

*The DTL-H2700  
Development System  
and  
Performance Analyzer*



# Overview

- ▶ Introducing the DTL-H2700
  - The Hardware
  - The DTL-H2700 as Development System
  - The DTL-H2700 as Performance Analyzer
- ▶ Why Analyze?
- ▶ Methods for Optimizing PlayStation Programs
  - Optimizing the CPU Process
  - Optimizing the GPU Process
- ▶ Using the Performance Analyzer to Analyze PlayStation Programs

# *Introducing the DTL-H2700*

- ▶ Introducing The DTL-H2700
  - The Hardware
  - The DTL-H2700 as Development System
  - The DTL-H2700 as Performance Analyzer

# *Introducing the DTL-H2700*

## ▶ The Hardware

- 3 Card Design
- Plugs into a single ISA slot on an Intel-based PC
  - Requires space for 3 full-length cards
  - Does not use PCI bus
    - Most computer systems do not have room for 3 full length PCI bus cards.
    - Even if your system DID have room, would you want to use up 3 PCI bus slots?
- Takes power connector like a disk drive.

# *Introducing the DTL-H2700*

## ▶ The Hardware

- Uses DTL-H2510 CDROM mechanism
  - Unlike the DTL-H2010 “black box” external CDROM drive used with the DTL-H2000 development system, the DTL-H2510 uses the same mechanism as contained in PlayStation.

# *Introducing the DTL-H2700*

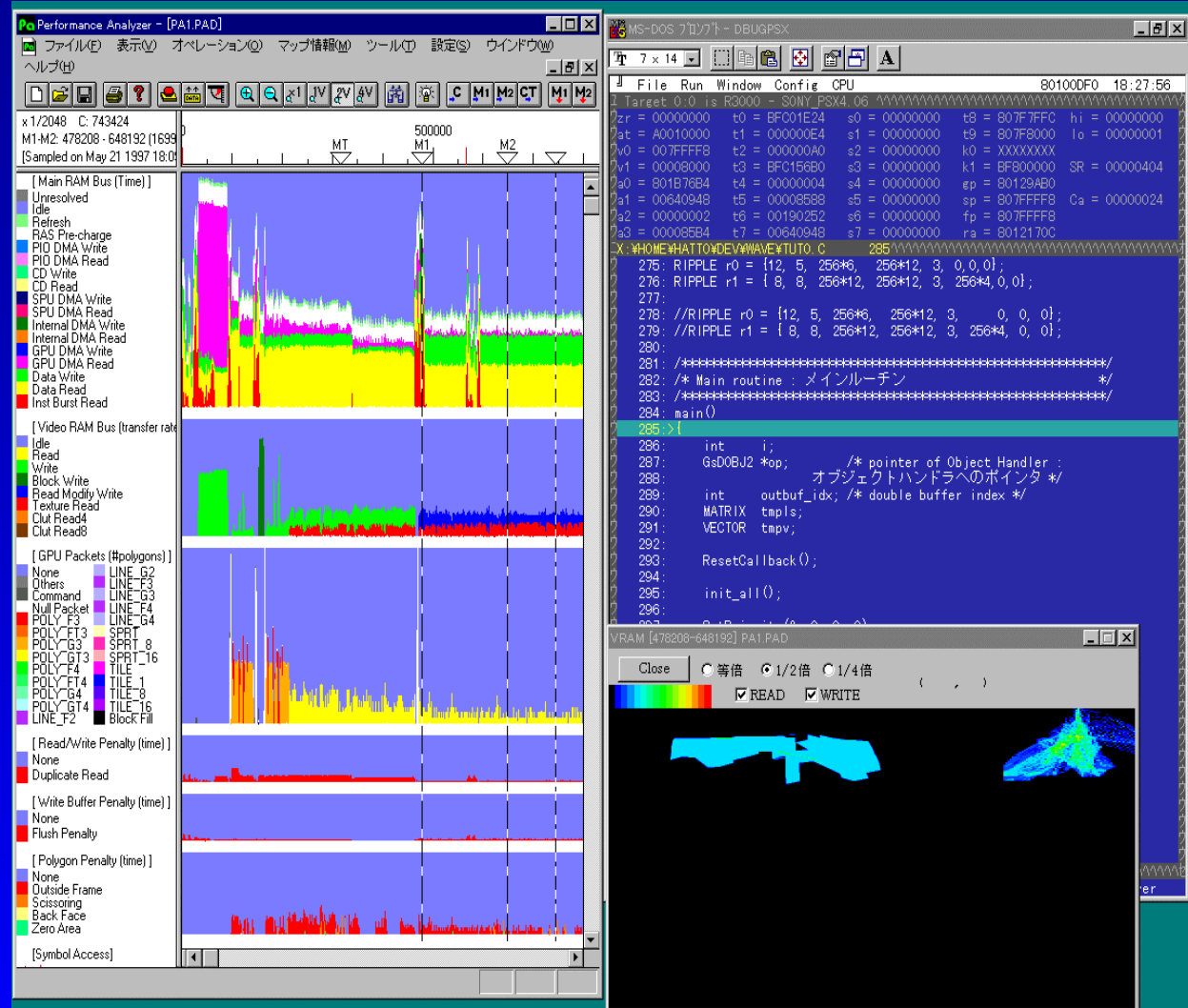
- ▶ DTL-H2700 Development System
  - 8 Megabytes DRAM
  - Compatible with DTL-H2000 drivers & tools
    - DEXBIOS, etc.
  - Can be used with CD Emulator
  - Uses Flash ROM BIOS
    - Does not require SNPATCH.CPE
  - No special programming requirements

# *Introducing the DTL-H2700*

- ▶ DTL-H2700 Performance Analyzer
  - No special software drivers required
    - Standard DTL-H2000 drivers may be installed without conflict
  - Used in conjunction with, but separately from, development system.
    - Other PlayStation programming tools may be used concurrently (DBUGPSX, etc.)

# Introducing the DTL-H2700

## ▶ Performance Analyzer Software & Psy-Q Debugger





# *Introducing the DTL-H2700*

- ▶ DTL-H2700 Performance Analyzer
  - Monitors whatever software process is executing on the rest of the DTL-2700 hardware.
    - Downloaded CPE files
    - EXE files from Bootable CDs running on DTL-H2510
    - EXE files from bootable image running on CD Emulator
  - When trigger condition detected, captures signals on the PlayStation bus for a predetermined length of time
    - 64mb of capture RAM
    - Up to roughly 7.4/60ths of a second
    - Captures on manual trigger switch, GUN interrupt, specific RAM location accesses, or link cable signals

# *Introducing the DTL-H2700*

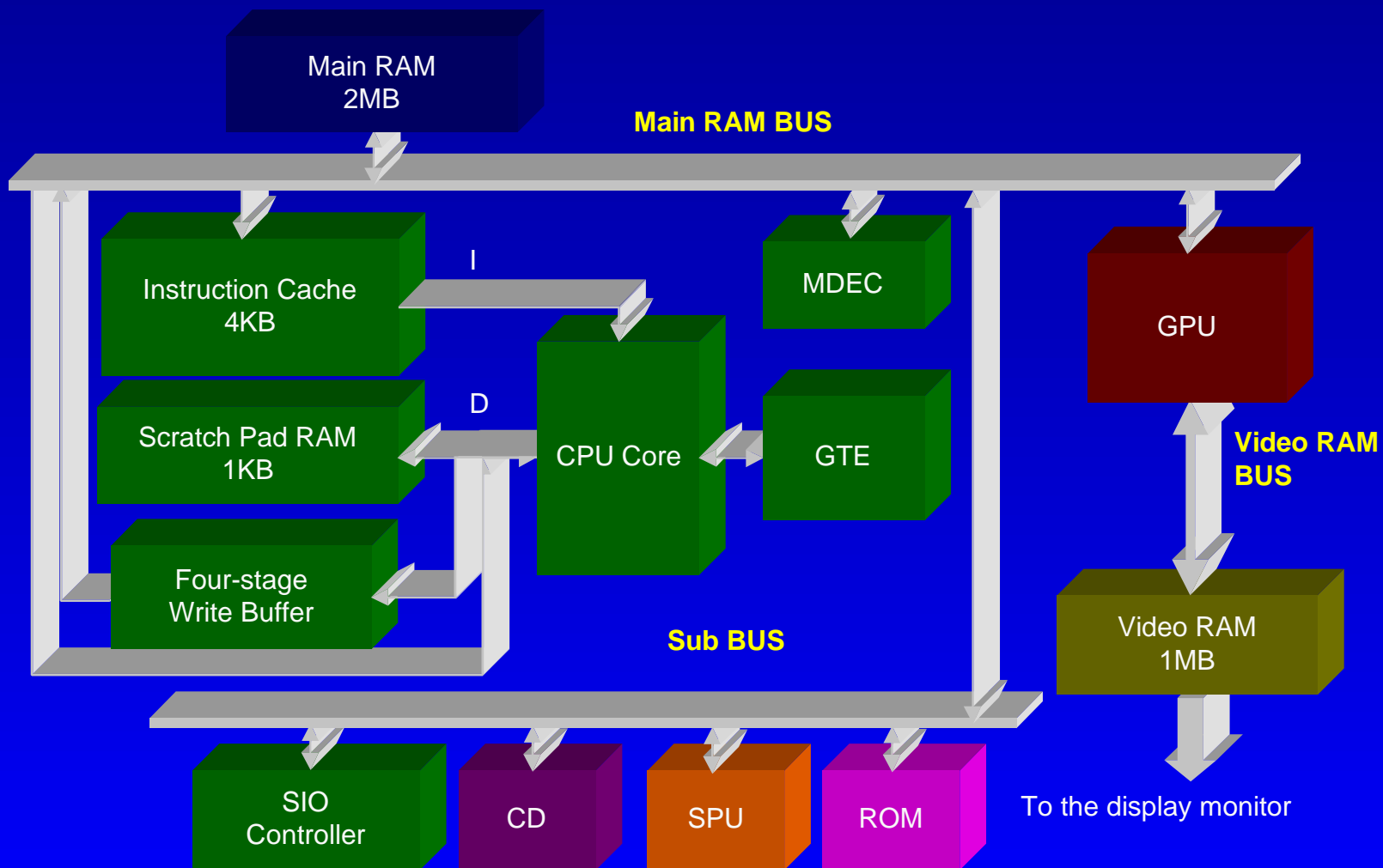
- ▶ DTL-H2700 Performance Analyzer
  - Performance Analyzer application for Windows
    - 16-bit Windows application runs under Windows 3.1 or Windows 95
      - Does not work with Windows NT
    - If no analysis is required, the Performance Analyzer software is not required.
    - Displays graphical analysis of captured bus signals.
      - Shows how much time is spent loading I-Cache, doing DMA operations, performing data reads & writes
      - AND MUCH MORE!

