DTL-H2700 Performance Analyzer / Development System

The DTL-H2700 is a new product which will soon be made available to licensed PlayStation developers. It combines a fully functional development system with a new and improved version of the PlayStation Performance Analyzer.

What is the Performance Analyzer?

The Performance Analyzer is a useful tool for determining how successfully a PlayStation application takes advantage of the system's features and available bandwidth. Specifically how it works is that it captures a record of all of the signals on the system bus, and then it converts this information into a graphic display. This allows developers to determine which aspects of an application's performance will benefit the most from fine-tuning and optimization. We'll discuss specific features and capabilities in depth a bit later.

The original version of the Performance Analyzer was not made available to developers for several reasons. First, it was hand-built and somewhat fragile. It required a dedicated PC with a specialized interface card. It was not a true development system; there were no interactive debugging capabilities and it required a bootable gold disc that would run on a standard blue test PlayStation. There was no CD emulation system, and it was not possible to test small bits of code without burning a new gold disc each time. Furthermore, there was no way to control what portion of the application was analyzed except via the imprecise mechanism of physically hitting a trigger button on the front of the unit.

Because each Performance Analyzer was constructed individually, it would have been necessary to charge in the neighborhood of $100,000 or more if they had been made available to developers. As it was, only 4 were ever made. Because they were not generally available, developers were invited to visit Sony Computer Entertainment with the necessary materials and use the Performance Analyzer under the supervision of Sony's developer support engineers.

Despite the factors that prevented the original Performance Analyzer from being made available to developers in general, it was obvious that this was an extraordinarily useful tool for developers, and would be even more so with some improvements. So now we have the DTL-2700, which combines and builds upon all of the features of the earlier Performance Analyzer as well as the DTL-2000 and DTL-2500 development system boards.

The DTL-H2700 Hardware

The DTL-H2700 is an ISA-based PC card which plugs into a standard Intel 80486 or Pentium-based computer. Actually, the DTL-H2700 is three cards connected together. Two of the cards are essentially similar to the DTL-H2000 development system boards, except that only one ISA slot is used. The third card contains the memory and other circuitry of the Performance Analyzer.

The DTL-H2700 only uses one ISA slot, but it takes up as much space as three full-length ISA cards. Also, because of the power requirements of the three boards, a separate power connector is used to draw power directly from the PC's power supply.

One significant difference from the DTL-H2000 is that the DTL-H2700 uses the same DTL-H2510 internally-mounted CD mechanism as the DTL-H2500 development system, rather than the DTL-H2010 external black box CD drive. The DTL-H2510 is the same CD mechanism as is contained in the standard PlayStation, eliminating any variations in performance or operation between the development system and a "real" PlayStation.

The DTL-H2700 Development System

As a development system, the DTL-H2700 is in most respects similar to the DTL-H2000. It contains 8 megabytes of DRAM and can be used with the CD emulator board. It uses the same software drivers as the DTL-H2000 and is identical from a programming perspective.

The DTL-H2700 Performance Analyzer

The Performance Analyzer portion of the DTL-H2700 is essentially separate from the rest of the system in most respects. The software for the Performance Analyzer operates under Windows 3.1 or Windows 95. It requires no special software drivers be installed before it is used, although the standard drivers for the DTL-H2000 may be

installed without conflict. The Performance Analyzer software will **not** work under Windows NT because it directly accesses the DTL-H2700 hardware.

The Performance Analyzer software works best with 1024x768 or higher resolution, 256 colors or more.

Below, we'll refer to the Performance Analyzer portion of the DTL-2700 and the Performance Analyzer software together as a single entity separate from the rest of the DTL-H2700. When we refer to the DTL-H2700, we'll be talking about the development system portion.

The Performance Analyzer monitors whatever software process is executing on the DTL-2700 hardware and has no facility of its own for starting an application. It presumes that whatever application is going to be analyzed is already running. This may be done by executing a CPE file, booting a gold disc in the DTL-2510 CD drive, or by booting an image from the CD emulator. These actions would be done using the standard methods and programming tools. The application may be started either before or after the Performance Analyzer software is started.

