Performance Analyzer Technical Reference

Table of Contents

List of Figures

About This Manual

This manual is the latest release of instructions relating to the PlayStation® Performance Analyzer as of Run-Time Library release 4.3. The purpose of this manual is to describe how to measure software performance and interpret the results using the Performance Analyser.

Changes Since Last Release

There have been no substantial changes to this document since its last release.

Related Documentation

This manual should be read in conjunction with the *Performance Analyzer User Guide*, which provides general instructions on the use of the Performance Analyzer.

Developer Reference Series

This manual is part of the *Developer Reference Series*, a series of technical reference volumes covering all aspects of PlayStation development. The complete series is listed below:

Manual	Description
PlayStation Hardware	Describes the PlayStation hardware architecture and overviews its subsystems.
PlayStation Operating System	Describes the PlayStation operating system and related programming fundamentals.
Run-Time Library Overview	Describes the structure and purpose of the run-time libraries provided for PlayStation software development.
Run-Time Library Reference	Defines all available PlayStation run-time library functions, macros and structures.
Inline Programming Reference	Describes in-line programming using DMPSX, GTE inline macro and GTE register information.
SDevTC Development Environment	Describes the SDevTC (formerly "Psy-Q") Development Environment for PlayStation software development.
3D Graphics Tools	Describes how to use the PlayStation 3D Graphics Tools, including the animation and material editors.
Sprite Editor	Describes the Sprite Editor tool for creating sprite data and background picture components.
Sound Artist Tool	Provides installation and operation instructions for the DTL-H800 Sound Artist Board and explains how to use the Sound Artist Tool software.
File Formats	Describes all native PlayStation data formats.

Data Conversion Utilities	Describes all available PlayStation data conversion utilities, including both stand-alone and plug-in programs.
CD Emulator	Provides installation and operation instructions for the CD Emulator subsystem and related software.
CD-ROM Generator	Describes how to use the CD-ROM Generator software to write CD-R discs.
Performance Analyzer User Guide	Provides general instructions for using the Performance Analyzer software.
Performance Analyzer Technical Reference	Describes how to measure software performance and interpret the results using the Performance Analyzer.
DTL-H2000 Installation and Operation	Provides installation and operation instructions for the DTL-H2000 Development System.
DTL-H2500/2700 Installation and Operation	Provides installation and operation instructions for the DTL-H2500/H2700 Development Systems.

Typographic Conventions

Certain Typographic Conventions are used through out this manual to clarify the meaning of the text. The following conventions apply to all narrative text except for structure and function descriptions:

Convention	Meaning
courier	Indicates literal program code.
Bold	Indicates a document, chapter or section title.

The following conventions apply within structure and function descriptions only:

Convention	Meaning
Medium Bold	Denotes structure or function types and names.
Italic	Denotes function arguments and structure members.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America	*In North America*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd., 2nd floor Foster City, CA 94404 Tel: (650) 655-8000	E-mail: DevTech_Support@playstation.sony.com Web: http://www.scea.sony.com/dev Developer Support Hotline: (650) 655-8181 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In Europe	*In Europe*
Attn: Production Coordinator Sony Computer Entertainment Europe Waverley House 7-12 Noel Street London W1V 4HH Tel: +44 (0) 171 447 1600	E-mail: dev_support@playstation.co.uk Web: https://www-s.playstation.co.uk Developer Support Hotline: +44 (0) 171 447 1680 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT or BST/BDT)

Introduction

The performance analyzer visualizes information such as bus traffic by sampling the signals in the PlayStation. Some expertise is required to tune a program based on such information. This manual details the expertise required. This manual assumes that the user has read the *Performance Analyzer User Guide* and is familiar with the method of using the performance analyzer. This manual also assumes that the user is familiar with the architecture and programming of the PlayStation, including programming techniques specific to high-speed processing in the PlayStation.

What Can the Performance Analyzer Do?

The performance analyzer enables its user to obtain information processed by the CPU and GPU. Based on this information, the user can use programs to determine, for example, where the CPU is stalling, or the cause of reduced drawing performance. However, the information obtained from the performance analyzer is, in itself, not enough to solve all the problems that may occur. For example, the performance analyzer can detect cache misses, duplicate read accesses and similar problems, but cannot indicate whether the actual algorithm of the program is satisfactory. Some programs may be capable of much faster processing if the problem-solving techniques that they apply, including their algorithms, are modified. For example, suppose that the user always wants to process all the circuit data of a racing game. This would result in there being too much global data for the CPU to process for the polygons to be displayed. Tuning alone cannot overcome this CPU bottleneck. To overcome this problem, high-speed processing should be implemented by the application of a problem solving technique -- an example would be to divide the global data into groups to enable efficient data access. Unfortunately, the performance analyzer cannot automatically analyze problem solving techniques to identify such problems. Therefore, the programmer is responsible for determining whether a tuning technique is appropriate. The performance analyzer should be used as a measuring tool to help the programmer make this decision.

The performance analyzer samples hardware signals directly, allowing it to double as a debugger. Since the performance analyzer cannot measure signals in the instruction cache, scratch pad, and other devices inside the chip, however, its use is limited as an alternative to a full-featured debugger or in-circuit emulator.

In summary, the performance analyzer can be used to:

- Determine the degree to which the current algorithm can improve performance, and identify those locations where problems have arisen.
- Determine the currently available processing margin and how many more polygons could be displayed with that margin, if any.
- Measure the processing speed of each candidate technique while considering which algorithm to use, and also in the graphic design phase.
- Measure undesirable phenomena caused in real time by using the trigger function.

Flow of Diagnosis

The PlayStation includes devices such as a CPU, GTE, GPU, MDEC, and DMA controller. All operate independently of each other. The PlayStation can be thought of as a system in which a CPU and GPU operate in parallel. In many cases, therefore, programs that run on the PlayStation are developed using the double-buffer method so that a CPU process and GPU process can run concurrently to enable higher-speed processing. This means that the tuning of the CPU must be balanced with that of the GPU to achieve high-speed processing. By monitoring the CPU and GPU buses, the performance analyzer visualizes the processing states of the devices through main RAM bus analysis and video memory bus analysis.

Based on the above information, a program is diagnosed using the performance analyzer. The flow of the diagnosis is outlined below.

Measurement

Perform measurement of degraded processing, particularly of an overloaded scene or scene to be improved. Detect degraded scenes using the trigger function.

Determining whether the bottleneck is in the CPU or GPU

Upon the completion of measurement, the results are displayed. From the main RAM bus analysis and video RAM bus analysis, find the processing end points of a CPU process and GPU process. From the processing end points, determine the process to be tuned.

When a CPU process is to be tuned

Collecting a total statistical amount to determine whether a desired performance improvement can be achieved by tuning

Align markers M1 and M2 with the start and end points of a CPU process, respectively, to collect statistical information. From an estimate of the number of CPU stall cycles, check whether the desired improvement can be made using the available tuning methods. If the check reveals that an improvement can be made, perform tuning as described below.

Finding the cause of an instruction cache miss

If an instruction cache miss produces a long CPU stall time, make improvements mainly where a stable pattern endures for a long time. A portion with a stable pattern is regarded as performing loop processing, and further improvement can be expected with less work. (This concept applies to improvements in data read/write access, detailed later.) Enlarge such a section, read the symbol information, then detect the global symbols accessed in that section to identify a function that causes a conflict in the instruction cache. Then, perform improvement by changing the address of such a function, or by using inline expansion or DMPSX. Note, however, that the method of changing the address of a function should be employed in the last stage because the method is affected by modifications to the program source code.

Detecting duplicate reads from main RAM

The PlayStation has no data cache. So, when a read access is made to main RAM as a result of a load instruction, the CPU stalls for four cycles. This means that duplicate memory access should be avoided. Instead, the scratch pad or a register should be used whenever possible. The performance analyzer displays the read/write penalty "duplicate read" for a portion in which multiple read accesses have occurred successively with no write operation performed. The ratio of the area of such a portion is regarded as representing a CPU stall time caused by duplicate reads. For higher-speed processing, emphasis should be placed on finding such a section.

Some causes are listed below, together with the corresponding countermeasures.

- Move global data to the scratch pad before processing the global data.

- If a stack area is accessed, pass the argument(s) of a function to a register after reducing the nesting level and the number of arguments. Or, allocate a stack in the scratch pad. (In the latter case, however, note that the program may crash if the scratch pad overflows.)

- If a long expression is coded, some compilers may allocate a temporary variable in a stack area. This is because such compilers assume no penalty in stack area access. In this case, decompose a long expression explicitly, and use a register variable instead of a temporary variable. Or, allocate a temporary variable in the scratch pad.

- Multiple half-word or byte-data accesses to adjacent areas eventually become multiple long-word accesses. So, arrange the processing such that memory is accessed once on a long-word basis, with conversion from long word to half-word or byte data being made between register variables.

Detecting a write buffer flush penalty

With the "PlayStation", a data write from the CPU to main RAM is performed through a four-stage write buffer. This means that, usually, no penalty is incurred until the write buffer is full. If a write access is immediately followed by a read access, however, the main RAM bus is not released until the write buffer has been entirely flushed. In this case, the CPU stalls for about four cycles. The performance analyzer indicates such a stall time by the write buffer penalty "flush penalty." Perform analysis, focusing on those portions that incur a long stall time, then improve the efficiency by, for example, changing the order of reads and writes, and moving an instruction, if possible, to a point after a write access.

If a CPU write cycle is immediately followed by another access on the main RAM bus, this analysis may indicate a four-cycle penalty even when the CPU is actually executing another instruction. Thus, a penalty may be indicated when no stall has occurred. Therefore, assume that penalty indications merely represent the possibility of the CPU having stalled due to flushing.

A flush penalty occurs, for example, when store and load instructions for the main RAM spaces are executed alternately. In this case, the number of stall cycles can be reduced significantly by performing four write accesses at a time.

When a GPU process is to be tuned

Checking for null packets

GPU packet analysis performs a check for null packets. If successive null packets are detected, the CPU tends to stall because drawing stops and the main RAM bus is occupied accordingly. This problem often occurs when a background packet is placed at the start of the ordering table, or when a space is drawn which has no object at a given depth. If idle times caused by null packets cannot be ignored, place the background packet next to a polygon placed at the far end to start drawing there, or use multiple ordering tables.

Checking the drawing start point

Drawing is started by calling a function such as GsDrawOt. Check whether time-consuming processing is inserted between the establishment of V blank synchronization (with a function such as VSync) and the start of drawing.

Checking areas with a low drawing efficiency

A video RAM bus analysis indicates the amount of data being transferred over the video RAM bus. A higher value represents a higher transfer rate. A height approaching 100% is indicated for a transfer rate of 32 bits per clock cycle. On the other hand, a half height represents a maximum rate, for example, in the drawing of a polygon. Actually, however, the write transfer rate (indicated in green) is reduced for causes such as texture reading, as described later. Check for an extremely low green pattern. Enlarge such a pattern, if any, to determine the cause.

Detecting a texture cache miss

In texture mapping, the GPU moves texture data to the internal texture cache before starting drawing. The size of the texture cache is limited, however, so that a cache miss may occur frequently for polygons with a large texture or for far polygons with an excessively high texture resolution. If a cache miss occurs, the GPU stops writing to video RAM, and loads texture data into the texture cache. If video RAM read/write cycle switching occurs frequently, an extremely low transfer rate results. If a decrease in the drawing efficiency is caused by texture cache misses, enclose the polygon with markers M1 and M2, then check the write area and read area with the video RAM viewer to determine the size and direction of the drawn polygon, as well as the access roughness, direction, and color representing an access frequency of the texture area. Then, perform cause analysis as described below.

- For one polygon, the texture area accessed for reads is extremely large.
  This means that the texture data is too large to be held in the cache. When texture data is to be shared by multiple polygons, the processing can be speeded up if the texture data can be held on the cache. As described later, a decrease in processing speed, depending on the direction of

rotation, of a polygon can be avoided by reducing the size of the texture data, even when the data is used only once. Reduce the sizes of the polygon and texture data, or use 4-bit texture data.

- For a small write area, a large texture read area is used.
  This means that the texture resolution is too high. With the video RAM viewer, a texture area access pattern is represented by thin horizontal parallel lines. Namely, when a texture cache miss occurs, 64-bit texture cells arranged horizontally and extending to the right from the texture area are read, regardless of the texture resolution. This read operation results in a thin horizontal line. Vertically, however, texture cells are not read continuously, but are skipped. Thus, an access pattern like that described above results.
  For correction, reduce the texture resolution. If the resolution varies over a wide range, use mip-mapping or mip-modeling.

- The texture access area is long vertically. The texture read frequency is greater than the drawing frequency.
  If a texture cache miss occurs, texture cells are read horizontally as explained above. When a vertical bar is drawn, for example, not all of the read texture cells may be used. In such a case, a higher efficiency may be achieved by placing the texture data horizontally.
  With a rotating polygon, the texture read efficiency changes. A poor efficiency results particularly when the texture read direction differs from the drawing direction by 90 degrees. In this case, when the drawing section of a polygon is viewed using the video RAM viewer, the frequency of reading from the texture area tends to be displayed in a color that represents a higher frequency than that of writing to the drawing area. Several countermeasures can be applied. Divide the polygon so that the polygon data can be held in the cache. Read from the texture area by rotating the texture pattern so as to ensure efficient read access at all times. Or, provide multiple texture patterns to match the directions of rotation and to switch between them.

Detecting CLUT switching

Compared with an 8-bit texture, a 4-bit texture allows double the number of texture cells to be held in the texture cache. However, a 4-bit texture supports only a limited number of colors, so frequent CLUT switching will often be required. The "PlayStation" uses the Z sort method for drawing, such that CLUT control cannot be applied explicitly. If CLUT switching occurs frequently with the same Z value, the time required for CLUT switching may become so large that it can not be ignored. With the performance analyzer, CLUT reads, when displayed, are colored by the video RAM bus analysis, enabling CLUT switching to be identified. If the texture resolution is high, and texture cells are used on a skipping basis, many texture cells that are not used may be read even when an attempt is made to improve the cache efficiency by using a 4-bit texture. If this occurs, the use of a 4-bit texture does not speed up processing, compared with that possible with an 8-bit texture. If the penalty incurred by CLUT switching cannot be ignored in such a case, the use of an 8-bit texture with an increased number of colors can eliminate CLUT switching, thus speeding up drawing.

Checking whether the resolution of the ordering table is too high

If the resolution of the ordering table is too high, an excessive number of null packets will be read, such that the drawing efficiency decreases. Check this point carefully.

Detecting transparent pixels

When a tree is to be drawn, for example, this can be done by creating a texture with a transparent color and representing the tree with one quadrangle polygon. This method is not efficient, however, because the drawing of a transparent color, including the reading of a transparent texture and the drawing of pixels, requires as much time as the drawing of ordinary pixels. So, divide the object in such a way that the leaves are represented by triangle polygons, and the trunk by thin rectangles, such that the transparent area is minimized.

Detecting a GPU preprocessing bottleneck

GPU processing can be divided into two steps: preprocessing for converting packet data to parameters required for the drawing engine internal to the GPU, and drawing processing. The time required for drawing

processing is roughly proportional to the drawing area. However, the preprocessing time depends not on the drawing area, but on the type of the polygon. So, for a small polygon placed at the far end, most of the GPU processing time is generally used for preprocessing. Thus, preprocessing tends to reduce the drawing efficiency. In particular, when there are many small polygons with a Gouraud texture that require much preprocessing, the drawing efficiency drops considerably. Using the performance analyzer, such a preprocessing bottleneck can be detected by applying GPU packet analysis. A portion in which packets requiring much preprocessing have high patterns involves many small polygons that tend to cause a preprocessing bottleneck. If polygons that have small drawing areas and which require much preprocessing need not be used, replace such polygons with other polygons that require relatively little preprocessing when creating packets.