# *Methods for Optimizing PlayStation Programs*

# *Why Analyze?*

- ▶ Determine why a program runs slowly.
- ▶ Estimate how much performance improvement can be achieved.
- ▶ Determine available margin for program additions.
- ▶ Evaluate algorithms & graphics designs at early stages
- ▶ Monitor hangups

# PlayStation System Architecture



# *Optimizing the CPU Process*

- ▶ Reduce I-Cache misses
- ▶ Reduce Read Accesses to main RAM
- ▶ Reduce Write Buffer Flush Penalties

# *Optimizing the CPU Process*

- ▶ Reduce I-Cache misses
  - Move functions which are executed together into the same source code module
  - Use inline expansion of small functions or DMPSX
  - Change the address of the function causing the cache miss
    - This is usually done at the final stages of development

# Optimizing the CPU Process

## ▶ Reduce I-Cache misses

- Check **switch** blocks and structure of loops
  - Inner loop size should be less than 4kb
  - When necessary and possible, use multiple consecutive loops instead of a single loop.

```
/* This may be slow if */
/* functions blow each */
/* other out of I-cache */

for( I = 0; I < 10; I++ )
{
    function_a();
    function_b();
    function_c();
}
```

```
/* This may be faster */

for( I = 0; I < 10; I++ )
    function_a();

for( I = 0; I < 10; I++ )
    function_b();

for( I = 0; I < 10; I++ )
    function_c();
```



# *Optimizing the CPU Process*

- ▶ Reduce Read Accesses to main RAM
  - CPU stalls for 5 cycles on read access of RAM
  - Use the Scratchpad RAM area
    - Temporary work area
    - Permanent or temporary stack area
  - Eliminate unused arguments to LIBGTE functions by using DMPSX instead.
  - Avoid multiple byte or short-word accesses to the same long word address
    - 8-bit, 16-bit, and 32-bit accesses all stall for 5 cycles.
    - Load the 32-bit long word and extract bytes & shorts from it.

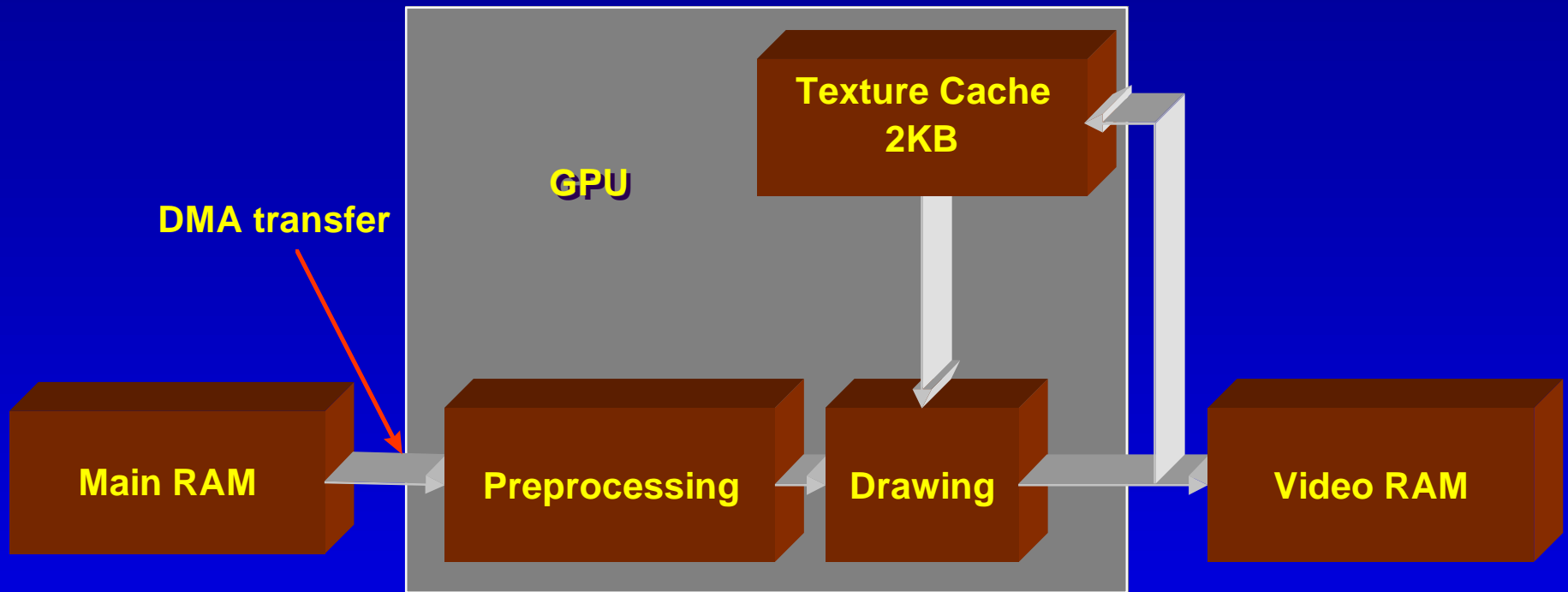
# *Optimizing the CPU Process*

- ▶ Reduce Write Buffer Flush Penalties
  - Avoid alternating accesses such as READ, WRITE, READ, WRITE, ...
  - Change the order of instructions so that multiple WRITE operations are executed together without intervening READ operations.
  - Put instructions that do not access memory after instructions which WRITE to memory.

# *Optimizing the GPU Process*

- ▶ Eliminate intervals without drawing
- ▶ Reduce texture reads
- ▶ Check CLUT switching
- ▶ Reduce GPU preprocessing time
- ▶ Eliminate unnecessary drawing

# GPU Drawing Flowchart



**Pipeline processing time =  $\text{Max}(t(\text{preprocessing}), t(\text{drawing}) + t(\text{texture cache fill}))$**

**Preprocessing time depends only on the polygon type. Drawing time is proportional to drawing area. Texture cache fill time is proportional to texture cache miss ratio**

**In a real case, page-breaks of the video RAM and CLUT read penalties are added**

# *Eliminate Intervals Without Drawing*

- ▶ Eliminate NULL packets in ordering table
  - Check to see where background is registered
    - Are some OT entries always unused?
  - Use multiple ordering tables linked together
    - Use low-resolution main table and link in higher resolution sub-tables
    - Use non-linear ordering tables
      - For example, an ordering table with 100 entries might be set up so that:
        - Last 10 entries represent 50% of depth values farthest from viewpoint
        - First 90 entries represent 50% of depth values closest to viewpoint

# *Eliminate Intervals Without Drawing*

- ▶ Check to see if start of drawing is delayed.
  - Is your program calling **DrawOTag()** as early as possible?
    - If objects are somewhat pre-sorted, then:
      - Process farthest ones first, then call **DrawOTag()** for that batch.
      - Process remaining objects and call **DrawOTag()** again.
    - Use **DrawSyncCallback()** and **VSyncCallback()** to synchronize drawing and buffer switching.
      - Remember that some portions of your game, like AI, don't have to be synchronized to VBLANK.
    - Use triple-buffering scheme if have available memory.
      - 1: Frame being displayed
      - 2: Frame drawn & ready to display at next VBLANK
      - 3: Frame being drawn

# *Eliminate Intervals Without Drawing*

- ▶ Modify graphics design to reduce completely transparent texture areas.
  - Transparent pixels take just as long to process.
  - Watch out for textures with holes in them.
    - Don't draw the hole!
    - Redo polygons in 3D model to go around holes.

