AMERICAN
INTERACTIVE
M E D I A
A Philips/PolyGram Corporation

AIM Technical Note #63

Display Synchronization in CD-I

Alty van Luijt February 14, 1991

CD-I has a fairly elaborate display architecture and correspondingly complex system software support. The different functions are explained in this technical note, with the emphasis on setting up the display system in such a way that glitches can be avoided. This is probably not an issue for "simple titles", where the functionality as provided by UCM is already adequate. However titles that rely on more complex imagery composited from both video planes need to pay attention to the problems raised in this note. Two solutions are described, one that relies on careful planning of transitions, while the other describes a more fundamental solution by properly synchronizing the transition with the display timing.

Table of Contents

Display Synchronization in CD-I

1) Introduction

The video functionality of CD-I as described in the Green Book represents a sophisticated, yet fairly straightforward, model. The actual underlying hardware that performs the functions implements a slightly different model in its native mode of operation. A rather complicated driver translates from the physical model onto the logical model for the first generation of players. The operation of this driver can sometimes take a longer time than one would anticipate because of the internal mapping that takes place. Also, some of the functions that control the hardware are not atomic and can lead to timing complications. So, from a pure functional point of view, programming for the CD-I display system might seem straightforward once a basic familiarity has been acquired with the concepts involved. However, programming for high performance, glitch free operation is by no means a trivial challenge.

Luckily, most simple titles show one plane at a time and make transitions between them using visual effects. These titles will never encounter more complicated behavior. It becomes manifest only when both planes are visible simultaneously, and a new display structure is activated for one of the two planes.

2) The Basic Two Plane Problem

Unlike the operation of almost any other machine, CD-I has multiple video planes. There is a grand total of four such planes if one includes the (small) cursor plane as the front most, and the Motion Video, or backdrop plane, as the rearmost plane. Nevertheless, from a software engineering point of view, the two middle planes are the most important ones because they carry all the functionality that is under software control. The fact that the total resulting video presentation is made up by the sum of activities taking place in these two planes is one of the fundamental problems that needs to be addressed in creating a glitch-free display. The underlying hardware scans the two planes and their control tables in perfect synchronization. Writing into the control structures for the two planes by the CPU is, per definition, a sequential operation. This can lead to a situation where, when the contents of one plane have been modified, the old contents of the second plane may still be present when the hardware scan reads those locations.

This is a fundamental race condition that a title engineer needs to address, either by sequencing the updates so that there is no visual glitch during the transition or by scan synchronizing the update so that the new steady state for both planes is achieved in a timely fashion before the next field controls are interpreted by the display logic.

3) The Basic Display Control Structure

Besides the two plane architecture, the other feature that sets a CD-I display structure apart from more traditional PC display architecture is the notion of control tables and the execution of a Display Control Program (DCP).

In a traditional environment a display is set up when the CPU writes into control registers. In CD-I, the hardware registers of the display logic are dynamically filled by memory-resident tables. This creates tremendous freedom for the CPU to prepare tables "off line"; these tables are then interpreted on the fly in sync with the display. This methodology enables many of the special features of CD-I to be performed without undue CPU loading. Visual effects, scrolling screens, and partial screens are all examples of the use of these tables.

There are two such tables per plane: one that is interpreted during the vertical retrace, called the Field Control Table (FCT), and a second that is interpreted during horizontal retraces, called the Line Control Table (LCT). Both are, in essence, pointer-driven, so that it is possible to prepare multiple tables and switch dynamically between them, at least in principle. In a later section of this note, it will be shown that there are some timing complications associated with the system software support for these updates.

4) Function Calls Available to Control the Display Structure

The three function calls that are relevant for the logic of the DCP are:

• DC_FLnk	Determines which LCT gets executed after the selected FCT
• DC_LLnk	Allows the creation of a linked list of LCT fragments for subscreens
• DC_Exec	Governs the overall execution of the display programs

Note that DC_Exec determines the behavior of both planes "simultaneously;" and the "Link" instructions are plane specific.

The catch in the previous statement is "simultaneously." Because the update of the pointers for the two planes done by the driver requires multiple CPU cycles, there is an inherent chance that one update has already occurred and the update for the second plane is pending when the "new" FCT is interpreted by the hardware and executed. This race condition would creafe a glitch of a single field duration.

The only way for system software to prevent this transparently to the application is when the driver is scan synchronized to update the two plane pointers at a moment when the display logic is guaranteed not to be reloading. As far as we have been able to trace, the current 1.1 Microware and Philips drivers do not do this. Since they actually determine the functionality of the players in the field, this is becoming one of the "facts of life" for a CD-I application.

How serious is this ? Here are some orders of magnitude for the timings involved: DC_Exec takes slightly more than 20 msec, DC_Flnk is in the order of 8 msec, and DC_LLnk takes roughly 2 msec. We do know that the actual pointer update instructions for the two planes are less than 100 microseconds apart, and we have requested Microware to further reduce this window in future releases. Thus, the odds of this causing a problem would be reduced below the current 10% in the totally asynchronous case. However, no matter how small the odds, it is good engineering practice to prevent a situation that can lead to these conditions altogether. The remainder of this note focuses on this.