Detecting polygons with no drawing area, back-face polygons, and polygons subject to GPU clipping

A polygon that has no area requires longer preprocessing time than ordinary polygons do. So, before packets are registered in the ordering table, the areas of the constituent polygons should be checked; this check also serves to prevent useless polygons from being sent to the GPU.
Check also whether there are any back-face polygons for which normal clipping is not performed.
The GPU can perform two-dimensional clipping by hardware. However, a polygon that reaches the left and upper boundaries of the screen requires the same amount of processing time as when the polygon is not clipped. If there are many such polygons, the drawing efficiency decreases. The performance analyzer displays the distribution of these polygons by using polygon penalties.
To check for polygons with no area, an outer product value returned from the GTE function can be used. Note, however, that when a quadrangle polygon is checked, the area of one of the two partial triangles is returned, and a gap can result.

Checking the time required to draw a background

Usually, a background is drawn by using sprites. However, a longer drawing time may be consumed by duplicate drawing or drawing using polygons. Using video RAM bus analysis, check whether the time required to draw a background is sufficiently short.

The typical flow of tuning using the performance analyzer is described above. This method may not be able to solve many other problems, which may be associated with the use of libraries, or may be specific to individual programs. In such a case, contact the development support section.

Measurement Techniques

With the performance analyzer, a V-blank interrupt serves as a reference point for measurement data. When a program with one frame containing 2 V (two V-blank interrupts) or more is to be measured, the first reference point does not always represent the start point of the main loop. To store data by matching a reference point with the start point of frame processing, use one of the methods described below.

Methods for storing data by matching a reference point with the start point of frame processing:

a) Using synchronized data after repeated measurements
This is the simplest method. When a longer measurement time is to be set because of a low frame rate that is caused, for example, by degraded processing, the probability of start point synchronization is reduced. Accordingly, the measurement will have to be repeated. After measuring the data, stop the data transfer before it ends. Then, display the results of main RAM bus and video RAM bus analysis for 100,000 cycles. At the start of a loop, a pattern for clearing the ordering table occurs on the main RAM bus, and a pattern for clearing the background occurs on the video RAM bus. Continue to take these measurements until these patterns are obtained. When these patterns are obtained, do not take a measurement, but instead transfer data to obtain the entire data.

b) Sampling for a longer period of time, then clipping and saving the required portion
Measure the section to be measured by doubling the number of V blanks, then clip and save the required portion. With a program that has a low frame rate, however, a measurement section may be too large to be held in the memory of the performance analyzer. Moreover, if measurement is repeated several times, the data transfer time is doubled, thus resulting in reduced efficiency.

c) Using the trigger function
The trigger function of the performance analyzer can be used. For example, GsDrawOt (executed after VSync in the main loop) is used as a trigger address. After reading the symbol information, check the main RAM access address item in the trigger setting dialog box, then select a function address to be used as a trigger condition. An instruction cache miss must occur to enable the performance analyzer to detect a selected address. When addresses are accessed in the main loop, however, one cache miss is expected to occur for each frame, thus enabling synchronization to be established.

The above methods may not be suitable for measuring a phenomenon that occurs only rarely. For example, if the frame rate drops in a rare case, the pinpoint measurement of such a phenomenon should be performed. For this purpose, use a counter, for example, to find the time required to process the frame before calling VSync. Then, set up the processing such that a particular global symbol in main RAM is accessed when the frame rate drops. Namely, modify the program as follows:

```
#define HCountThreshold    525   /* frame length in H count (needs an
                                     adjustment) */
volatile long ReqTrigger;
main()
{
      ...
      for(;;){  /* main loop */
              ...
              /* compare H-counts since the last VSync exited */
              if(VSync(1) > HCountThreshold)
                    ReqTrigger = 0;
              VSync(0);
              ...
      }
}
```

Thus, frame rate drop can be measured by setting an access to the ReqTrigger variable as a trigger condition, and setting NFV of the trigger condition to the number of V blanks applied to one frame. By modifying a program as described above, a variety of phenomena can be measured using the trigger function.

Interpreting Measured Data

This section describes how to interpret the measured data.

Figure 1: Motortoon Grand Prix 2 (scene 1)
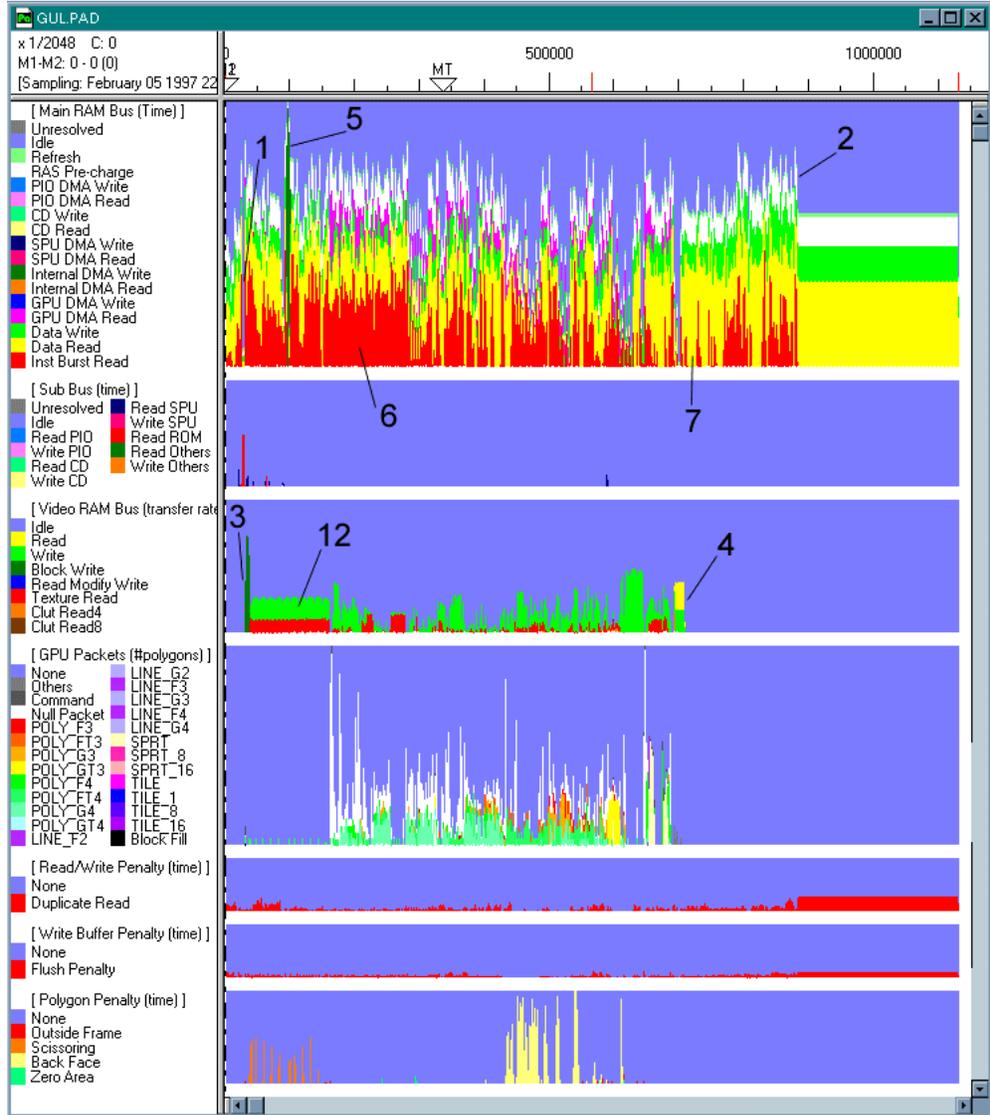
Figure 2: Motortoon Grand Prix 2 (Scene 2)
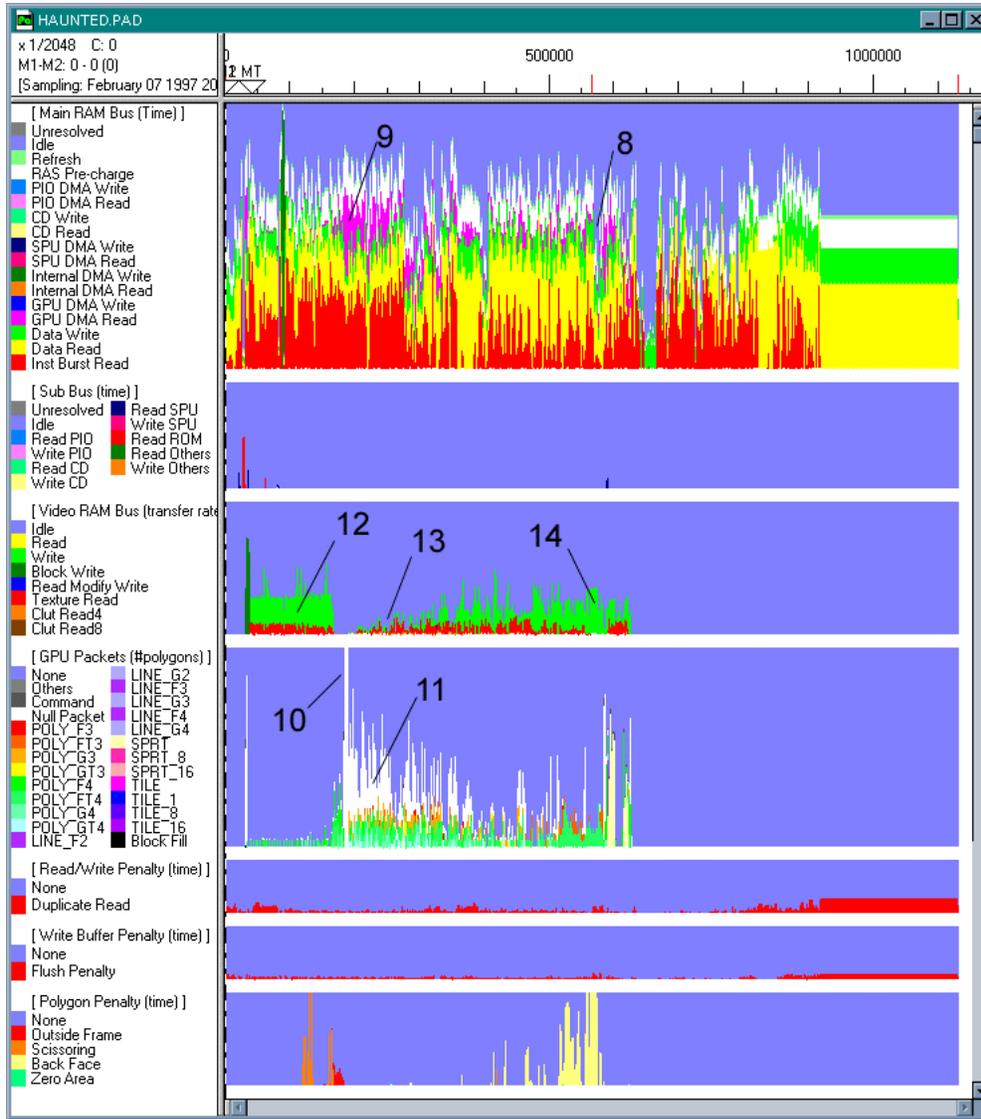
Figures 1 and 2 show the results of measuring the car racing game, Motortoon Grand Prix 2. This program employs the double-buffer method, a standard PlayStation programming technique. Analysis indicate that the main RAM bus and video RAM bus are accessed at the same time so that a CPU process and GPU process are performed concurrently. In Figures 1 and 2, a 2V section is measured; the V blank positions are indicated by red lines on the upper ruler. (Figures 1 and 2 are based on NTSC. For PAL, these red lines are spaced further apart.) How to read the patterns numbered in Figures 1 and 2 is described below.

Reading Analysis

Start of a CPU process

Figure 3: V-Blank Interrupt
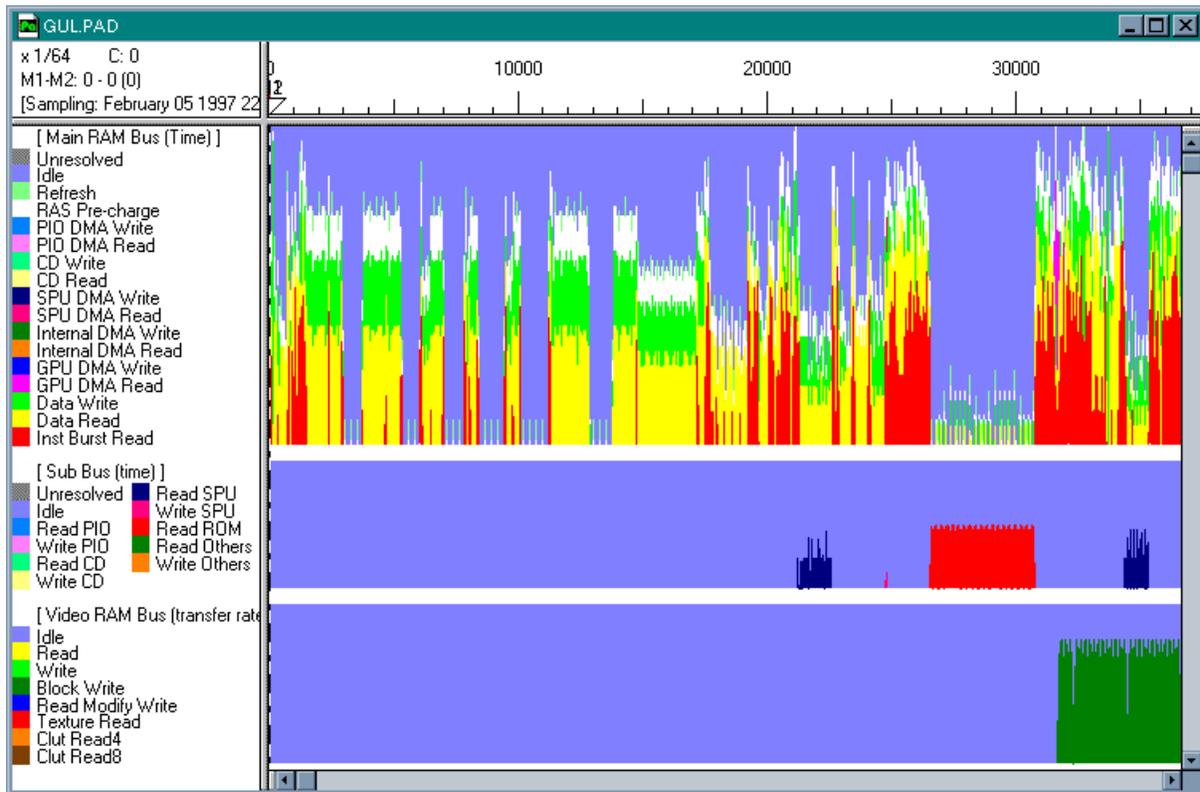
A CPU process starts when the VSync function ends in the program. The measurement data first indicates an interrupt routine caused by a V-blank interrupt. In Figure 3, the starting portion is enlarged. As can be seen from Figure 3, about 20,000 clock cycles are required for interrupt routine processing, after which the user program is executed. In Figure 1, this point acts as the start point of a CPU process. (This means that symbol main is the first point to be accessed.)