# *Reduce Texture Reads*

- ▶ Use 4-bit textures instead of 8-bit or 16-bit
  - Effective cache size will be increased
  - More texture pixels will be loaded in one burst-read
    - Fastest access speed is obtained when the texture pixel data can be read sequentially in the same order as the pixels being drawn, even if the texture data doesn't all fit in a single texture window.
- ▶ Allocate texture positions on cache size boundaries



# *Reduce Texture Reads*

- ▶ Use smaller textures for small polygons
  - Technique known as MIP-MAPPING
- ▶ Avoid rotating textures if texture does not fit in cache.

# Check CLUT Switching

- ▶ What is CLUT switching?
  - When a texture uses a different Color Look Up Table from the previous texture.
- ▶ Controlling CLUT switching is not easy in a Z-sort architecture
- ▶ Using 4-bit textures can require more CLUTs
  - Using 8-bit textures to get 256 colors can sometimes give a better result because CLUT switching is not required.
    - A burst-read access is no longer efficient for scaled-down textures even if 4-bit mode is used.

# *Reduce GPU Preprocessing Time*

- ▶ Small polygons with scaled-down textures can cause a preprocessing bottleneck.
  - Texture data cannot be read efficiently
- ▶ Different Polygon types have different preprocessing requirements

**Gouraud-shaded textured**

**Gouraud Shading non-textured**

**Flat-shaded non-textured**

**Flat-shaded textured**

Preprocessing Time



# *Eliminate Unnecessary Drawing*

## ▶ Back-face polygons

- GPU preprocessing takes longer to draw from right to left instead of left to right.
- Texture reading must be done pixel by pixel.
- Use two polygons & mirrored texture instead.
  - Place polygons back to back, one uses original texture, one uses mirrored copy of original texture.
  - You can create mirrored texture on the fly at runtime.
- Most back-face polygons should not be drawn
  - Hidden faces of objects overdrawn by other polygons anyway
  - Don't draw polygons with surface normals having positive Z components

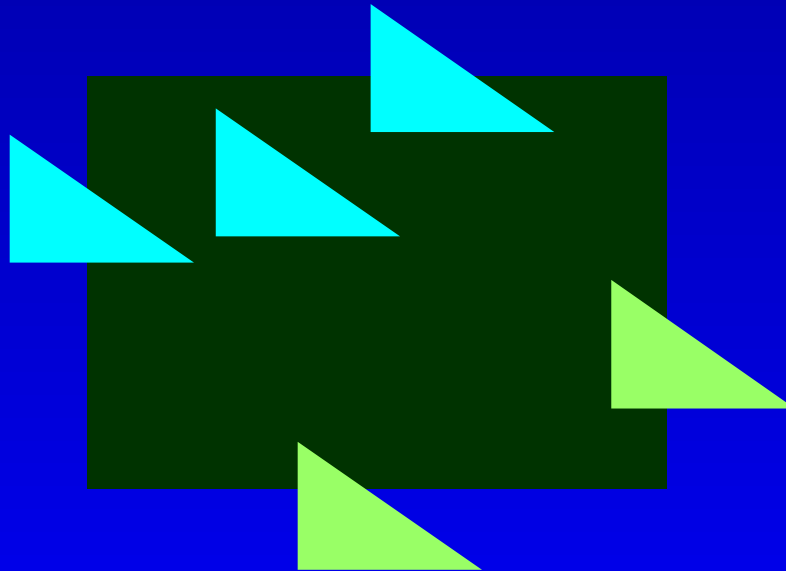
# *Eliminate Unnecessary Drawing*

- ▶ Zero-area polygons
  - GPU preprocessing takes longer.
  - Should be discarded before drawing begins.
- ▶ Polygons outside the screen boundaries
  - Decreases GPU preprocessing performance
  - If CPU process faster than GPU process, apply CPU clipping

# *Eliminate Unnecessary Drawing*

- ▶ Polygons overlapping screen boundaries
  - Subdivide and apply CPU clipping

**Polygons overlapping top or left side of screen take as long to draw as non-clipped polygons**



**Polygons which overlap bottom or right side of screen require only processing for non-clipped portions**

# *Avoid Doubly-Drawn Backgrounds*

- ▶ Typically the entire background is drawn, then certain portions such as the sky or ground are overdrawn by other polygons
  - Don't draw portions of the background you KNOW are going to be covered up later by foreground objects.
  - Don't use **ClearImage()** if you're going to draw the whole screen anyway.



# *Summary of Optimization Methods*

- ▶ Determine if optimizing the GPU process or CPU process, or both, will make the most difference
- ▶ Determine if the desired improvement can be achieved without changing the basic paradigm.
  - Check the number of polygons to be displayed
  - Estimate data access time & processing time.



# *Summary of Optimization Methods*

- ▶ Tune the CPU process
  - Reduce data access penalties and I-cache misses
  - Make sure the R3000 write buffer is used effectively
- ▶ Tune the GPU process
  - Check intervals where no drawing or data transfer takes place.
  - Improve texture cache usage, maybe change texture modes, Reduce GPU preprocessing time
  - Avoid unnecessary drawing of zero-area polygons, unclipped polygons, doubly-drawn backgrounds...

# *Using the Performance Analyzer to Analyze PlayStation Programs*

# *How to use the PA & Read the Results*

## ▶ Sample the data

- Play the game until you get to the portion where performance needs to be improved (i.e. where the frame rate drops)
- Trigger the data capture
  - Press the trigger button plugged into the DTL-H2700
  - Optionally have your program initiate the capture itself by accessing a specific data location which means “trigger performance analyzer NOW”

## ▶ Interpret the data

- View the graphic displays and information provided.

# *Information Provided by the PA*

## ▶ Bus Usage Analysis Graphs

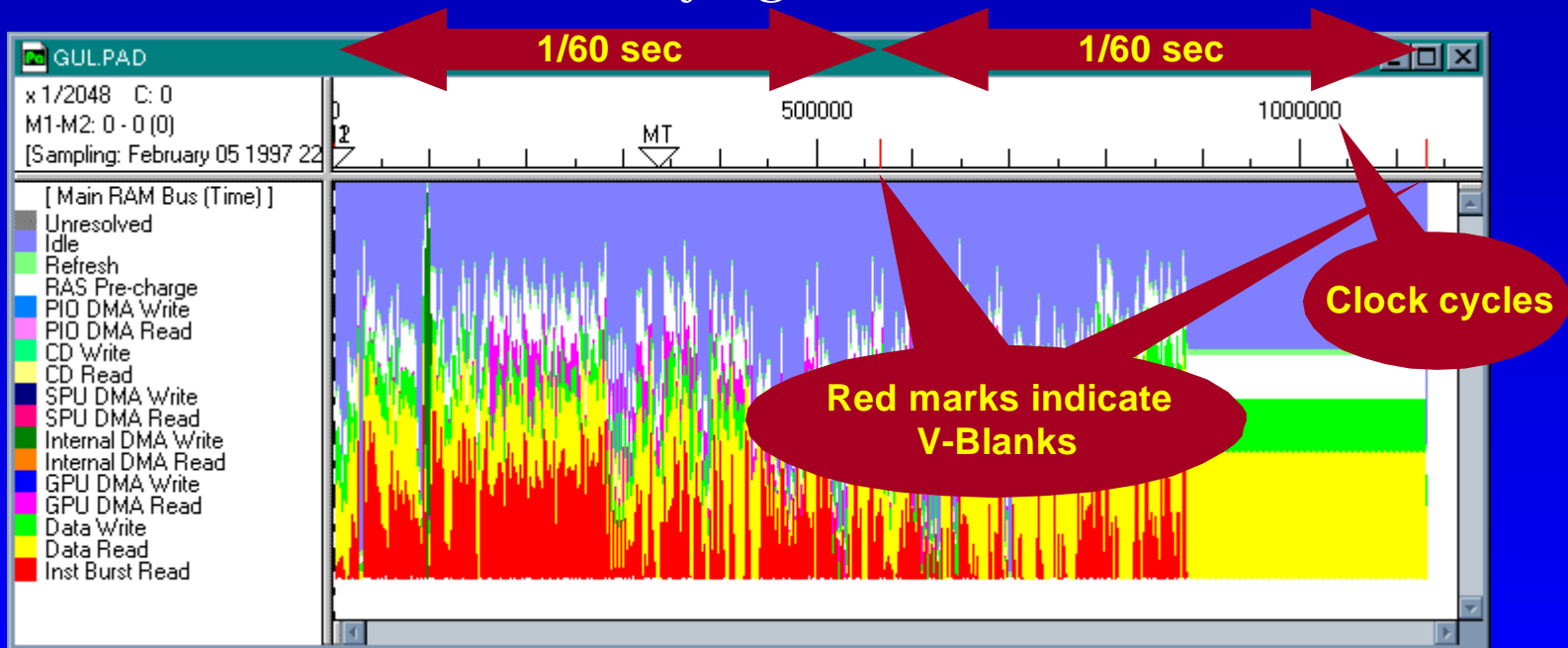
- Main RAM bus
- Sub bus
- Video RAM bus
- GPU packet processing
- RAM Access Penalties
  - Duplicate reads, write buffer flush
- GPU Polygon rendering penalties

