AIM Technical Note #61

Software Sprites in CD-I

Alty van Luijt

January 23, 1991

Architecturally, CD-I is not very similar to a "game machine." Characteristics, such as sprites and collision detection, are not supported by the hardware. Yet CD-I's powerful graphics capabilities, relatively powerful CPU, two-plane video display architecture, built-in, run-length logic, and superior audio circuitry—all under the control of a real-time operation system—can provide substantial functionality in the domain of action games, albeit at a different level of coding sophistication.

SOFTWARE SPRITES IN CD-I

Introduction

The characteristics of CD-I differ quite significantly from the archetypical "game machine. However, it must be said that some of the new 16-bit game machines move closer towards CD-I in their graphical capabilities. The most outstanding feature that distinguishes the two is the availability of "sprites" in typical game architecture. Sprites are small areas of memory that can be individually positioned on the screen without the need to alter memory contents. The CPU has only to modify some position registers; this is highly efficient—especially when multiple sprites have to be moved simultaneously. Classical personal computers are not able to support sprite operations; they have to rely on more powerful CPUs to manipulate the contents of a bit map.

CD-I falls between the game machines and personal computers in terms of dynamics. (CD-I is far superior to game machines and personal computers in image quality.) The features that allow CD-I to perform action games better than the typical PC fall in three areas:

1) In audio, CD-I has a powerful sound generation capability that is superior to anything a game or PC has to offer.

2) In video, there are two planes with transparency control. In addition, every video line is pointer-based and can, therefore, be scrolled independently.

3) Run-length decompression logic that is built into the hardware and the interrupt generation capability on any arbitrary display line offer opportunities that are not provided by a typical PC.

Generally, the interrupt-driven architecture, the real-time operating system, and the large available RAM (by game machine standards) allow programs to run efficiently in the CD-I environment.

The implications of these capabilities for a typical action game is that a simple "port" is not possible. The functionality of the game must be analyzed from the ground up, and a specific CD-I solution must be designed for each but the simplest action games. For the simplest games, where a software blitter provides adequate performance, a more direct mapping of the original game code is possible.

Blitting

In a classical PC environment, blitting in a single video plane is the only way that sprite functionality can be achieved. Some architectures use a bit map for each color. More modern architectures, such as CD-I, contain all the information that pertains to a pixel in a single byte. The performance difference between the two categories is significant. Blitting is the term that describes a process of copying successive versions of a memory-based character (usually a rectangular shape with some color defined as transparent) into a displayable piece of memory in successive locations to create animated motion. At every new step of the animated motion, the following steps are typically performed:

1) Save the background image for the rectangle that will be overwritten.

2) Copy the character shape to the display memory.

Then, at the next phase of movement:

3) Restore the background image from the previously stored version.

For CD-I, as long as sprites are not overlapping, steps #1 and #3 can be omitted by arranging the scene into background and foreground planes. The background plane is the playfield; the foreground plane contains the characters. The background plane can be set to any CD-I display mode; it is not restricted to graphics. It could be a natural image, coded as DYUV, or a run-length animated scene. The entire foreground plane can be cleared to the transparent color before the animation starts and the actual blitting is reduced to:

1) Copy the character shape to the display memory

and then, for the next phase of the movement:

2) Clear the foreground plane rectangle that is freed back to transparency. (Note: Clearing memory is much faster than copying.)

This technique makes blitting roughly twice as fast as operations in a pure bit-map environment. Furthermore, since the display area is pointer-based in CD-I, the display can be double-buffered at no additional CPU overhead. Thus, the glitches and shearing normally associated with blitting into the active display can be avoided. The fact that efficient and "clean" blitting is feasible in CD-I has led AIM to invest in developing a family of blitters.

The basic blitter, of course, is the straight-copying type. However, higher

performance versions with different color/speed tradeoffs use a more compressed format that is expanded "on the fly." These blitters are again significantly more effective that the conventional ones. In a single title, and even on a single screen, different blitter versions are used for sprites with different characteristics. The "Dark Castle" is an example of a title that uses multiple blitting methods.

Run-Length Coding

The run-length coding method offers a totally different opportunity for obtaining dynamic screen movement. This method is preferably used in combination with the two-plane properties of CD-I. A prime example of this technique is in the "Golf" disc in which a series of images of a golfer in various positions is stored in a run-length, compressed format in memory. This methodology gives better-than-video-game visual quality. This motion is constrained by the fact that run-length coding extends across full TV lines. Thus, multiple active characters can coexist in vertical separation, but not side by side inside in the same horizontal strip. Most remarkable about this technique is that there is virtually no limitation on visual quality or on the size of the moving object. The only limitation is available memory; 512K are available per plane. However, quite a few versions of the character can be stored, and, if necessary, new versions can be loaded from the compact disc.

Color Cycling

Color cycling provides a further possibility for animation; color cycling works well for repetitive patterns. In many titles, color cycling is used as a low cost way to "keep a screen active." However, it is not limited to this simple form. It works especially well for regular structures and/or periodic motion. The "Renault" discs make extensive use of this technique to animate illustrations on engine operation and the like.

Hybrids

Potentially the strongest presentations can be obtained by combining the techniques describe above. This can be done effectively, because the various techniques draw on different resources in the CD-I player. Blitting is CPU limited. Run-length coding is memory intensive, but uses almost no CPU. Color cycling is cheap overall; it uses neither much CPU, nor much memory. Blitting works best on smaller objects. With the AIM blitter arsenal, highest efficiency is obtained for the simplest shapes. Run-length is appropriate for display of a smaller number of complex objects. Thus, several types of "shoot-'em-up" games, in which there is a large spaceship or other device firing all kinds of projectiles at small, fast-moving objects are examples where a hybrid solution works best—run-length for the large objects, blitted sprites for the smaller ones. A combination of foreground run-length

animation against a background that is blitted into or is run-length, allows Laservision-like action games. "Escape from Cyber City" is an example of this category. Really superior games can be created using this combination approach. The potential for exciting new games in this category are endless.

Software Consequences

CD-RTOS provides an environment that lends itself well to real-time programming. All external events, such as data becoming available from the disc or new coordinates becoming available from the user's joystick, are handled transparently by the system hardware and software. The results are signaled to the main program in a way analogous to a software interrupt. Also, detailed timing can be made available to the CPU, because it is possible to generate a similar interrupt on any arbitrary video scan line. Of course, the only viable option is to code in assembler for time-critical sections of the games. The higher layers of software that have been developed in run-time systems generally carry too much overhead to be really useful. The Balboa cursor/hotspot mechanism might be an exception to this, because they this mechanism is highly efficient. However, we have not yet tried using this mechanism yet in a game, so this approach still needs to be proven. In CD-I, the fact that different resources need to be used to create game-like effects makes it necessary to completely rethink the implementation. In the process of doing so, it is sometimes desirable to expand on certain features that were limited in the original design by the constraints of the machines for which they were intended. The easiest way to do so is to create a richer look for the graphics and/or objects. This also allows differentiation between the more expensive CD-I machine and the typically cheaper game machines.

Conclusion

Although it is not intended as a game machine, several characteristics of CD-I make it a more suitable platform for games than traditional PCs. If the additional audio-visual capabilities of CD-I can be exploited, exciting games can be created. The future addition of full-screen, full-motion video can only enhance those capabilities further.