End of a CPU process (detection of VSync wait state)

The end of a CPU process is not detected automatically. However, the end point can be obtained by finding a stable read/write pattern that appears at the end of the processing. This is enabled by the VSync function polling the variables set by the interrupt routine. Such a stable pattern often represents loop processing. The end of a CPU process corresponds to the start of a pattern of the VSync function. Usually, the VSync function operates on the cache, so that no red pattern, representing a cache miss, appears.

Start of a GPU process (drawing)

The start of drawing is represented by the first access revealed by video RAM bus analysis.

End of a GPU process (drawing)

The end of drawing is represented by the last access revealed by video RAM bus analysis.

Clearing of the ordering table

The ordering table is cleared using a function such as ClearOT. The DMA controller in the CPU chip writes a long word to main RAM in each clock cycle. Thus, a high peak appears in the first half of the processing.

Instruction cache miss

A burst read from memory, caused by an instruction cache miss, is indicated by a red pattern.

On-cache pattern

If the program causes no instruction cache miss, no red pattern is displayed, as indicated here. In this case, the CPU operates efficiently without stalling.

Interrupt

Usually, an interrupt is a V-blank interrupt. Other types of interrupts, such as sound interrupts and timer interrupts, can be generated. When an interrupt is generated, an instruction cache miss occurs and a red spike-like pattern is produced, as indicated here. If such an interrupt is generated frequently, the main task processing is impeded. So, be careful if such a pattern is detected frequently.

GPU packet read

The GPU usually reads packets from main RAM by means of DMA transfer. This pattern is represented in pink. When more data is transferred, a thicker pattern results. Thus, main RAM bus access by the CPU is impeded, causing the CPU to stall.

Null packet

A null packet is a packet for which there is no entry on the ordering table. If null packets occur in succession, no drawing packet is transferred to the GPU, such that drawing is delayed accordingly. GPU packet analysis uses a white pattern to represent null packets. Null packets appearing in succession not only delay drawing by the GPU, but also cause frequent useless GPU packet reads, as can be seen from a main RAM bus analysis; thus, the availability of the main RAM bus becomes very low.

Ordering table resolution

A white pattern mixed with polygon packets like this indicates reduced drawing efficiency, caused by the ordering table resolution being too high. By means of video RAM bus analysis, enlarge and check this portion to determine the adverse effect of high resolution on the processing.

Background drawing (texture mapping)

Figure 4: Background Section Specification
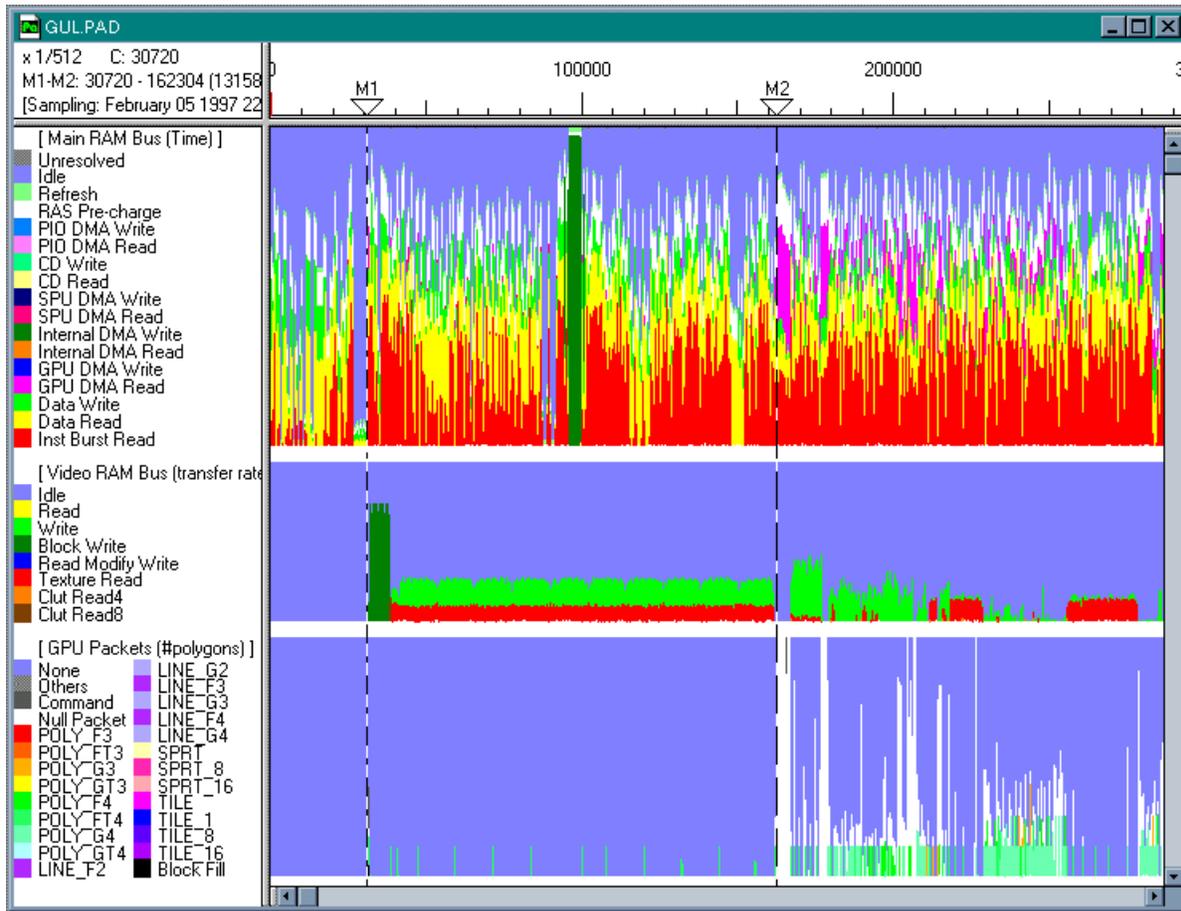
Figure 5: Background Drawing
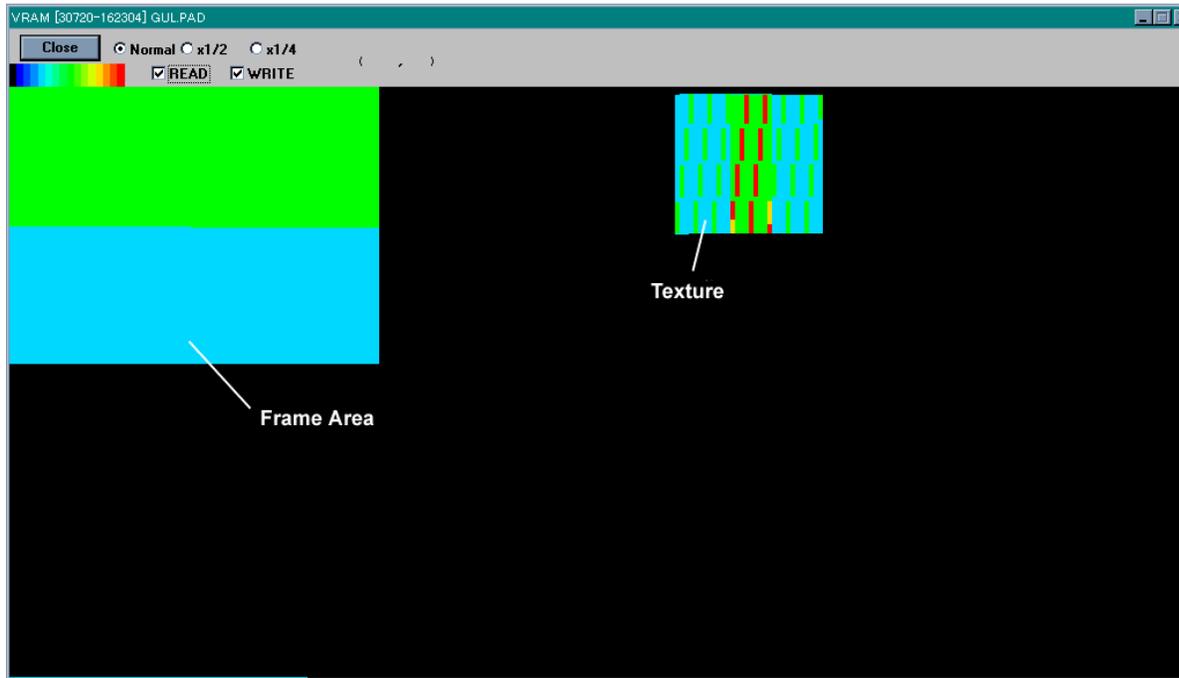
To determine a background boundary, enclose the first pattern in the video RAM bus analysis with the M1 and M2 markers as shown in Figure 4. Then, obtain the drawing area by executing the video RAM viewer command, as shown in Figure 5, and check that a background is drawn. An access frequency is represented by a color, enabling a doubly drawn area to be detected. If the resolution of the texture is too high in background drawing, video RAM bus analysis indicates texture reads in red and a lower write pattern in green. If background drawing takes a long time, tuning should take this point into consideration.

Pattern exhibiting low drawing efficiency

A low green pattern like that in this portion represents a low drawing transfer rate. For troubleshooting, enlarge such a portion.

Pattern exhibiting a high drawing efficiency

Efficient drawing is indicated by a pattern like this. No texture reads or CLUT reads are performed. Moreover, the time required for polygon preprocessing can be ignored, relative to the drawing time, and a high green write pattern is indicated. In polygon drawing, the pattern height can increase by up to 50%.

Semi-transparent polygon

A semi-transparent polygon is represented by a navy blue pattern. Note that the drawing of a semi-transparent polygon takes three times longer than an ordinary write.

**When a GPU process is longer than a CPU process**

Figure 6: When a GPU Process is Longer Than a CPU Process
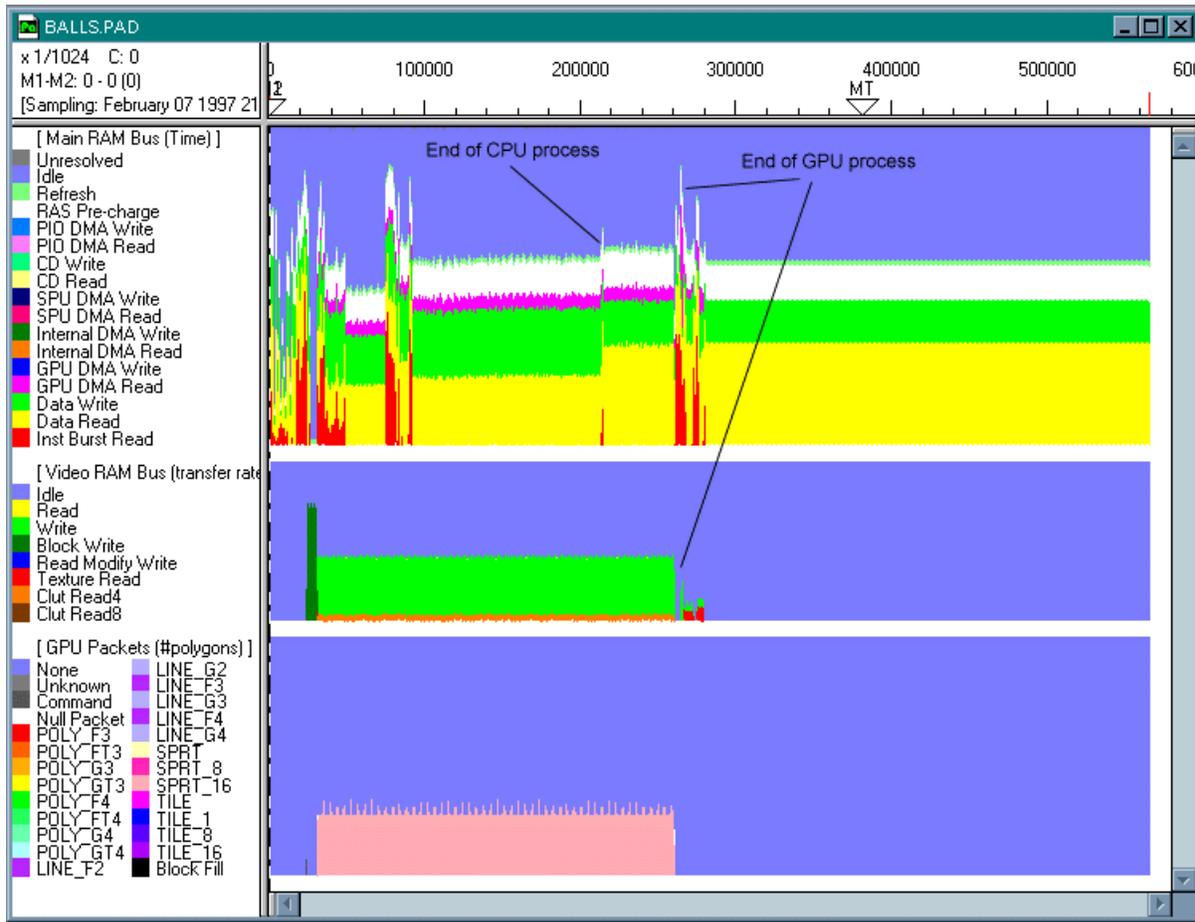
Figure 6 shows a GPU process that ends after a CPU process ends. As shown in Figure 6, even when the CPU is performing loop processing in the VSync function, GPU packets are read on the main RAM bus. So, the VSync wait pattern does not appear in a stable manner when compared with Figure 1. In this case, however, the CPU has ended its processing. Moreover, it can be observed that an interrupt for drawing termination was generated upon the completion of drawing, and a spike pattern representing an instruction cache miss due to interrupt handling occurred on the main RAM bus.

Next, an example of measuring a streaming program is shown in Figure 7. Each numbered pattern is explained below.

Figure 7: Streaming
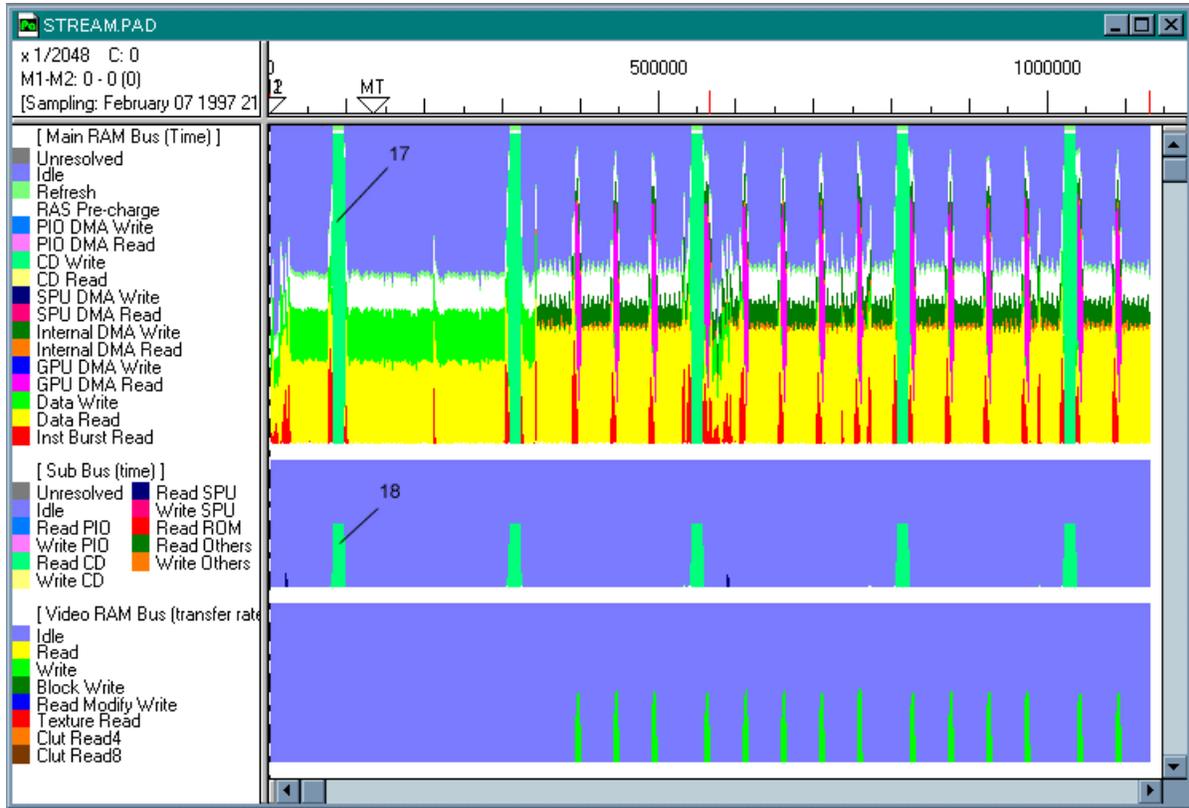


DMA transfer from the CD to main RAM

A pattern like this occurs when data is transferred from the CD to main RAM. A high pattern is displayed. However, main RAM bus analysis indicates the period during which the main RAM bus is occupied. This means that data is not always transferred in each cycle. Data transfer from the CD occupies the main RAM bus for a long time although the amount of data is not large. If the CPU attempts to access the main RAM bus while data is being transferred from the CD, the CPU will stall for a long time.