# *Information Provided by the PA*

- ▶ Waveforms
  - Shows bus signals such as VBLANK
- ▶ Symbol Access
  - Shows which symbols are being accessed for I-cache burst reads or data reads & writes
- ▶ Data Dump
- ▶ Video RAM contents viewer
- ▶ Statistics

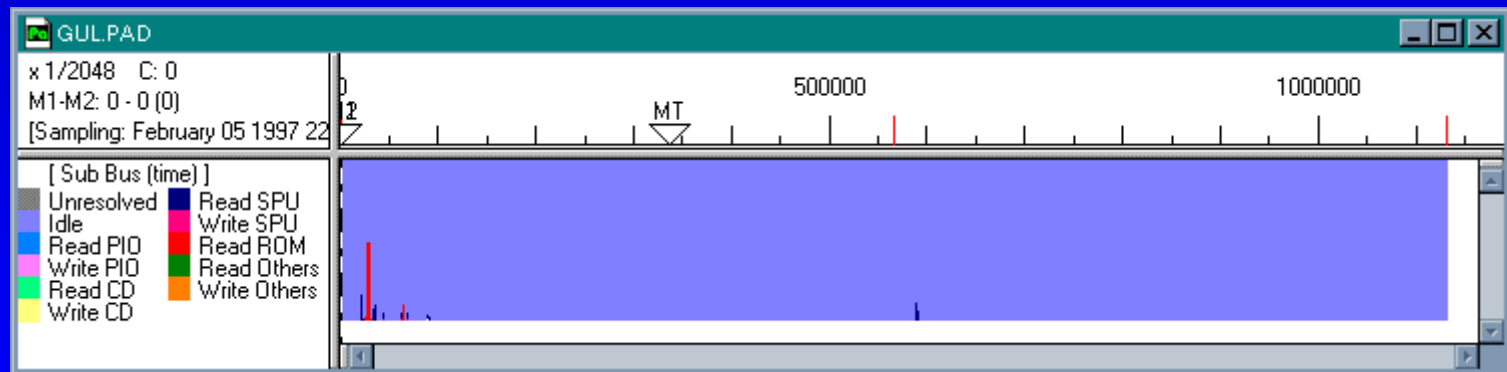
# Main RAM BUS Analysis

- ▶ Shows histogram of an amount of time spent for each transaction type on the main RAM bus.
  - The less **RED**, **GREEN**, and **YELLOW** in the display, the more efficient a CPU process is.
  - Running on cache while accessing the scratch pad RAM results in the CPU staying off the main RAM bus.



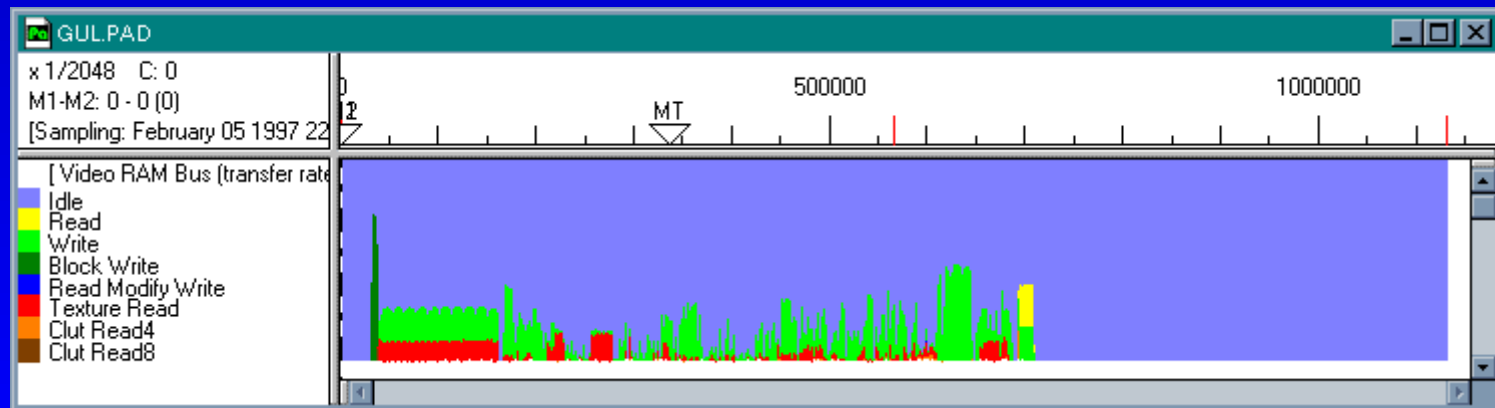
# Sub BUS Analysis

- ▶ Shows histogram of an amount of time spent for each transaction type on the Sub-bus.
  - Used to identify the access to an I/O device, and helps to identify interrupt processes too.



# Video RAM BUS Analysis

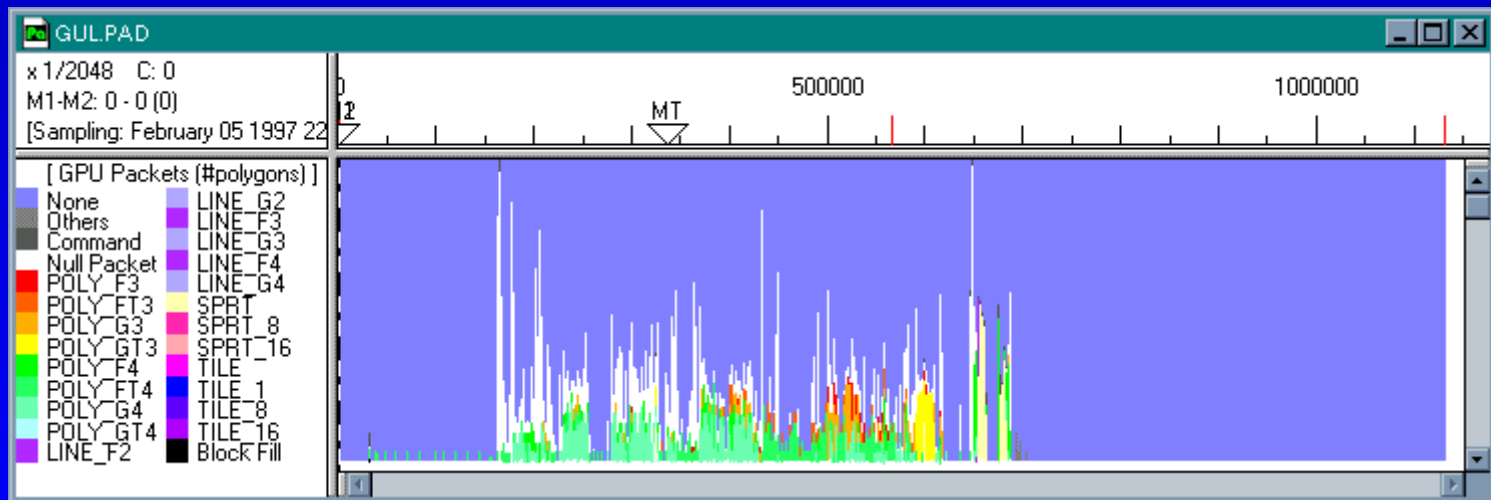
- ▶ Shows histogram of an amount of data transferred in each transaction type on the video RAM bus.
  - Used to tune a GPU process.
  - Higher patterns are, faster the data transfer is.





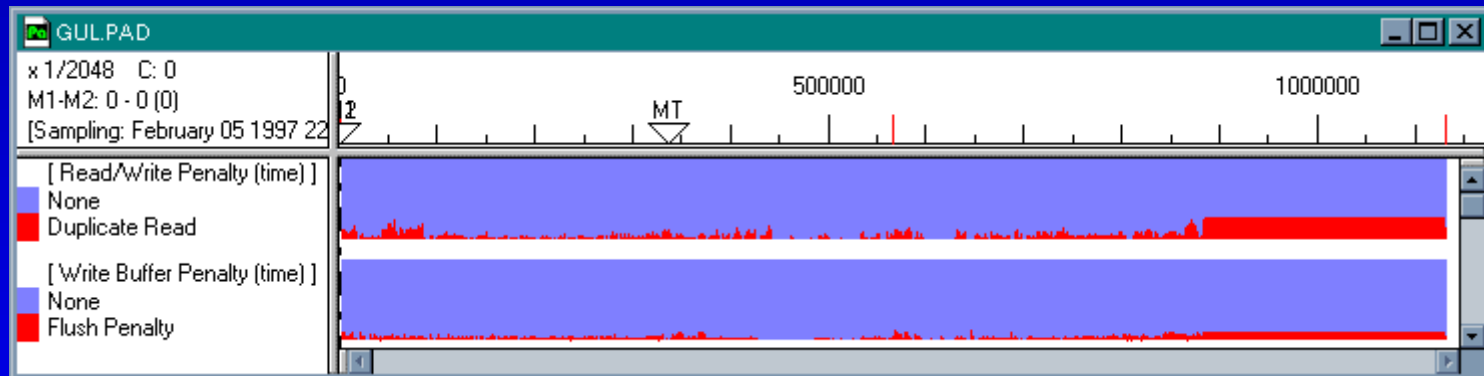
# GPU Packet Analysis

- ▶ Shows histogram of GPU packets which are DMA transferred to the GPU.
  - Used to identify polygon types and detect null packets.
  - Corresponding patterns on the video RAM bus analysis are delayed.
  - This does NOT show timing of the CPU creating the packets and linking them into the ordering table.