If no performance analysis is required, then the Performance Analyzer software is not required and the other programming tools may be used from within the DOS and/or Windows environments as desired.

When the Performance Analyzer software is running, it is either displaying previously captured information or it is waiting for the trigger condition to indicate that it should capture new information. There are two methods of triggering the capture. First, the user presses a switch which is plugged into the program analyzer board. Second, a particular memory location previously specified to the Performance Analyzer is accessed. This means that captures may be triggered under software control by simply accessing a particular memory location.

When the appropriate trigger is detected, the Performance Analyzer captures the signals on the bus for the predetermined length of time, specified in CPU clock cycles. The maximum capture is roughly 7.4/60ths of a second long, which is typically long enough for at least 2 game frames (at 15 fps). Games in an early stage of development which have frame rates below 10 fps may not be able to capture a single complete frame. In this case, the option of triggering under software control mentioned earlier can be used to capture a frame portion by portion.

The Performance Analyzer displays how much time is spent by things like loading the I-Cache with new instructions, performing data reads and writes, MDEC DMA transfers, etc. It shows how the GPU is being used, how data is being transferred to the SPU, and so forth.

In the next few sections, we will details some specific examples of how the Performance Analyzer can be used to detect specific problems.

CPU Efficiency Analysis

The Performance Analyzer shows how many clock cycles are spent when the CPU must load instructions from RAM into the instruction cache. Ideally, your program code should execute from the cache as often as possible, but since the cache is only 4k long that simply isn't possible. However, with a little fine tuning it's possible to arrange your code so that it executes from the cache as efficiently as possible. The Performance Analyzer shows where your program is using the cache efficiently, and where additional tuning is required for the best performance.

The Performance Analyzer also shows how many clock cycles are wasted when the CPU stalls due to inefficient ordering of instructions which read and write RAM. In many cases, simply changing the order of certain instructions can improve performance.

The Performance Analyzer also shows how much time is spent with the bus idle. Idle time on the bus can be a good thing or a bad thing, depending on WHY it is happening. For example, if the CPU or GTE is executing code from the instruction cache, and all the appropriate data is either in the scratchpad or within registers, then idle time is a good thing, because it means that RAM is not being accessed and processing is proceeding at the maximum possible speed.

On the other hand, if the bus is idle because you're not using the GPU to render anything, or you're not using the MDEC to decompress image data, or you're not doing anything with the SPU or CDROM, then idle time is a bad thing because it indicates that you're not getting the most out of the system.

The Performance Analyzer will show where you have the most idle time and where your bus bottlenecks are occurring, and help you tune your code to get the maximum possible performance.

GPU Efficiency Analysis

One of the most significant additions to the new version of the Performance Analyzer is the ability to give detailed information about the efficiency of a program's GPU usage. This allows you to determine how much processing time can be recovered by tuning the program to use the GPU more efficiently. Below are just a few examples:

You can determine how many clock cycles are spent processing null entries in your GPU primitive list. These are the unused positions in your program's ordering table (OT), and each one takes 8 clock cycles to process. This information can help you determine if your OT is too large or arranged inefficiently and if tuning your OT handling can help you save memory and improve rendering times.

It will tell you how many clock cycles are being wasted by inefficient drawing. For example, polygons which are outside the clipping area must still be processed by the GPU even though they do not affect the display, and the Performance Analyzer will tell you where these occur. It will also tell you how many stalled clock cycles are caused by backface polygons (polygons where the vertices are specified in reverse-order). Texture-mapped backface polygons are much slower because the texture information cannot be processed as efficiently.

The Performance Analyzer will also point out where texture information is not being used efficiently, such as when a large texture is used for polygons which are coming out to be only a few pixels on the display. The GPU reads 8 pixels of texture data at a time when rendering a polygon. When a polygon that is 3x3 pixels on screen specifies a 128x128 texture, the GPU must scale down the texture considerably. It must read a lot of texture data that is never used. For this example, it will have to read 8 pixels of texture data for each pixel that is drawn, which translates into a lot of wasted clock cycles. It's not a big problem for a polygon or two here and there, but when you have several hundred polygons per frame doing this sort of thing, the wasted time can become significant.