CD read

The CD device is connected to the sub-bus, another bus used by peripheral devices. So, CD reads are displayed by a sub-bus analysis like this.

Figure 8 shows an enlargement of part of the streaming. The patterns described below are observed.

Figure 8: Streaming (enlarged)
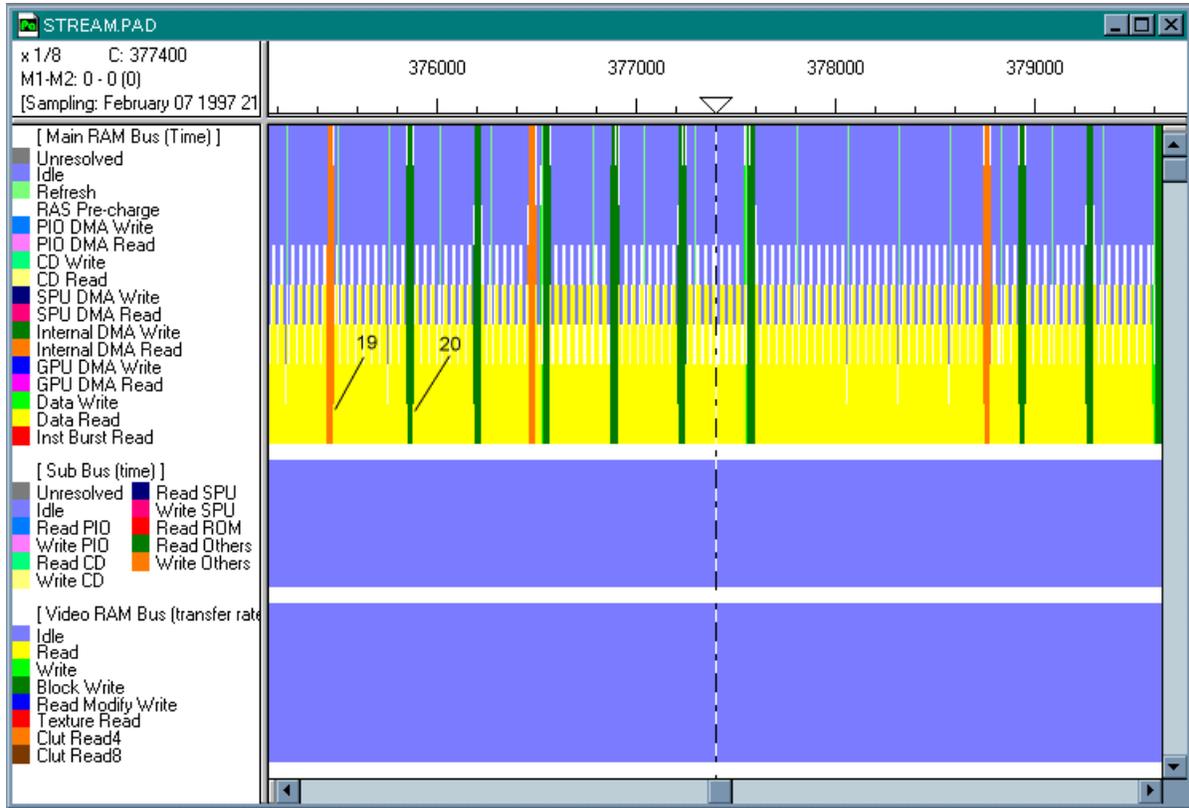
Transfer from main RAM to MDEC

Data transfer from main RAM to MDEC is represented by an orange pattern. Usually, compressed data is transferred.

Transfer from MDEC to main RAM

Transfer from MDEC to main RAM is regarded as an internal DMA write transfer, so that a dark green pattern appears in the same way as when clearing the ordering table. MDEC expands data, such that data transferred from MDEC to main RAM is decompressed image data.

Measurement of a total statistical amount

Figure 9: Statistical Information (CPU Process)

```
STATISTICS GUL.PAD [30720-882776]                                    _ □ ✕

GUL.PAD [Sampling: February 05 1997 22:59:54]
Range: 30720 - 882776

Main Memory Bus:
                  Time(%) Bytes   Speed(MB/sec)  Clock Cycles/word  Estimated CPU Stall Cycles
       unresolved   0.0    ----       ----            ----              --------
             IDLE  38.8    ----       ----            ----              --------
          REFRESH   1.6    ----       ----            ----              --------
     RAS PRECHARGE   9.9    ----       ----            ----              --------
    PIO DMA WRITE   0.0      0        ----            ----              --------
     PIO DMA READ   0.0      0        ----            ----              --------
     CD DMA WRITE   0.0      0        ----            ----              --------
      CD DMA READ   0.0      0        ----            ----              --------
    SPU DMA WRITE   0.0      0        ----            ----              --------
     SPU DMA READ   0.0    280       58.40            2.3              --------
Internal DMA WRITE   0.6  18432     131.55            1.0              --------
Internal DMA READ   0.0      0        ----            ----              --------
    GPU DMA WRITE   0.0      0        ----            ----              --------
     GPU DMA READ   4.0  53064       53.38            2.5              --------
       DATA WRITE   7.9  75727       38.26            3.5              --------
        DATA READ  20.5 174520       34.00            4.0                174520
     I-BURST READ  16.8 321912       76.40            1.8                 81864
```

Statistical information can be obtained by placing markers M1 and M2 at the start and end points of a CPU process, respectively. Figure 9 shows the statistical information thus obtained, which includes the estimated number of CPU stall cycles for the program. By referring to that value, determine the type of tuning required. If the desired processing speed cannot be obtained even after the total number of CPU stall cycles is reduced, another solution should be found by determining which processing is overloaded.

Next, place markers M1 and M2 at the start and end points of a GPU process, respectively, then measure the statistical amounts in the same way (Figure 10). Information displayed here shows the number of polygons and sprites which the GPU processed. Such information can also be used to check whether the number of polygons to be displayed is adequate.

Figure 10: Statistical Information (GPU Process)
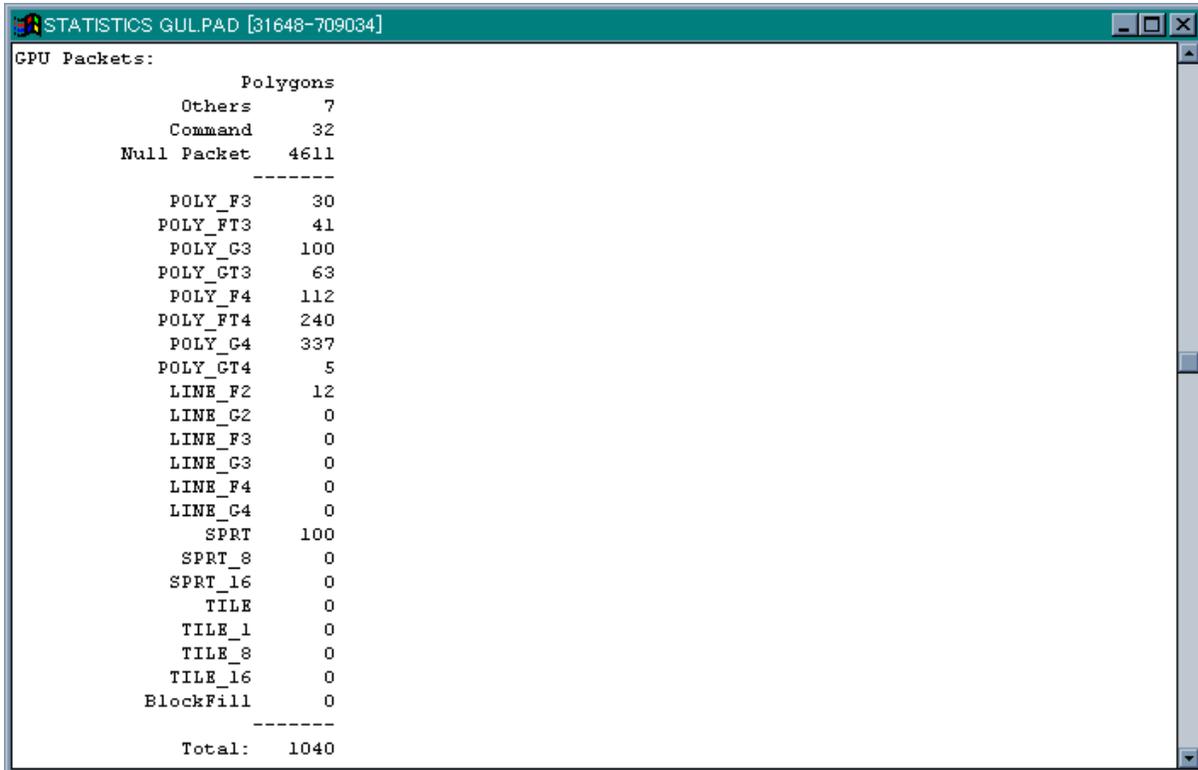
```
STATISTICS GUL.PAD [31648-709034]
GPU Packets:
                Polygons
        Others        7
       Command       32
    Null Packet    4611
                -------
       POLY_F3       30
      POLY_FT3       41
       POLY_G3      100
      POLY_GT3       63
       POLY_F4      112
      POLY_FT4      240
       POLY_G4      337
      POLY_GT4        5
       LINE_F2       12
       LINE_G2        0
       LINE_F3        0
       LINE_G3        0
       LINE_F4        0
       LINE_G4        0
          SPRT      100
        SPRT_8        0
       SPRT_16        0
          TILE        0
        TILE_1        0
        TILE_8        0
       TILE_16        0
     BlockFill        0
                -------
        Total:     1040
```

Details of Analysis

The methods of analysis are detailed below.

Detection of the Instruction Cache Miss Function

Figure 11: zimen\tuto5.cpe
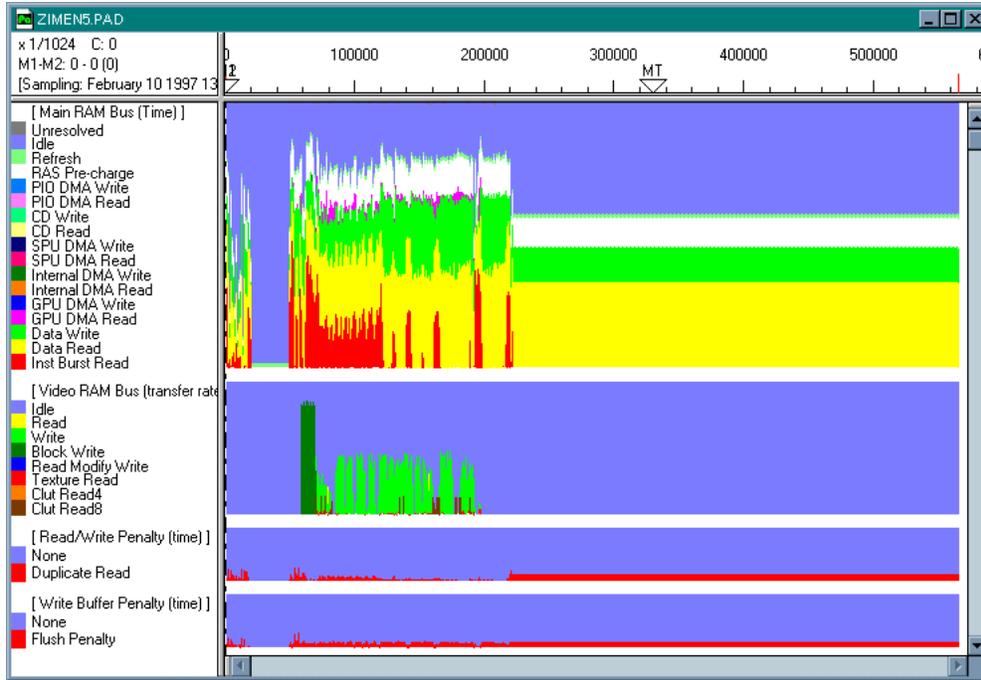
Figure 11 shows the results of measuring the sample program, *psx\sample\graphics\zimen\tuto5.cpe*, provided on the library CD-ROM. A 1V section measurement is made, and a main RAM bus analysis indicates a cache miss, in red, in the first half. We can determine the function that caused this cache miss. First, position the cursor to the area where the cache miss occurred, and enlarge that area. Then, the patterns shown in Figure 12 are obtained.

Figure 12: zimen\tuto5.cpe (Instruction Cache Miss Portion Enlarged)
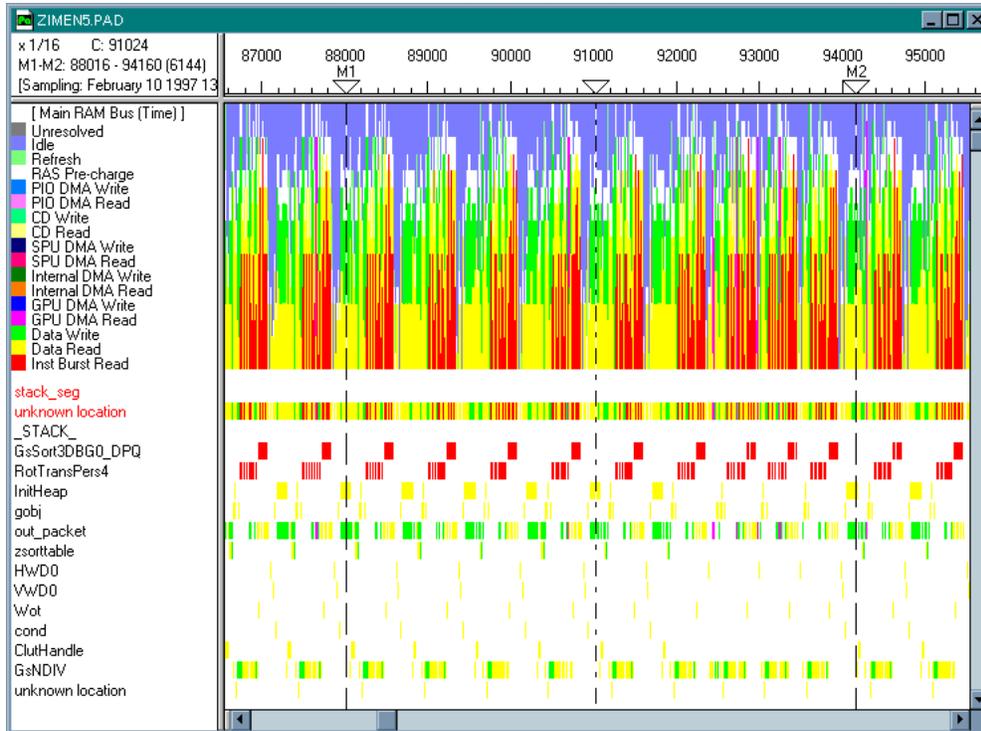
These patterns appear cyclically. From the clock cycle values indicated on the ruler, the period is found to be about 700 cycles. As this period is shorter, and these patterns represent a greater proportion of the overall processing, a greater improvement can be expected despite the application of less tuning. Here, enclose several cyclic patterns, regarded as representing loop processing, with markers M1 and M2. Then, read the mapping information, and display only those global symbols that are accessed in the section by selecting filtering from the menu. (Figure 12 already indicates these global symbols.) Next, enable global symbol access display. Furthermore, with the option menu, reset the scale factor specification for global symbol access display to enable display using the current scale factor.