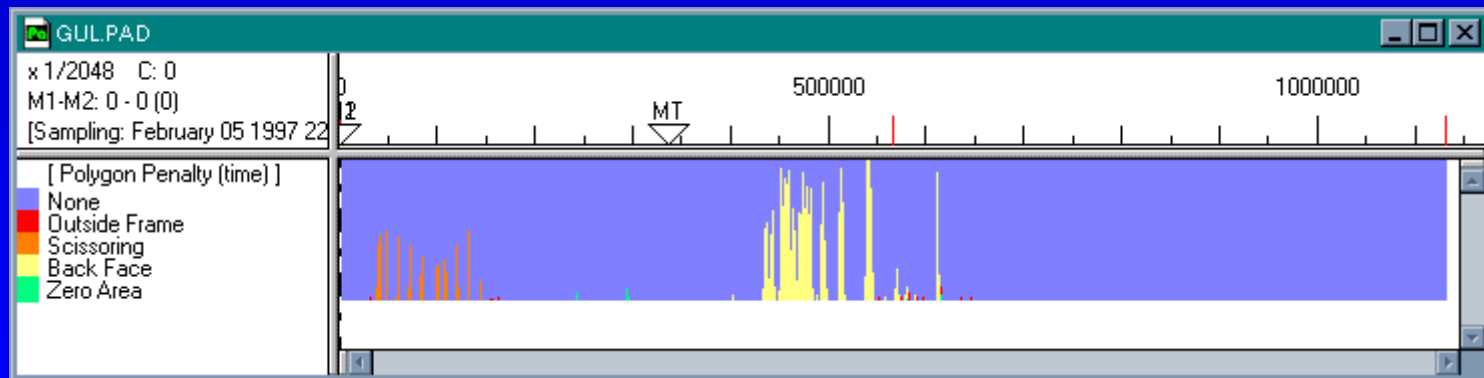
# RAM Access Penalties

- ▶ Shows histogram of CPU stall cycles caused by inefficient memory accesses.



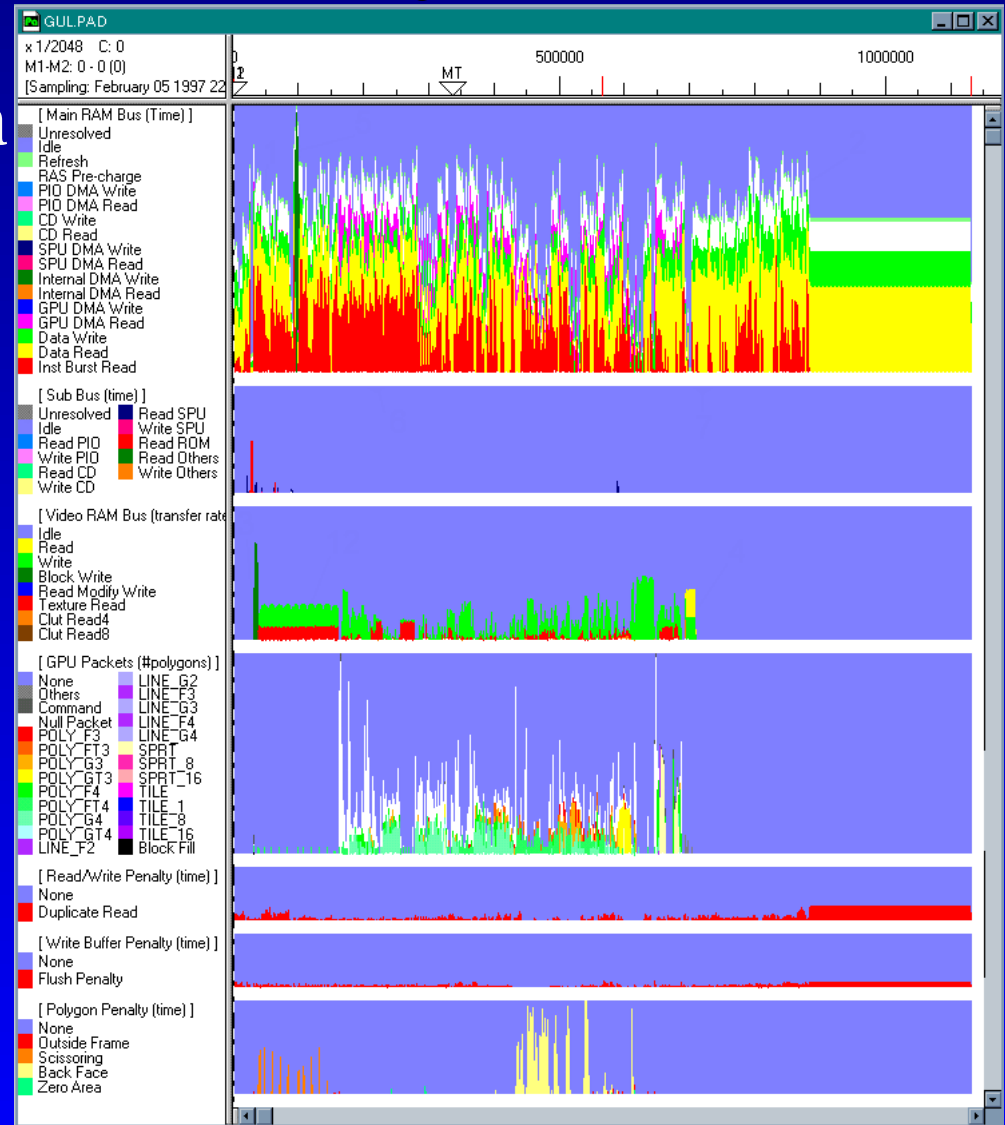
# GPU Polygon Rendering Penalties

- ▶ Shows histogram of each polygon penalty
  - Zero-area polygons
  - Back-face polygons
  - Polygons outside screen area
  - Polygons overlapping screen boundaries (scissoring)



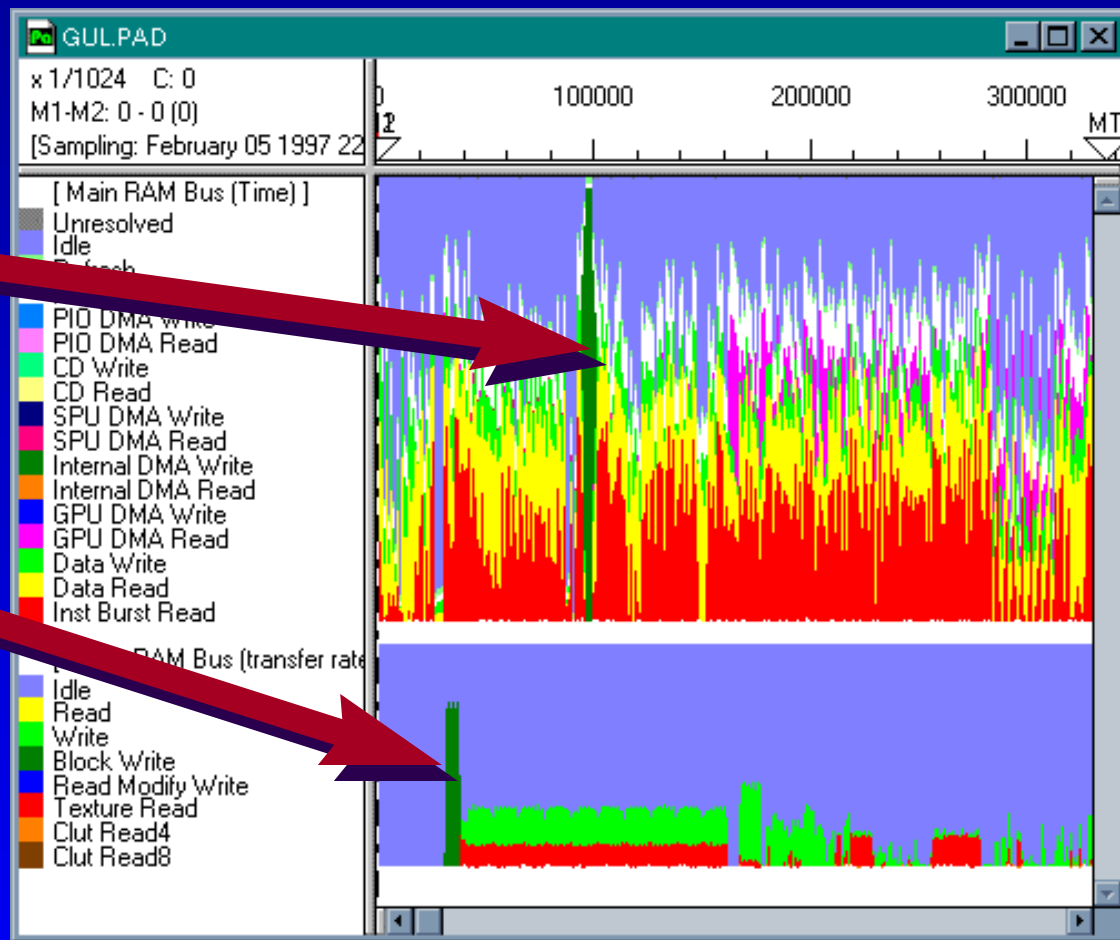
# How to Interpret the Analysis

- ▶ Various phenomena are shown as specific patterns
  - Identify problems and their causes by locating and inspecting those patterns.