5) Living with Asynchronous Display Updates

As described in the previous sections, there is the inherent risk of creating display glitches when relying on the DC_Exec mechanism in UCM for updating the display control structures. Fundamentally, there are two ways to make sure that the user does not notice this problem. Solution one is to make sure that at the moment of a DC_Exec the actual display is determined exclusively by the contents of a single plane before, as well as after, the execution of the DC_Exec. In this case, there is never an intermediate state where part of the logic that determines the display is already updated and the other is not; thus, there is no visible glitch. Solution two is to somehow scan synchronize the update, so that the race condition never occurs.

5.1 Visual stability

There are two areas of potential complication for visually masking this problem. One is the asynchronous execution of the two DCPs: this could cause a mismatch in the case of a partially transparent image. The other, and more serious problem, is that the image coding method for plane B is determined from the DCP in plane A. It is clear that both of these conditions need to be avoided to effectively avoid glitches. That means that an image coding method change or a content change in plane B needs to be avoided while parts of plane B are visible. So, practically speaking, what can one do if the design calls for a situation that would require this ? Let's treat this by example, rather than trying to formulate an abstract description.

Let's say that the design calls for a CLUT image over a DYUV background for a menu and that one of the selections from that menu launches a run-length animation in both planes. This is clearly a case where transparency is used both

LCT to set up the display, rather then control that particular line, a pointer mechanism cam be created that is powerful enough to be useful, efficient, and fully under application control. The penalty is the loss of the topmost video line (which is typically invisible anyway due to overscan tolerances on the typical TVs).

This model uses a fixed combined area of FCT and the first line of the LCT for planes A and B, for which the image coding method never changes, to set up the CLUT and other memory intensive operations. It also assures that the first visible line is set to black in both planes by setting ICM OFF in both planes. Then, the LCT for the second line is used to set up the real display. It contains the " DYUV start values," the "Select image coding method" instruction, as well as the "Load transparency control", "Load plane order," "Load display parameters", "Load image contribution factor," and the "Load display line start Pointer" The last entry is for the "Control table start pointer," which is reserved for system use. To achieve this, do not use the last column in the LCT, and then call DC_LLnk to do the actual update. Now double buffering takes place by rewriting these LCT entries and performing the two DC_LLnk function calls (one for each plane) in a scan synchronized fashion. Preliminary experiments with this technique indicate a reduced overhead compared to other double buffering techniques that use DC_Exec or DC_Flnk. This reduced overhead brings the timings back to values where it becomes realistic to think about scan synchronizing them. This methodology has the potential for creating a generic solution to the problem, provided that reliable scan synchronization is available.

6) Conclusion

First, the glitch phenomenon, as described in this note, is nothing to be overly concerned about. Simple titles do not suffer from it. Visually more complex titles do occasionally encounter it, but the effect is only cosmetic. The functionality of the title is not affected. Future releases of the system software reduce the odds of encountering it. Nevertheless, it is cleaner to prevent these glitches. In this note, two methodologies have been described to solve this: one is to carefully plan the visual transitions; the other is to scan synchronize the update of the display. Further work will make solutions, such as the latter, more easily accessible to the engineering community.

before and after the switch. Let's make some arbitrary allocations: the original DYUV background is in plane B; the menu selection areas are written in CLUT in plane A; the background of the run-length animation is in plane B, and the foreground animation characters are in plane A. It is clear that there is a risk that by "unlucky" timing there might be one field in the transition that has the new plane A FCT executed, while the plane B DCP is still in its old state. The one field then consists of some foreground animated character over a totally garbled background (the background is DYUV data interpreted as run-length coding: a sure recipe for psychedelic effects).

One way to treat this is to make the transition more complex so that it becomes more robust. In essence, one can replace a single transition by a series of harmless transitions. In our example, one could fade down the first menu, DC_Exec to the new situation, and fade up again. That way the screen is black when the timing problem occurs, thus, it does not translate into a visual glitch.

Alternatively, one could modify the allocation of the contents and the plane so that the buttons are in plane B with the background DYUV in plane A. Then, remove the menu buttons first so that a pure DYUV image remains with plane B off. Then, a transition could be made to the first frame of run-length animation to be background only from plane A (instead of plane B) with Plane B still off; the start of the foreground animation character moving could be postponed by one more field. Thus, a glitchy transition has been replaced by the gradual build down of one scene, followed by the build up of the next. This is much less annoying.

Finally, the best method for concealing the problem is to avoid it all together. If, in the previous example, we replace the CLUT-based menu graphics by RL7 based graphics, no switch in image coding method is required. If the planes are properly allocated so that the switching ICM is in plane A, the glitching problem can be avoided altogether.

While this might work fine for specific cases where plane allocation, etc., can be fully planned, in the generic case, where a user interrupts a presentation at an arbitrary moment, such a solution might not be viable.

5.2 Synchronizing the updates to the display

The alternative to carefully orchestrating the transitions so they are harmless is to design a transition methodology that can be scan-synchronized. It is not intuitively obvious how the application can do this in a hardware independent manner. One potential solution, which will be described here, has been suggested by Charlie Golvin. The underlying premise of this idea is to use the DC_LLnk function in a scan synchronized manner. Since the number of parameters that needs to be updated for a context switch is really limited, all essential information can be contained in a single line of the LCT. So, by sacrificing one video line and using its