Then, the global symbols accessed in the M1-M2 section are displayed as shown in Figure 12. Those functions that caused instruction cache misses are displayed in red. We can determine which functions caused a particularly large number of instruction cache misses. Figures 13 and 14 show the data dumped by positioning the cursor to each function and double-clicking.

Figure 13: Function that encountered an instruction cache miss (1)

```
DUMP ZIMEN5.PAD                                                              _ □ ×

  Prev          Next       Go to Marker        Page:  900

  P   MR   N   N   M           S    O    D    MMMM   S    S    V    V  V  V    V    V  V    V    G    RDC
  O   B/   W   B   A           Y    F    A    WWWW   B    A    B    M  M  M    M    M  M    M    U    XST
  S   W    O   Y   D           M    F    T    EEEE             L         M  A  X    Y    A  X    Y    N    DRS
      S    R   T   D           B    S    A    NNNN   S    N    K    0  C  0    0    C  1    1    I    111
      T    D   E   R           O    E         3210   T              D  C              C              N    ***
      A                        L    T         ****   A             E  0              1                    *
      T                                               T                                                  *

  89906:                                     ----      0   R/W                 W(335, 452) 0   111
  89907:                                     ----      0   R/W W(336, 452)                 0   111
  89908:                                     ----      0   R/W                 W(337, 452) 0   111
  89909: DR   1                              ----      0   R/W W(338, 452)                 0   111
  89910: DR                                  ----      0   R/W                 W(339, 452) 0   111
  89911: DR                                  ----      0   R/W W(340, 452)                 0   111
  89912: DR       807FFE5C      GsNDIV+798C2C 8004E2FC ----  0  R/W            W(341, 452) 0   111
  89913: H                                   ----      0   R/W W(342, 452)                 0   111
  89914:                                     ----      0   R/W                 W(343, 452) 0   111
  89915: IR   4                              ----      0   R/W W(344, 452)                 0   111
  89916: IR                                  ----      0   R/W                 W(345, 452) 0   111
  89917: IR                                  ----      0   R/W W(346, 452)                 0   111
  89918: IR       8001A5A0      RotTransPers4+000058 E90E0000 ----  0  R/W     W(347, 452) 0   111
  89919: IR       8001A5A4      RotTransPers4+00005C E9280000 ----  0  R/W W(348, 452)     0   111
  89920: IR       8001A5A8      RotTransPers4+000060 4848F800 ----  0  R/W     W(349, 452) 0   111
  89921: IR       8001A5AC      RotTransPers4+000064 48029800 ----  0  R/W W(350, 452)     0   111
  89922: H                                   ----      0   R/W                 W(351, 452) 0   111
  89923: DR   1                              ----      0   R/W W(352, 452)                 0   111
  89924: DR                                  ----      0   R/W                 W(353, 452) 0   111
  89925: DR                                  ----      0   R/W W(354, 452)                 0   111
  89926: DR       807FFE60      GsNDIV+798C30 807FFF10 ----  0  R/W            W(355, 452) 0   111
  89927: H                                   ----      0   R/W W(356, 452)                 0   111
  89928:                                     ----      0   R/W                 W(357, 452) 0   111
  89929: DR   1                              ----      0   R/W W(358, 452)                 0   111
  89930: DR                                  ----      0   R/W                 W(359, 452) 0   111
```

Instructions on cache line 5A

Figure 14: Function that encountered an instruction cache miss (2)

```
DUMP ZIMEN5.PAD                                                              _ □ ×

  Prev          Next       Go to Marker        Page:  901

  P   MR   N   N   M           S    O    D    MMMM   S    S    V    V  V  V    V    V  V    V    G    RDC
  O   B/   W   B   A           Y    F    A    WWWW   B    A    B    M  M  M    M    M  M    M    U    XST
  S   W    O   Y   D           M    F    T    EEEE             L         M  A  X    Y    A  X    Y    N    DRS
      S    R   T   D           B    S    A    NNNN   S    N    K    0  C  0    0    C  1    1    I    111
      T    D   E   R           O    E         3210   T              D  C              C              N    ***
      A                        L    T         ****   A             E  0              1                    *
      T                                               T                                                  *

  90056:                                     ----      0   R/W W(388, 454)                 0   111
  90057: DR   1                              ----      0   R/W                 W(389, 454) 0   111
  90058: DR                                  ----      0   R/W W(390, 454)                 0   111
  90059: DR                                  ----      0   R/W                 W(391, 454) 0   111
  90060: DR       8004E2FC      out_packet+017A68 001600E7 ----  0  R/W W(392, 454)        0   111
  90061: H                                   ----      0   R/W                 W(393, 454) 0   111
  90062: IR   4                              ----      0   R/W W(394, 454)                 0   111
  90063: IR                                  ----      0   R/W                 W(395, 454) 0   111
* 90064: IR                                  ----      0   R/W W(396, 454)                 0   111
  90065: IR       800125A0      GsSort3DBG0_DPQ+000590 0065102A ----  0  R/W   W(397, 454) 0   111
  90066: IR       800125A4      GsSort3DBG0_DPQ+000594 10400002 ----  0  R/W W(398, 454)   0   111
  90067: IR       800125A8      GsSort3DBG0_DPQ+000598 00000000 ----  0  R/W   W(399, 454) 0   111
  90068: IR       800125AC      GsSort3DBG0_DPQ+00059C 00A01821 ----  0  R/W W(400, 454)   0   111
  90069: H                                   ----      0   R/W                 W(401, 454) 0   111
  90070:                                     ----      0   R/W W(402, 454)                 0   111
  90071:                                     ----      0   R/W                 W(403, 454) 0   111
  90072: IR   4                              ----      0   R/W W(404, 454)                 0   111
  90073: IR                                  ----      0   R/W                              0   111
  90074: IR                                  ----      0   R/W                              0   111
  90075: IR       800125B0      GsSort3DBG0_DPQ+0005A0 9662000A ----  0  R/W                0   111
  90076: IR       800125B4      GsSort3DBG0_DPQ+0005A4 86640012 ----  0                     0   111
  90077: IR       800125B8      GsSort3DBG0_DPQ+0005A8 00025400 ----  0                     0   111
  90078: IR       800125BC      GsSort3DBG0_DPQ+0005AC 000A3C03 ----  0  R/W                0   111
  90079: H                                   ----      0   R/W W(320, 456)                 0   111
  90080: DR   1                              ----      0   R/W                 W(321, 456) 0   111
```

Other instructions on the same cache line 5A

The size of the "PlayStation"'s instruction cache is 4KB. The cache line size is 4 long words, and 256 cache lines are allowed. As shown in Figure 13, the lower two digits next to the lowest digit of a main RAM bus address represent the line number. This means that a cache miss is caused by instructions that are located at different addresses but which have the same lower two digits. There are several methods of eliminating such a cache miss. Use inline expansion or DMPSX, or link those functions to locate the functions at addresses close to each other if the functions are user functions. Note, however, that a cache miss is

unavoidable if the loop includes code of 4KB or more. One speedup technique for the "PlayStation" involves ensuring that no small loop includes a code of 4KB or more.

Next, position M1 and M2 to cover the section including all the patterns for loop processing involving a cache miss, then collect the statistical information for the section (Figure 15).

Figure 15: Statistical information of a portion where an instruction cache miss occurred


```
STATISTICS ZIMEN5.PAD [71040-120192]                                    _ □ ✕
ZIMEN5.PAD [Sampling: February 10 1997 13:01:21]                         ▲
Range: 71040 - 120192                                                    ▓

Main Memory Bus:
                    Time(%) Bytes   Speed(MB/sec)  Clock Cycles/word  Estimated CPU Stall Cycles
        unresolved    0.0   ----        ----            ----               --------
              IDLE   24.7   ----        ----            ----               --------
           REFRESH    1.6   ----        ----            ----               --------
      RAS PRECHARGE  12.3   ----        ----            ----               --------
      PIO DMA WRITE   0.0      0        ----            ----               --------
       PIO DMA READ   0.0      0        ----            ----               --------
       CD DMA WRITE   0.0      0        ----            ----               --------
        CD DMA READ   0.0      0        ----            ----               --------
      SPU DMA WRITE   0.0      0        ----            ----               --------
       SPU DMA READ   0.0      0        ----            ----               --------
  Internal DMA WRITE  0.0      0        ----            ----               --------
   Internal DMA READ  0.0      0        ----            ----               --------
      GPU DMA WRITE   0.0      0        ----            ----               --------
       GPU DMA READ   2.4   2884       83.31            1.6               --------
         DATA WRITE  14.4   5148       24.75            3.8               --------
          DATA READ  27.3  13440       34.00            4.0                 13440
       I-BURST READ  17.2  19120       76.71            1.8                  4842
                                                                         ▼
```

The CPU stall time caused by an instruction cache miss corresponds to about half of the red indication provided by main RAM bus analysis. Here, the CPU stall time is represented by the number of clock cycles. If cache misses can be eliminated from this section, processing can be speeded up by the corresponding number of clock cycles.

Detection of Duplicate Data Reads

A CPU data read cycle on the main RAM bus is indicated as a yellow pattern by main RAM bus analysis. The CPU stall time corresponds to the total yellow area. Data such as global symbols may be read from main RAM, but a work area and temporary variables should be accessed using, for example, the scratch pad. If main RAM must still be accessed, tuning can be achieved by checking whether there is a duplicate read, that is, by checking whether there is an access for reading from the same address more than once without writing. In Figure 11, such a duplicate read is indicated by the read/write penalty "duplicate read." A red area represents a duplicate read, or CPU stall time. Figure 16 shows an enlarged view of a portion that includes many read patterns. Here, position the cursor to a red pattern, then double-click to dump the data (Figure 17).

Figure 16: Duplicate read of data from main RAM (read/write penalty)
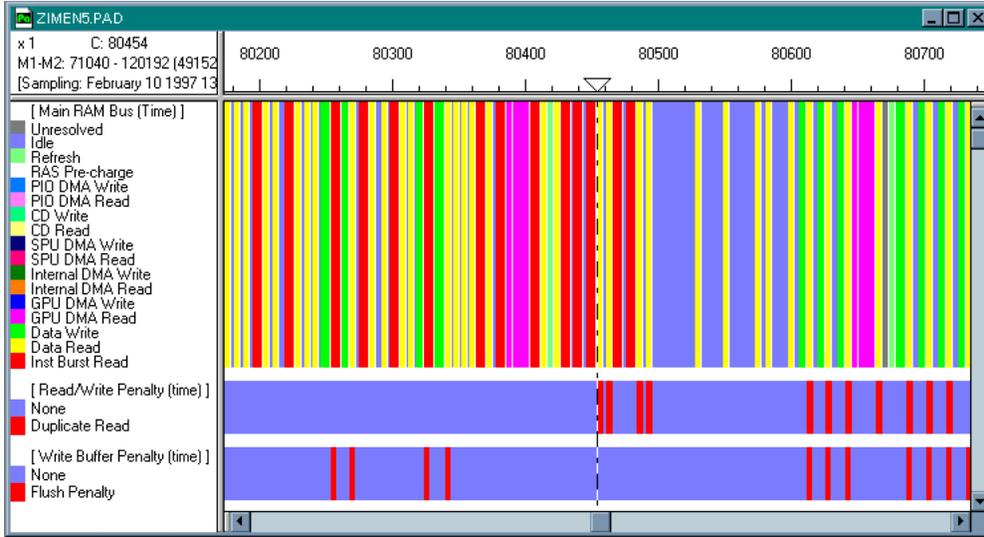
Figure 17: Duplicate read of data from main RAM (data dump 1)
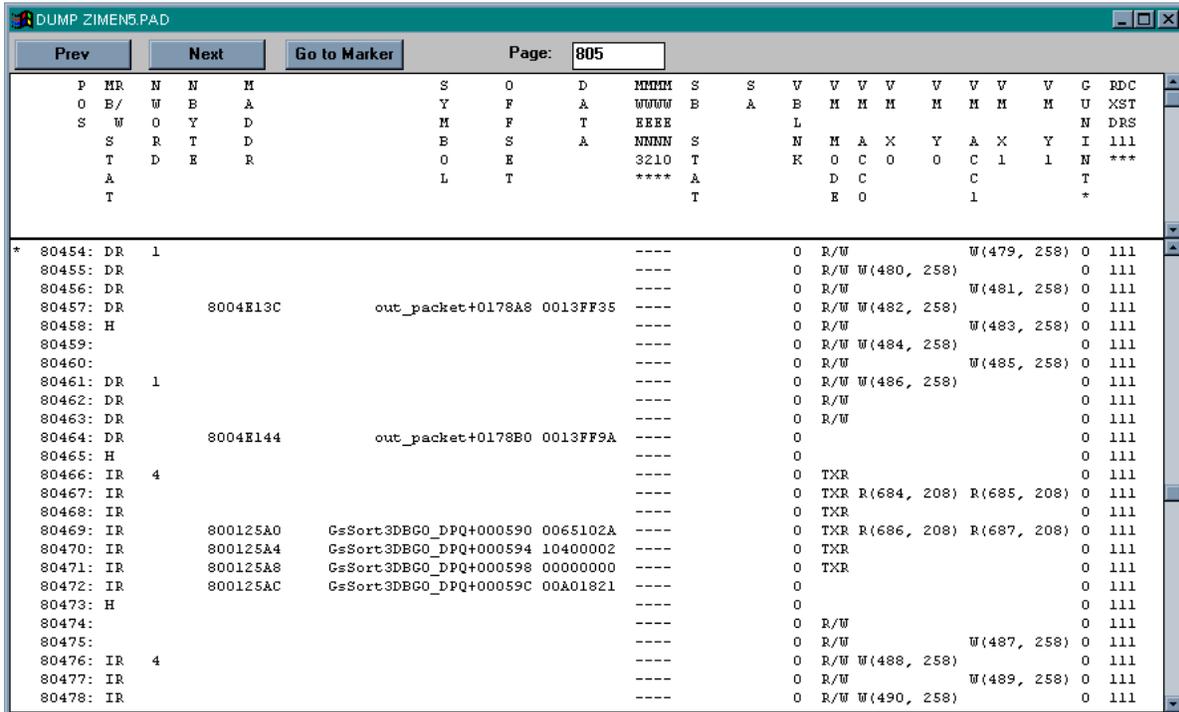
Next, by using the search command, identify an address that is accessed more than once. Figure 18 shows the dumped data.