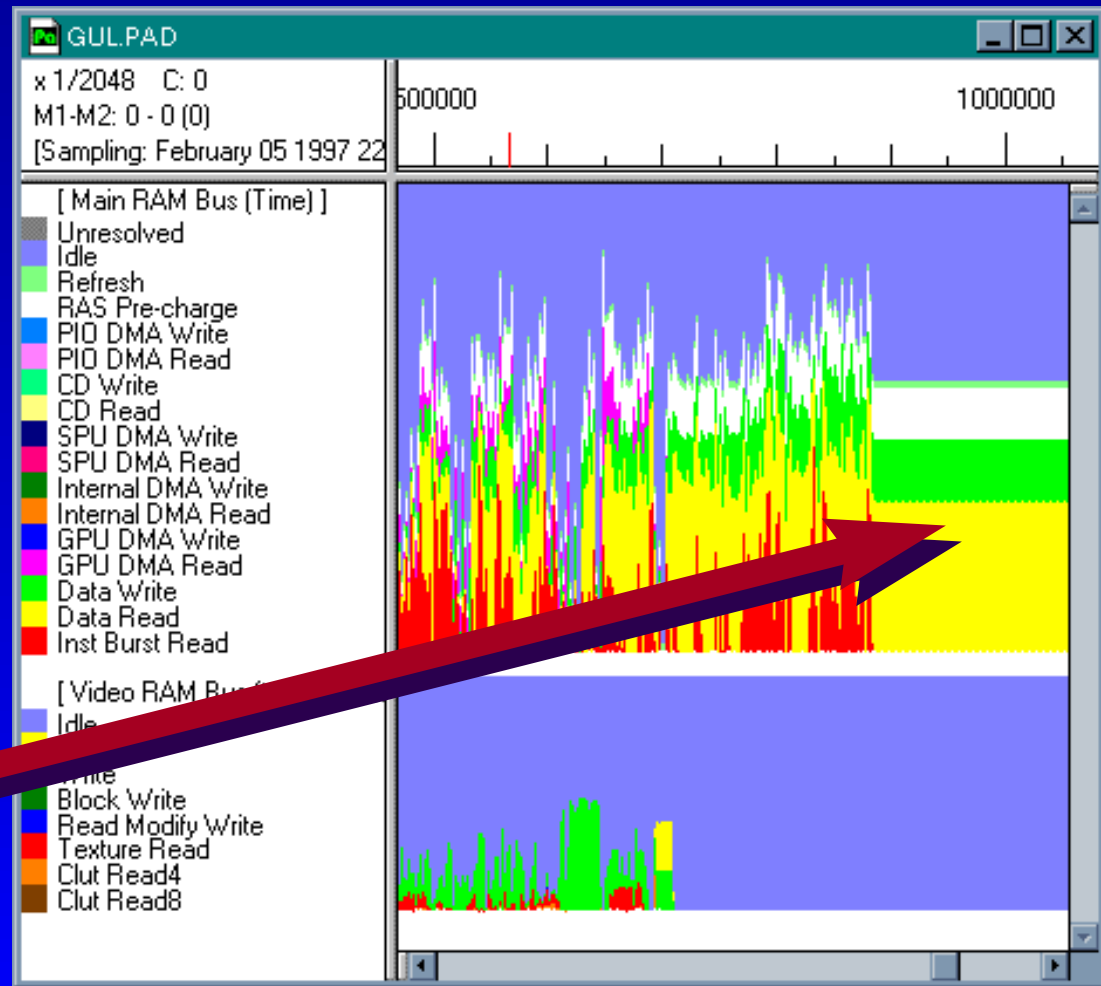
# Start of a CPU Process

- ▶ Spike-like dark green pattern
  - DMA operation performed by **ClearOTagR()**
  - Drawing starts on the video RAM bus analysis
- ▶ Main CPU process starts after VBLANK



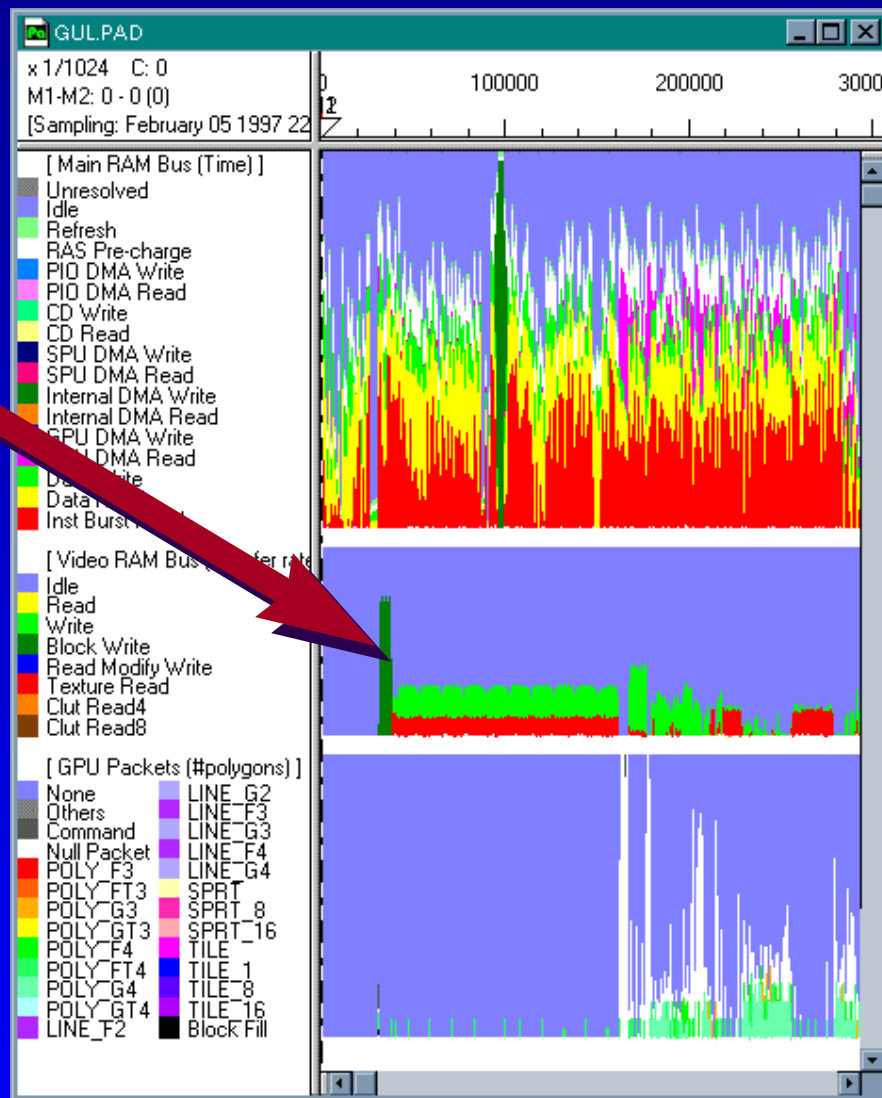
# End of a CPU Process

- ▶ The point where **VSync()** is called
  - A stable read/write pattern indicating a polling loop, such as done by **DrawSync()** or **VSync()** leading up to **VBLANK**



