in better responsiveness, title designers and software engineers need to take additional steps to assure satisfactory performance from a CD-I disc.

PERFORMANCE CONSIDERATIONS IN CD-RTOS

Introduction

One of the major concerns for CD-I designers and software engineers is performance. A disc that is slow to respond or that exhibits inconsistent response times to user actions is frustrating to the user.

The 1.0 Philips JNMS players we currently have show better responsiveness than the older 0.99 players. Two hardware improvements are responsible for this:

- A new drive mechanism and better interface protocol tot he player gives a much faster physical seek behavior.

- The CPU now operates on a clock frequency of 15 Mhz, instead of 10 Mhz.

Some improvements have also been made to CD-RTOS, resulting in a faster response. In the coming period, the 1.0 players will become the "reference players," so their performance level will be the yardstick for future CD-I players.

Nevertheless, in designing applications there are still other issues on which we can focus attention to considerably improve performance. The following are some recommendations for improving performance.

File Formats

In obtaining data from the disc, real-time files have the highest throughput. Non real-time files that read on integral sector boundaries have quite reasonable throughput; however, non real-time files that read on non-sector boundaries are very inefficient. (The driver continually allocates and deallocates temporary buffers to hold the partial sectors.) Due to the asynchronous nature of a "play" of a real-time file, the application can continue to do other things while the data is streaming off he disc. A "read" of a non real-time file causes the application to wait until the read is completed.

To read an equivalent amount of data from a larger record is more efficient than to accumulate it from multiple, noncontiguous smaller records. Probably the worst case is when data is left in IFF format and then put into non-real-time sectors with arbitrary boundaries between small files. The optimum is achieved by concatenating all relevant files into a larger real-time file. (Refer to AIM Technical Note #47, *Real-Time Code Loading*).

Opening Files

The system keeps the path table in memory. This list contains pointers to all directories, not to files. This implies that a directory needs to be read to an open file. A lot of seek time can be saved by opening multiple files belonging to the same directory at the same time. Note that this is true even though CD-RTOS currently does not cache the directory sectors. And even more so now version 1.1 buffers the most recently accessed directory sector.
There are two recommended options for opening files:

1) Open the first file. Put a picture up on the screen and possibly audio from memory) to present something to the audience. Then, open the other files from the same directory. This saves considerable seek time compared to opening files one at a time during the application.

2) Open subsequent files during times where the system is not doing anything useful or is playing back a sequence from memory.

Text

UCM text efficiency has been the subject of a great deal of criticism. Nevertheless, there are ways to use UCM that are faster than others. First, the overhead of coloring entries in one- or four-bit text is considerable. If your fonts are packaged in a CLUT 4 module, rather than in a quad bit module, the extra step of color mapping is avoided. (See the Green Book, pages VII-70 and 71.) If you are writing a four-bit CLUT draw map, such as a double-resolution draw map, this can save significant time. Also, longer strings of text are more efficient than shorter ones. Some of the settings in Dr_Txt can help as well. Maximum speed is achieved with "Xparency on" and "Clipping off."

Of course, for maximum speed at the expense of memory, you can use the font output library provided by AIM, or the Balboa Text functions

LCTs

the logical writes (with coordinate transformation) are about half as efficient as physical writes, at least if you are writing more than one line at a time. Generally, it is a good idea to maintain a "shadow area" and copy LCT entries in clusters that are as large as possible.

FCT's and double buffering

DC_Exec is a slow mechanism to support double buffering see AIM technote #63.

Draw Maps

The UCM routines are fairly general purpose; they take into account boundaries, clipping, and the possibility of making transparent copies. This implies that the overhead is fairly high, especially for copies of smaller areas. For animation purposes, you will probably always want to use a customer blitter to do a specific job very fast.

AIM publishes such a blitter, which was optimized for the Dark Castle project, as a general purpose function.

Allocation and Deallocation

Allocation and deallocation of various memory data structures are time consuming activities. For instance, creating an eight-bit draw map takes on the order of ten milliseconds. So, reuse your structures whenever possible. One of the methods is to create a large draw map and to build individual screen areas inside of the draw map. Use the pointer mechanisms to select the part to be displayed.

To take this one step further, treat your memory as general purpose buffers. Use the pointers of a single draw map and sound map to create a kind of audio/visual output mechanism for the general buffer area.

This latter step might be pushing overhead conservation to the extreme. For most titles, the overhead associated with draw maps and sound maps is quite tolerable, and you obtain ease of use in return. Just remember to reuse draw maps and sound maps whenever possible and avoid excessive dynamic creation and deletion.

Use Compression Techniques

Typically, compression techniques, such as UVLO, or software-coded versions of graphics compression are considered for cases where the system is pushed to its limits, as in motion video or animation. Yet those same techniques can give faster responses for more mundane tasks. Loading a smaller image and decompressing it by the CPU is often faster than loading he larger uncompressed image directly. If compression gives you an option for keeping things in memory rather than loading them from the disc, the speed improvement can be dramatic.

Use Available Hardware Resources

The two planes and hardware decompression facilities can sometimes be used to speed up operations that are traditionally time consuming. One example is to blit animation characters over a background. Storing the background in the second plane eliminates the need to store and restore the parts of the background that are covered by the foreground character. Similarly, moving part of an image across the screen—for example, in a windowing environment—can be done very efficiently by coping the part into the foreground plane, scrolling the foreground plane to its new position, and then copying the image back into the background plane.

Programming Style

Probably the most important factor of all is to know the limitations of the medium and to design around it. Just like in the old days with streamer tapes, you need to keep the data coming off the disc.

- Ask yourself whether you could be preloading something useful whenever you see a moment of interruption in the data stream coming off the disc or when you are waiting for user input. Even if you cannot actually load the data, perhaps you can already reposition the head, using SS_Seek.

- When you interleave real-time files, ask yourself what you could do during the play of a real-time file in the area of preload and make a conscious decision as to what level of interleave to support.

- Use the fact that short seeks on the disc are faster than long seeks by organizing the disc layout properly. Consider duplicating critical data on various positions so that there is always one version within reach of a shorter seek.

- Some programs lend themselves to a prediction of user behavior. If the probability of a user making a certain selection is very high, utilize that knowledge. One caveat in this area: Users become highly annoyed by unpredictable response times, so don't overdo the prebuffering if it could leave to such a situation.

- Consider clustering smaller files into one larger file if the data within those files is logically grouped and likely to be loaded in one step. Use the asynchronous mechanisms provided wherever possible.

Design

If the response times are still not acceptable after all efforts to streamline the operation of the program, you need to ask if the selected design is a good match for the medium. Design changes may be required to save a title. It is often the case that a small change in design can make a major difference in the responsiveness of the program.

If you, as a software engineer, see such a situation coming, don't hesitate to raise a flag to the producer and/or designer to the disc. The overall result can only benefit.