Figure 18: Duplicate data read from main RAM (data dump 2)
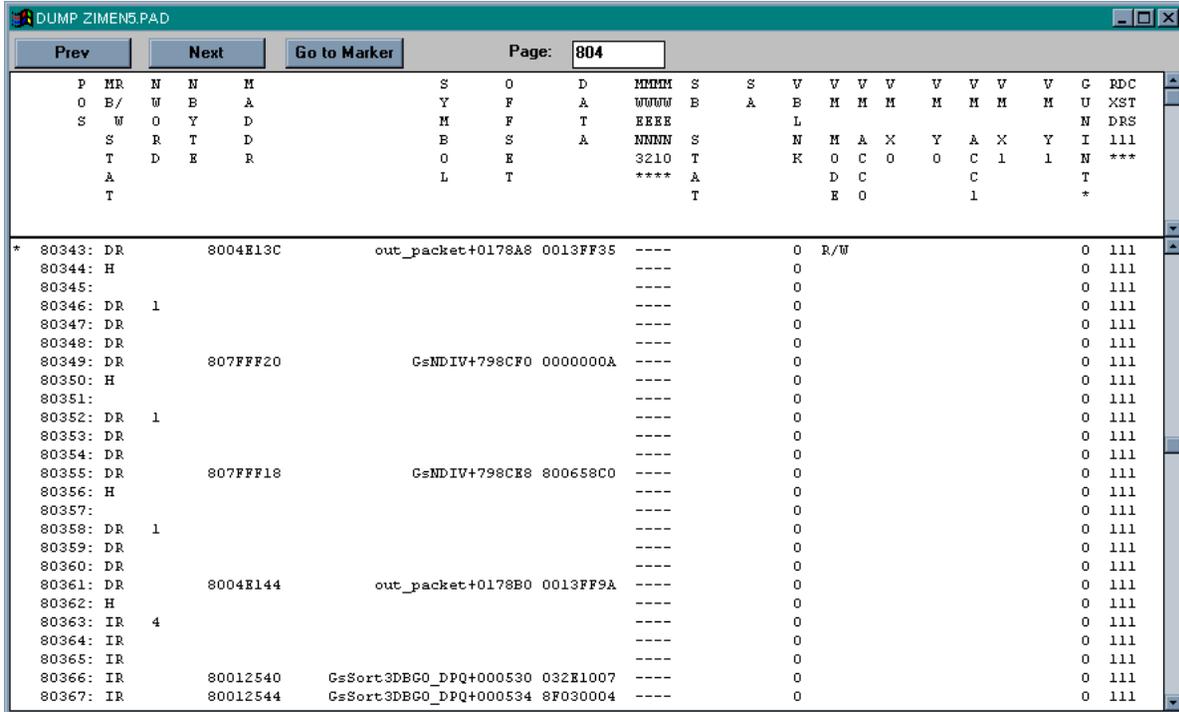
The reason for such a duplicate read is described in Flow of Diagnosis. The code should be modified, depending on the cause, to eliminate duplicate reads whenever possible.

A byte access or short-word access is handled as a long-word access on the main RAM bus. This means that, even when a read access which is not duplicated is coded, the performance analyzer may indicate the access as a duplicate read.

The CPU stall time for one read access is four cycles. So, if a code for using the scratch pad or a register to avoid access to main RAM is shorter than the number of duplicate accesses multiplied by four instructions, the use of such code will speed up processing.

A duplicate global symbol read can be identified from global symbol access and data dumping. However, access to a stack area requires some care. Namely, when a symbol file is read without first setting a stack area using the performance analyzer, an access to a stack area is mistakenly indicated as a highest-level access to a global symbol. In such a case, an access to a stack causes an extremely high offset value for a particular global variable.

Detection of a Write Buffer Flush Penalty

In Figure 11, a write buffer analysis indicates a portion where a device such as the CPU may stall because an access request is generated while the write buffer is being flushed. The area represents the stall time. For tuning, the portion where many red patterns are visible should be carefully checked.

Figure 19: Enlargement of a portion including a write buffer flush penalty
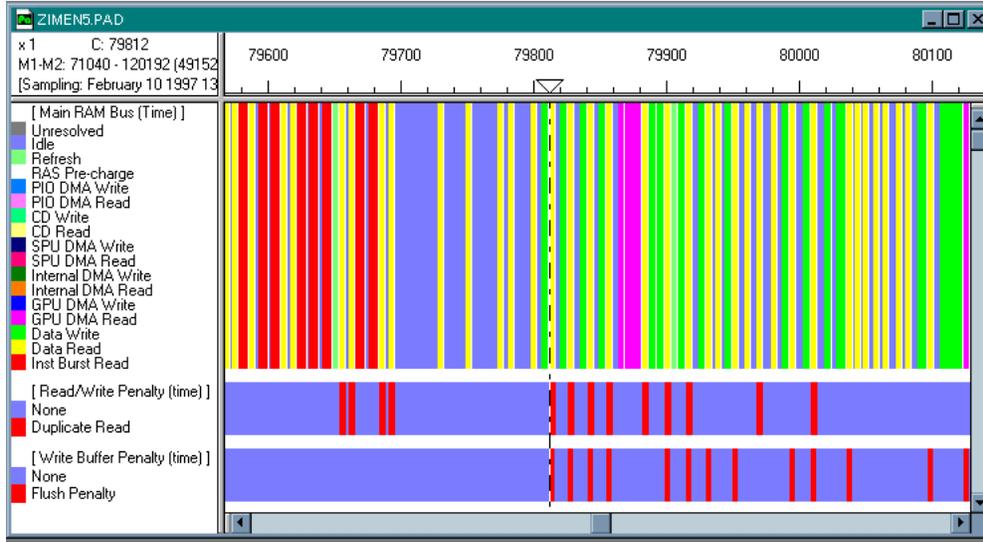
Figure 19 shows an enlarged view of the portion; at the cursor position, a CPU read access occurred immediately after a write access. Figure 20 shows the data dump, displayed by double-clicking.

Figure 20: Write buffer flush penalty (data dump)
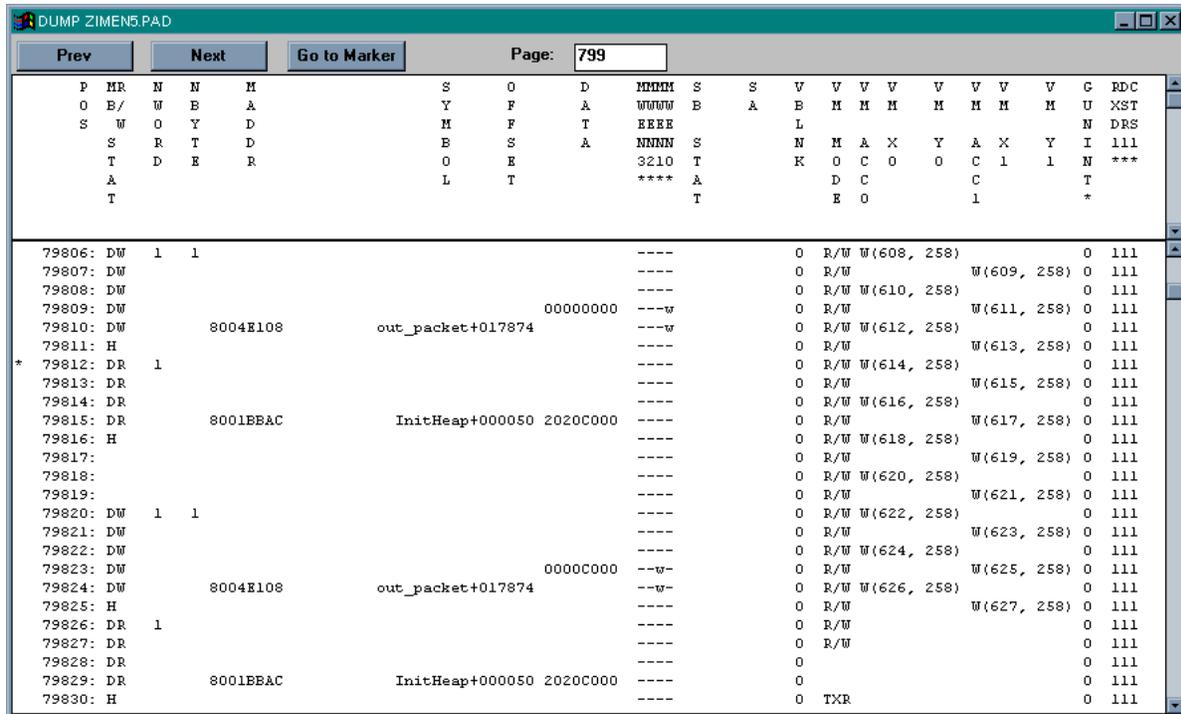
Check the code. If a store instruction is immediately followed by a load instruction, reduce the stall time by inserting another instruction between the two instructions or by exchanging the two instructions with each other, if possible. Particularly, when a read and write occur alternately and repeatedly, a longer stall time results. In such a case, take advantage of the four stages of the write buffer. That is, the stall time can be dramatically reduced by modifying the code so that four writes occur in succession. When 100,000 polygons are to be displayed per second, for example, the length of a loop for processing one polygon will be about 200 to 300 cycles. If a stall time of 10 cycles is reduced by tuning the write buffer, an improvement of 3% to 5% is achieved. This means that the number of polygons to be displayed can be

increased by such an improvement. Note that, even if an instruction other than a store instruction is executed immediately after flushing, a write buffer access causes a red pattern when a read access occurs immediately after on the main RAM bus. Patterns do not always represent penalties.

Detection of Null Packets

As shown by the GPU packet analysis in Figure 1, a null packet is represented by a white pattern. White patterns occurring in succession represent successive null packets in the ordering table. Successive null packets can be eliminated using multiple ordering tables, for example.
An enlarged view of a null packet indicates that the drawing is delayed accordingly, as shown in Figure 21.

Figure 21: Null packet detection
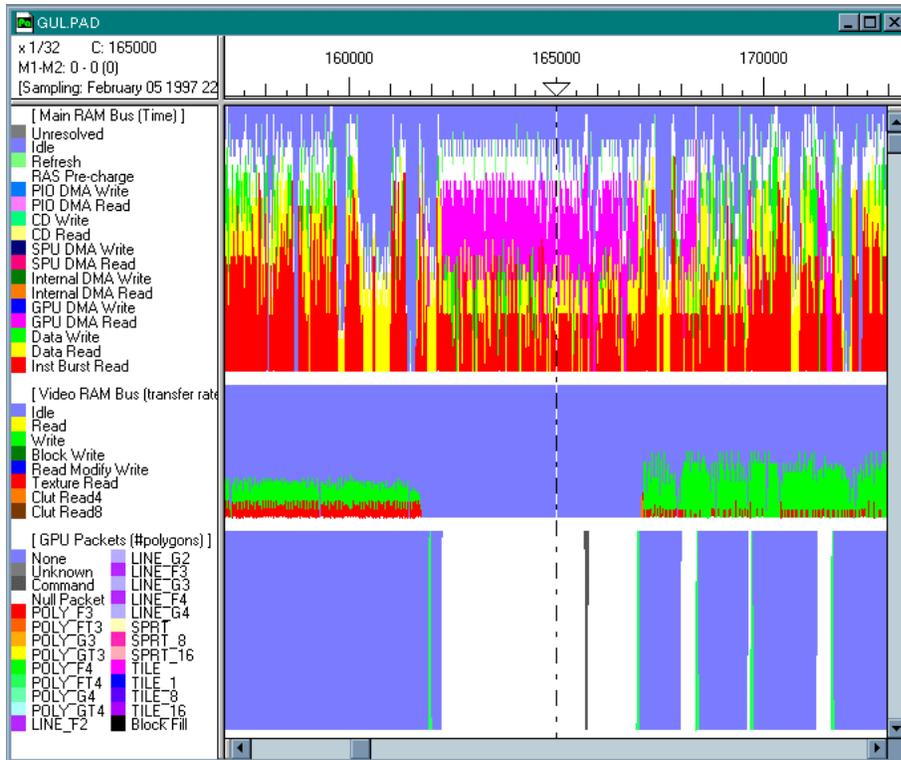
If null packets are mixed with other polygon packets, the drawing efficiency may have decreased because the resolution of the ordering table is too high. If video RAM bus analysis indicates that the green pattern is low, the problem may be solved by lowering the resolution.

Detection of Inefficient Texture Cell Reads

Figure 22: Drawing efficiency check
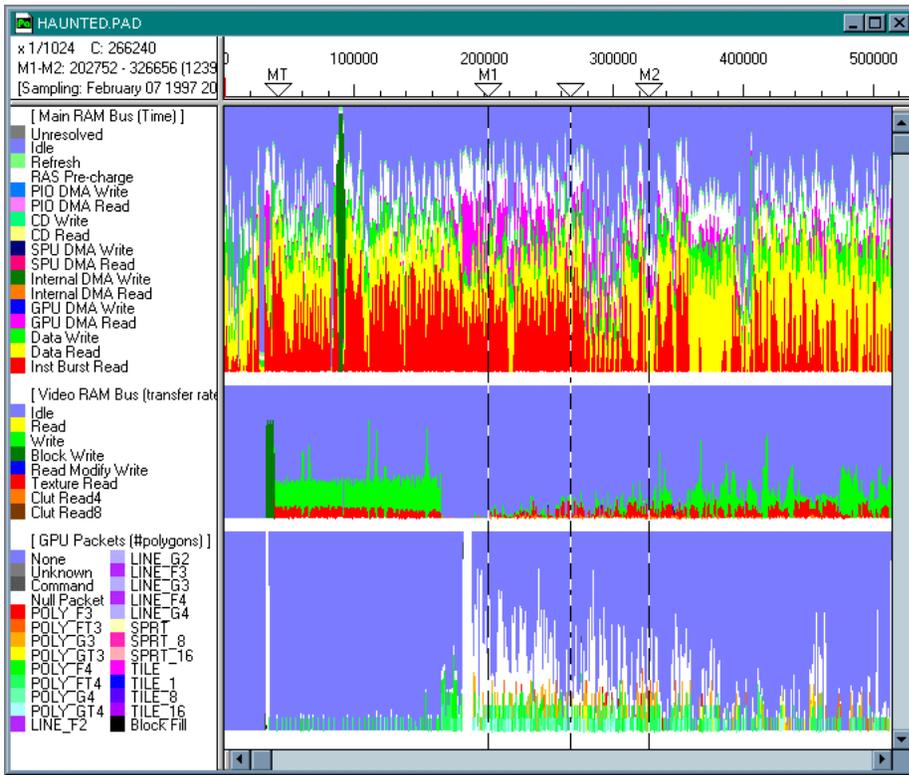
As shown in Figure 22, position markers M1 and M2 to a portion with fewer green patterns on the video RAM bus, and enlarge the enclosed portion. In this case, the pattern shown in Figure 23 is obtained.