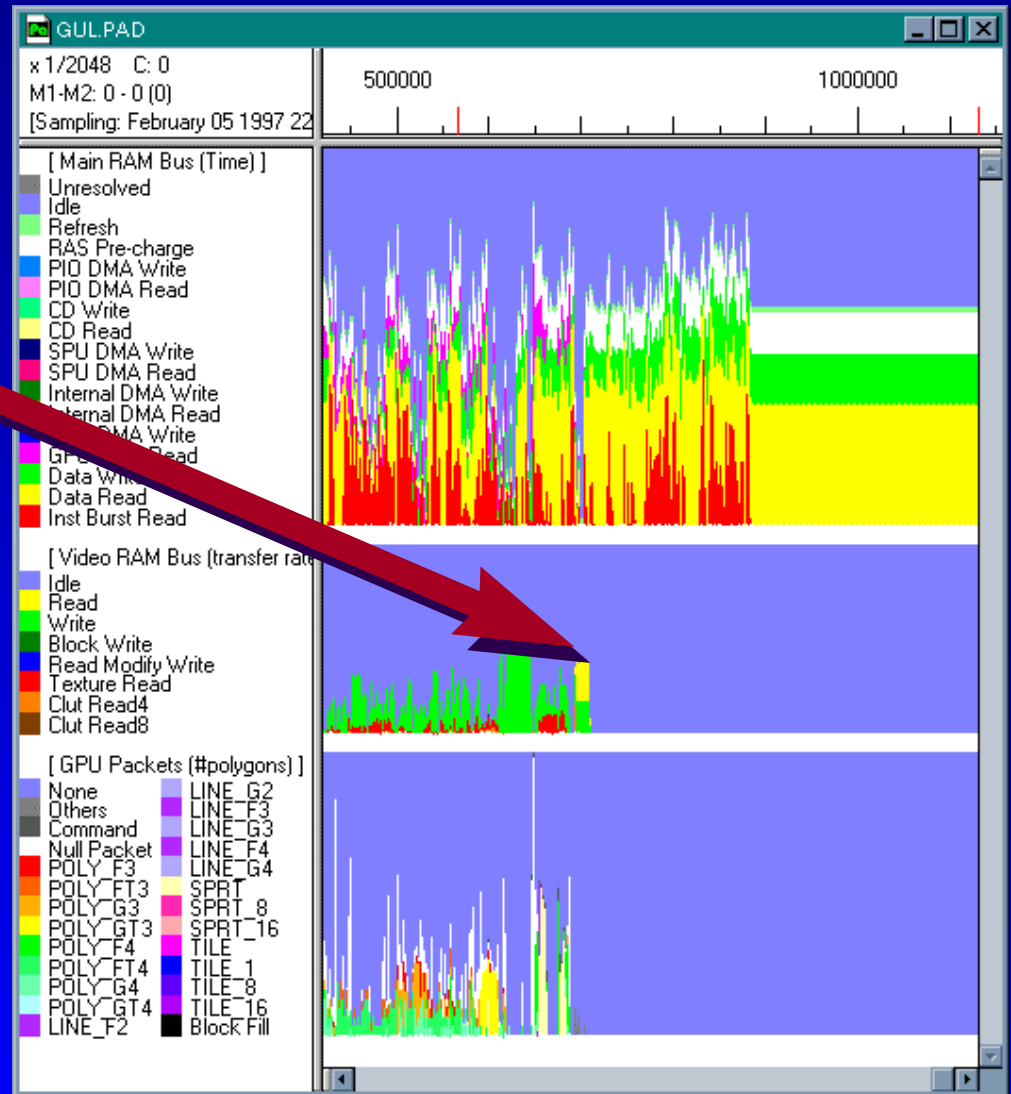
# Start of a GPU Process

- ▶ Drawing a background



# End of a GPU Process

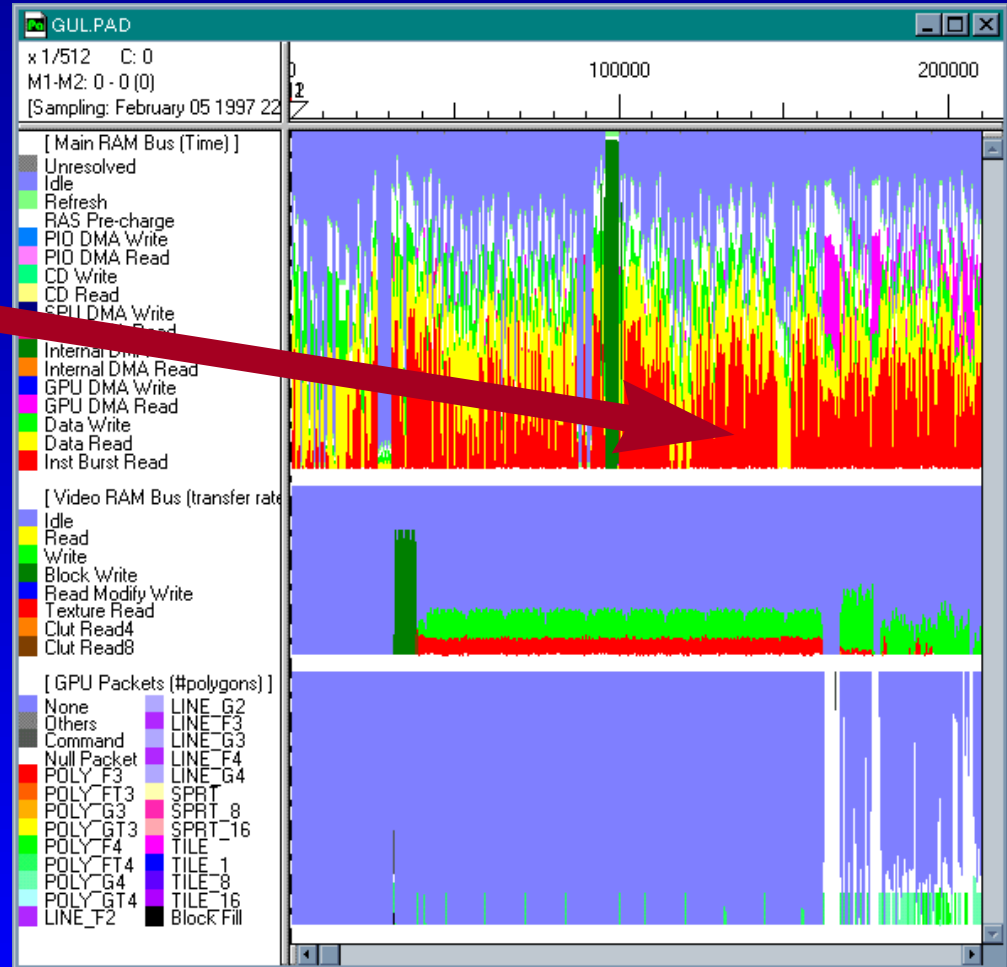
- ▶ The last drawing done in a frame





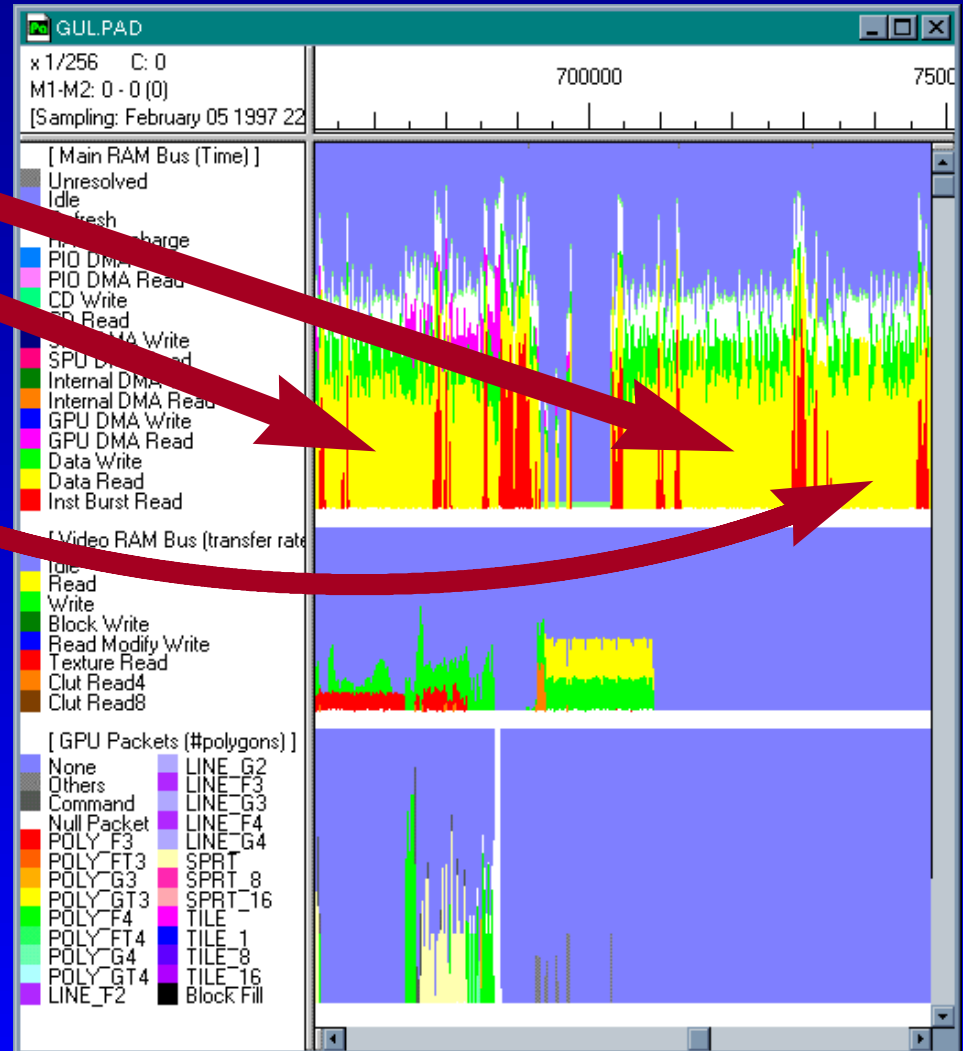
# Instruction Cache Misses

- ▶ Red area indicates instruction burst reads caused by instruction cache misses
- ▶ The CPU stalls for 70% of the time shown in red