Figure 23: Portion containing more texture cache misses
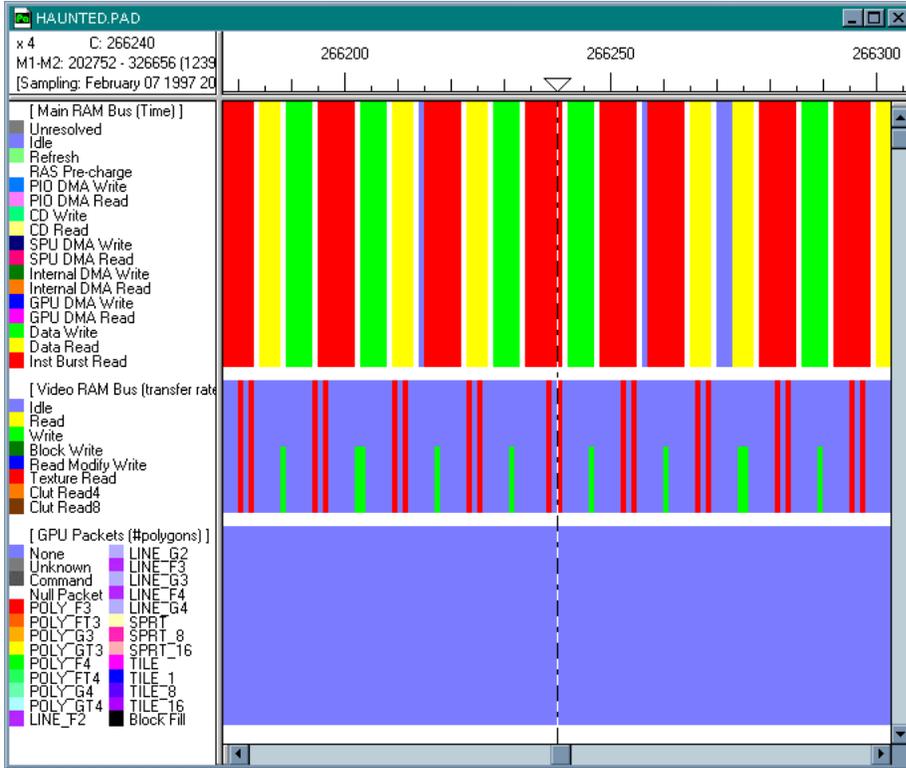
Next, position markers M1 and M2 to that portion having more green patterns, and enlarge the portion. In this case, the pattern shown in Figure 24 is obtained.

Figure 24: Portion containing fewer texture cache misses
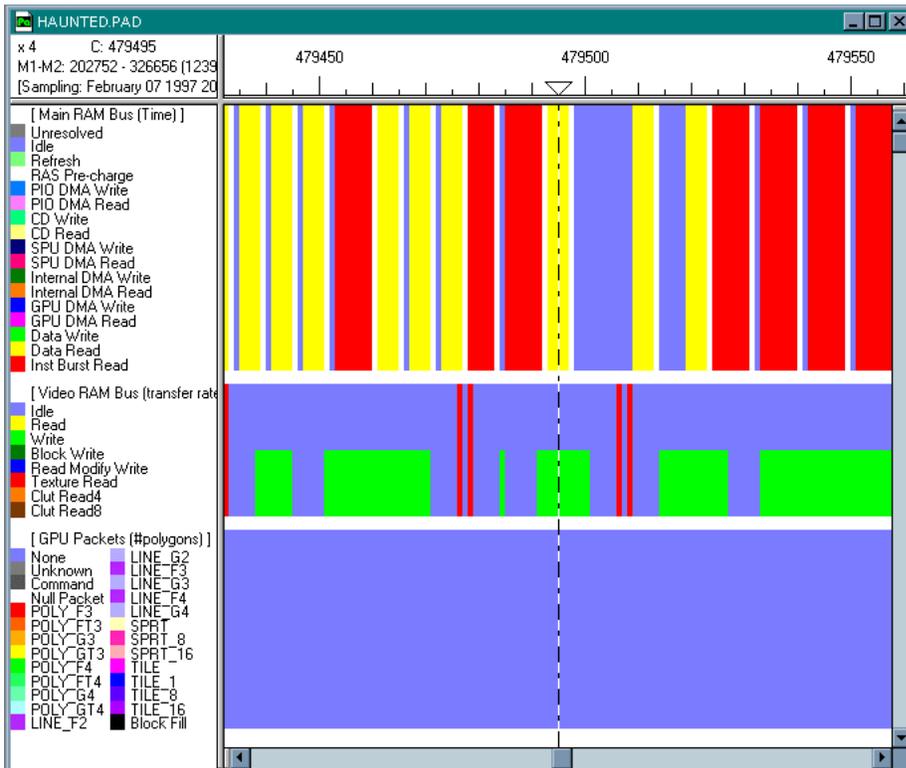
The two red stripes produced by video RAM bus analysis represent a 64-bit texture read. For texture read, the texture cells corresponding to the line size of the texture cache are read from video RAM if a texture cache miss occurs; the texture cells read at one time are always 64 bits long.

Figure 25 shows an enlarged view of a portion for which the drawing efficiency is high. When such a portion is checked with the video RAM viewer, a pattern like that shown in Figure 26 is obtained. On the other hand, when a portion for which the drawing efficiency is poor is enlarged, as shown in Figure 27, and is checked with the video RAM viewer, a pattern like that as shown in Figure 28 is obtained.

Figure 25: Portion of a high drawing efficiency (video RAM bus analysis)
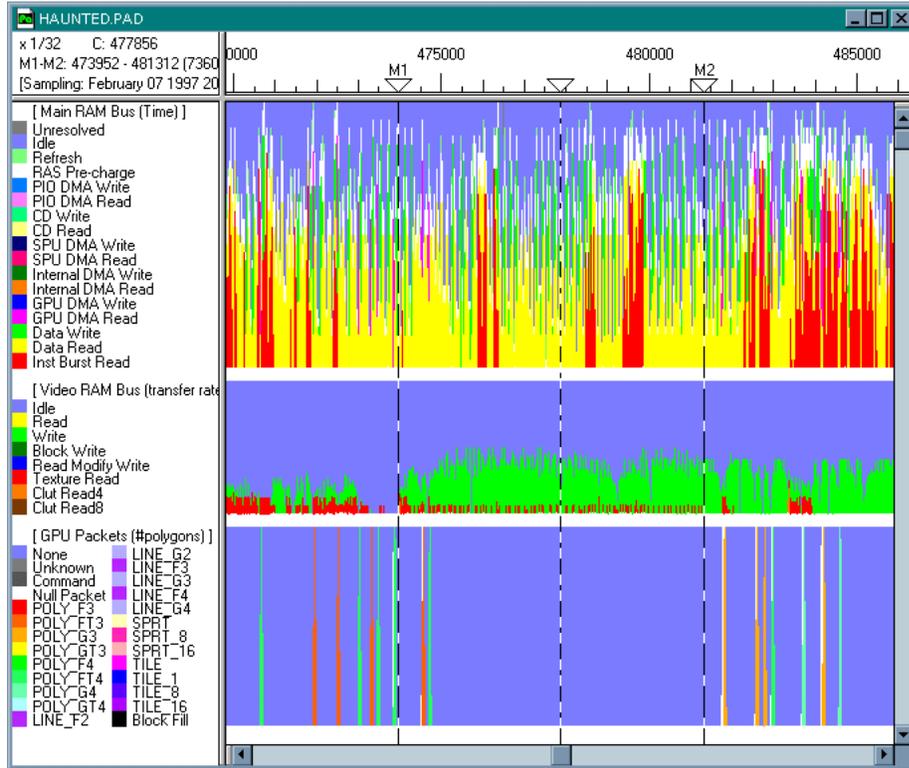
Figure 26: Portion having a high drawing efficiency (video RAM viewer)
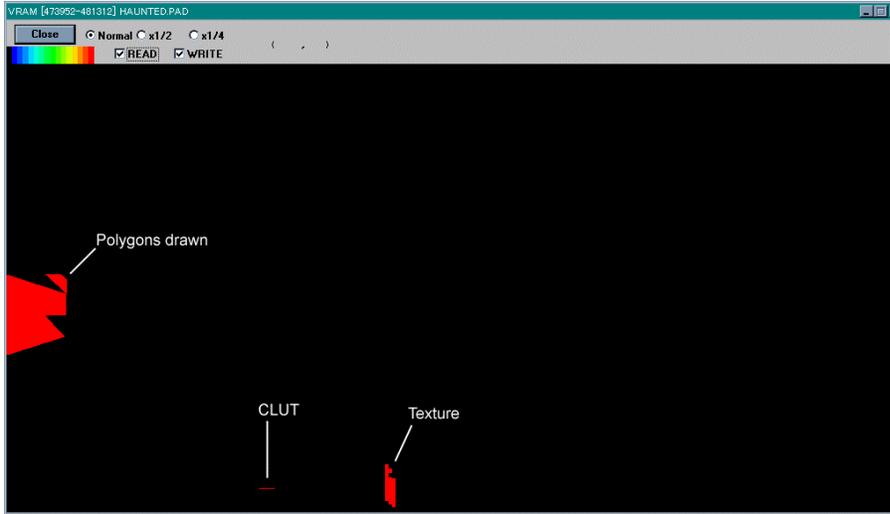
Figure 27: Portion having a low drawing efficiency (video RAM bus analysis)
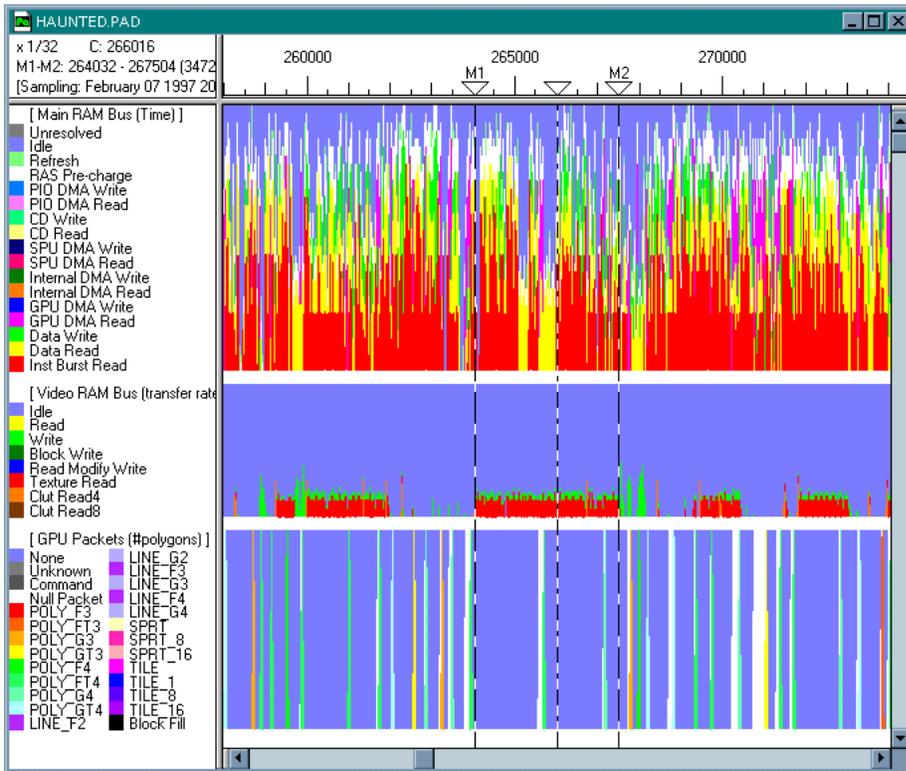
Figure 28: Portion having a low drawing efficiency (video RAM viewer)
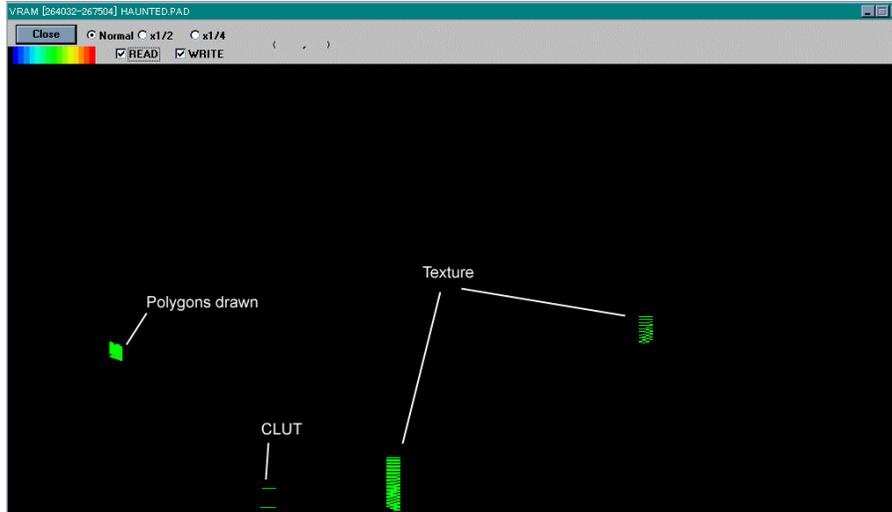
In a texture cache line fill, horizontally successive texture cells of the texture area are read. So, if the resolution of the texture is too high, and 4-bit texture mode is used, for example, 15 out of 16 texture cells may be discarded, and only one pixel may be drawn. An example is shown in Figure 23. In this example, only one pixel is drawn for each texture read caused by a texture cache miss. When Figure 26 is compared with Figure 28, it can be seen that more texture reads are performed in Figure 28.

A similar phenomenon occurs when a texture larger than the texture cache is used, and when a polygon to be drawn is rotated through 90 degrees relative to the texture pattern. Each cause can be identified from read access and write access patterns with the video RAM viewer. Apply appropriate action such as changing the texture size and texture resolution, depending on the identified cause.

Figures 29 and 30 show the improvement realized for the sample program, *psx\sample\graphics\mipmap\tuto5.cpe*, made by mip-mapping. These figures reveal that a significant improvement in drawing speed can be achieved by applying mip-mapping.

Figure 29: Sample program (without mip-mapping)
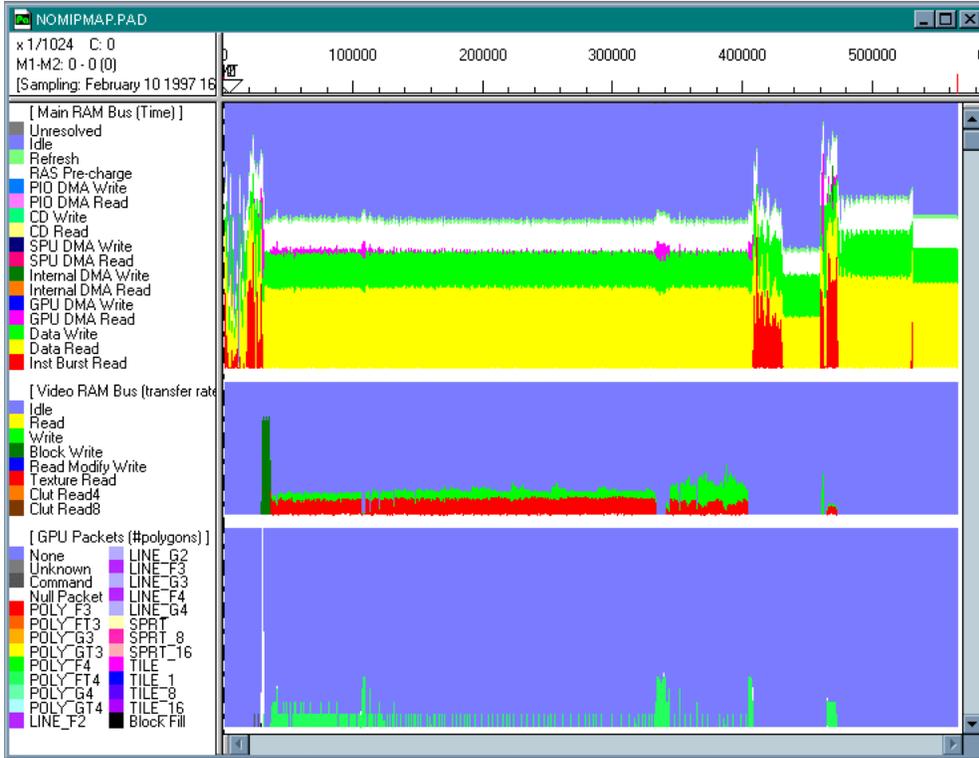

Figure 30: Sample program (with mip-mapping)
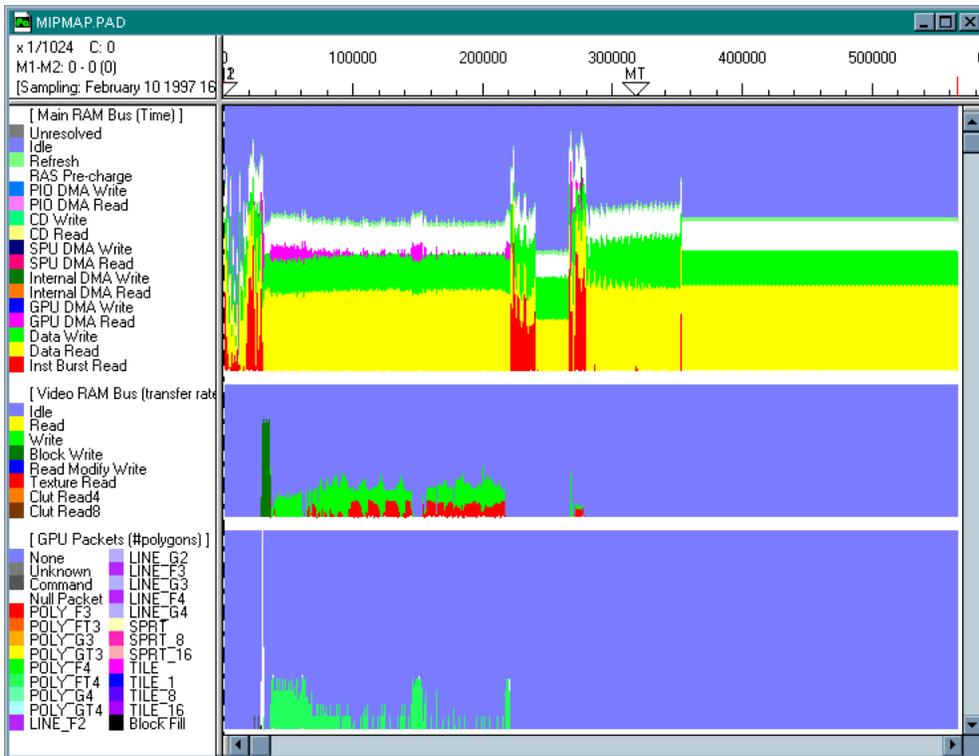
Detection of Transparent Colors

Figure 31: A polygon including transparent colors
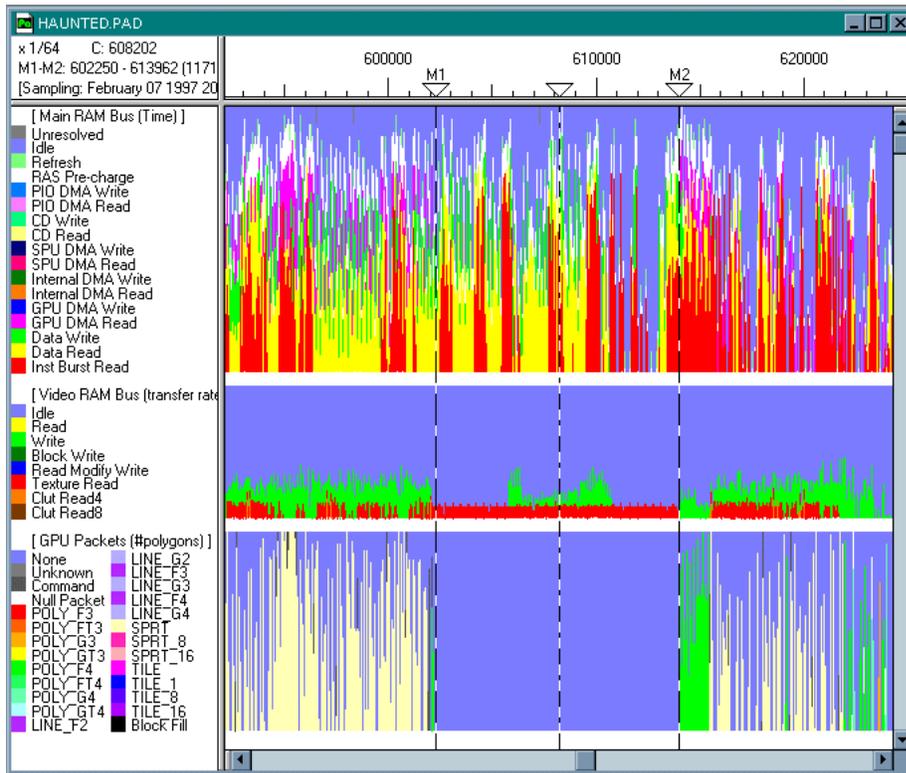
Based on GPU packet analysis, Figure 31 shows an enlarged view of a section containing no accesses. From the GPU packet analysis, the drawing of one large polygon is assumed. Video RAM bus analysis indicates that the texture is read constantly, but that drawing is not performed in many portions. Figure 32 shows the information obtained with the video RAM viewer by positioning M1 and M2 such that they enclose the drawing section of a polygon.

Figure 32 : A polygon including transparent colors (video RAM viewer)
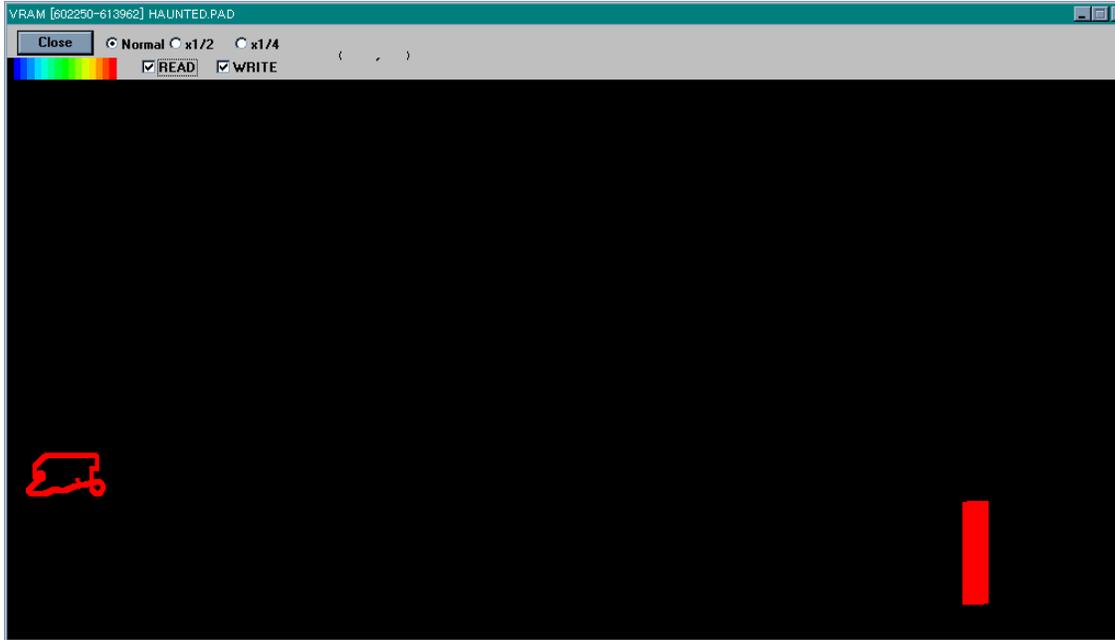
Usually, with the video RAM viewer, the left side represents the frame buffer, while the right side represents the texture area. When the double-buffer method is used, the frame area is divided into an upper area and lower area; each time the frame is switched, the access area is switched. Figure 32 shows that a transparent color is used in the frame buffer area. Thus, a polygon causing a problem can be identified using the video RAM viewer. A polygon with large transparent area should be divided to reduce the size of transparent area.

CLUT Switching

Figure 33: CLUT switching and polygons requiring considerable preprocessing
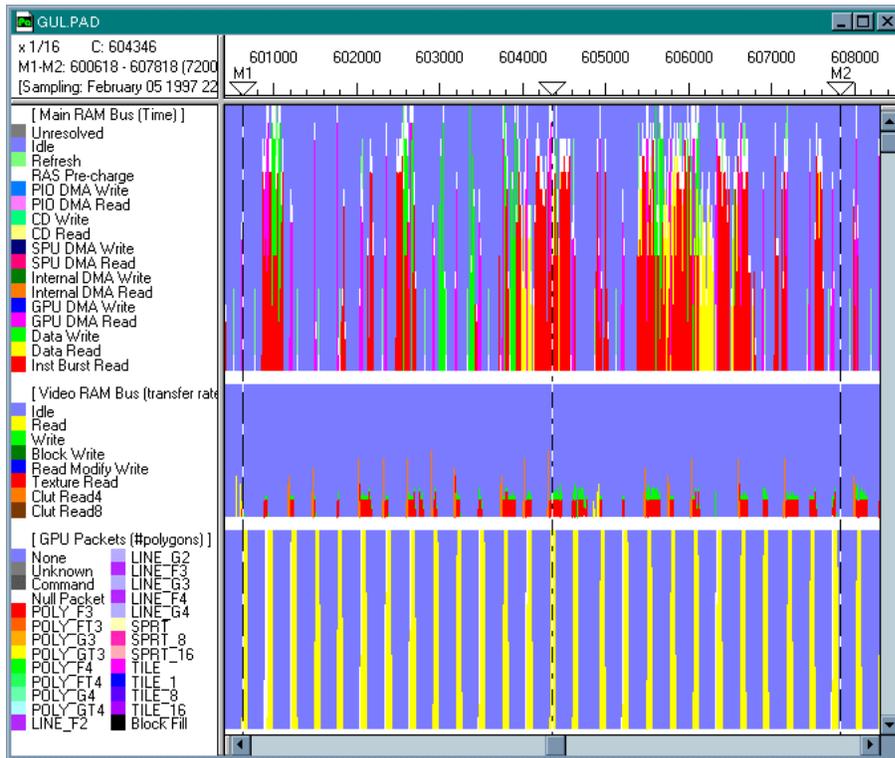
Another example having poor drawing efficiency is shown in Figure 33. A video RAM bus analysis shows that an orange pattern representing a 4-bit CLUT read occurs with each polygon. This means that many polygons use different CLUTs with close Z values. If the video RAM bus is occupied because of frequent switching between CLUTs, action is required.

Preprocessing Bottleneck

The GPU packet analysis shown in Figure 33 indicates that a Gouraud texture polygon is drawn. A video RAM bus analysis indicates that a relatively large portion involving no accesses precedes texture read or pixel drawing. This means that a long time is required for Gouraud texture preprocessing. To speed up the processing, those small polygons that require considerable preprocessing should not be drawn wherever possible.

Polygon Penalties

Figure 34 :Polygon penalties
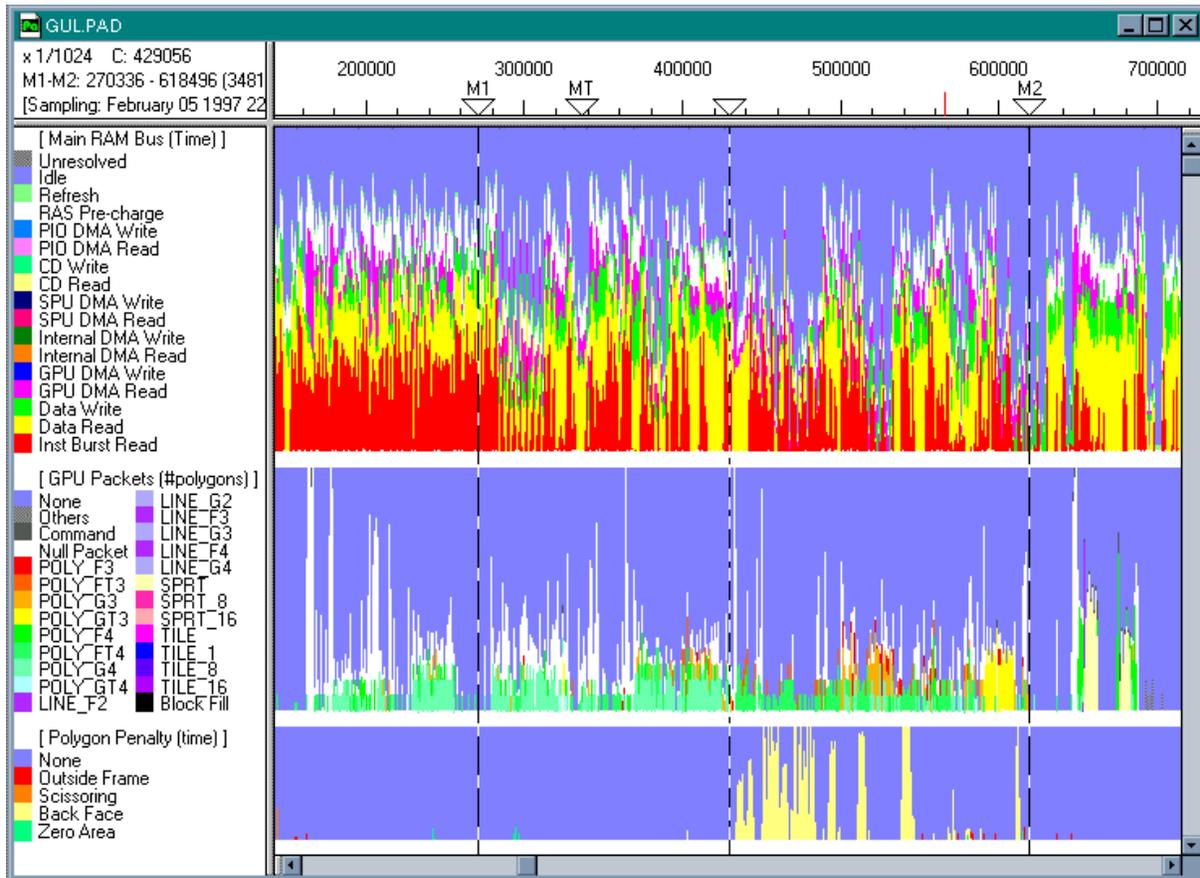
Figure 34 evaluates and indicates the penalties of zero-area polygons, polygons with a poor scissoring efficiency, polygons clipped by the GPU, and back-face polygons subject to normal clipping. To improve the processing, cause the CPU to perform normal clipping, area checking, polygon division, and area clipping for these polygons, if the CPU has enough processing time left to do them.

The performance analyzer identifies back-face polygons checking the order of vertices in a polygon packet. Some programs may assume such an order is for front-face polygons, or may assume any kind of order is for valid polygons which should be drawn. In such cases set the parameter in *options* dialog box to display correct polygon penalties.

Also the offset value and the screen size should be set in *options* dialog box to display correct polygon penalties.