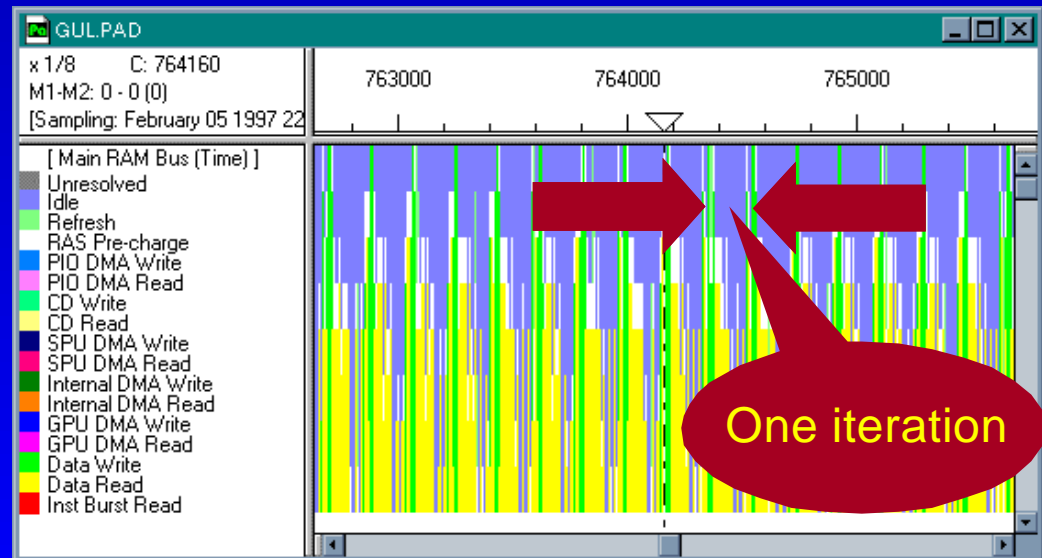
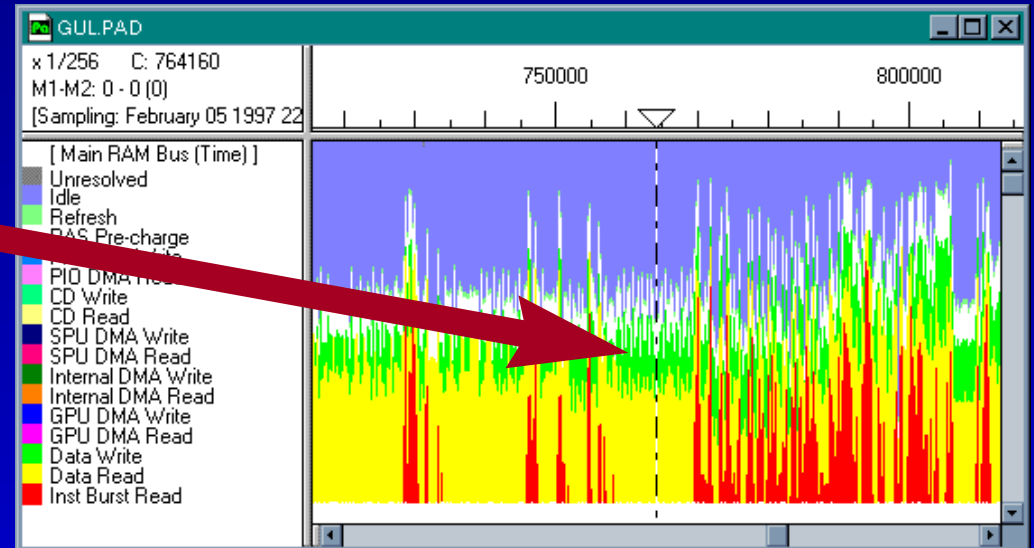
# Instructions Running On-cache

- ▶ Patterns without red



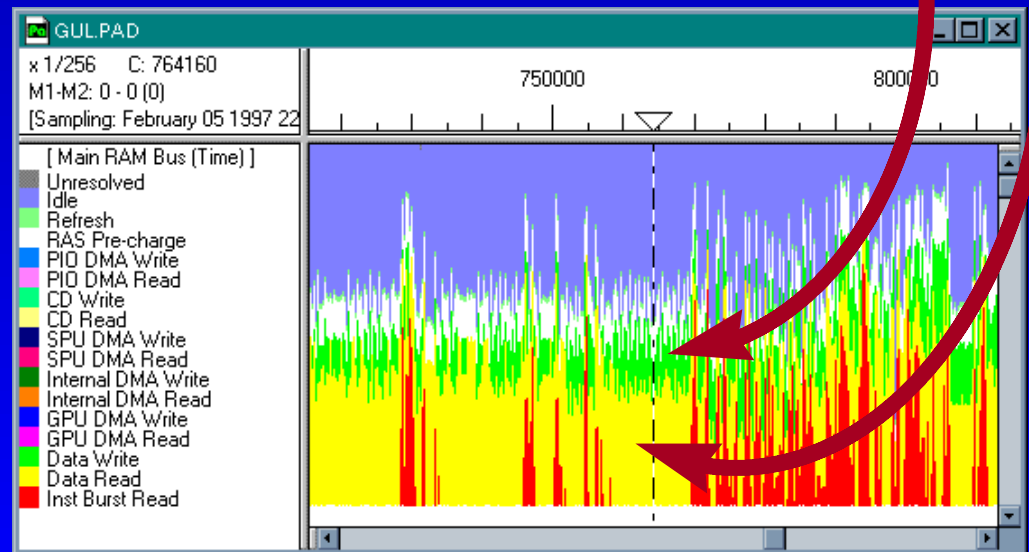
# Processing Loops On-cache

- ▶ A stable pattern means a loop is being processed.
- ▶ Each iteration of the loop can be seen by enlarging the display



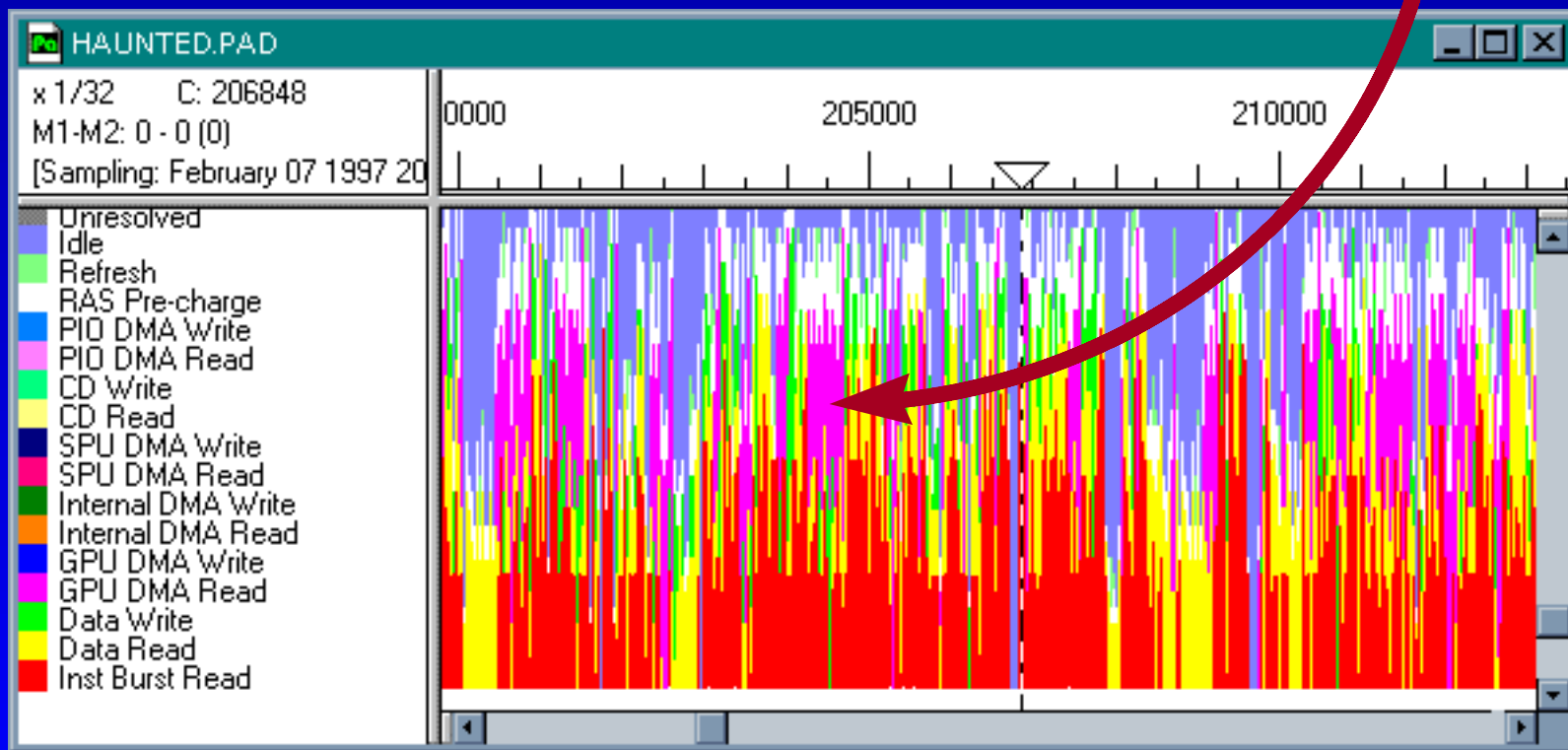
# CPU Data Read and Write Cycles

- ▶ Reading data from RAM by the CPU is colored yellow.
- ▶ Writing to RAM by the CPU is colored green.
  - The CPU does not stall if write buffer is not full.
- Accessing the scratchpad RAM does not appear on the main RAM bus analysis.



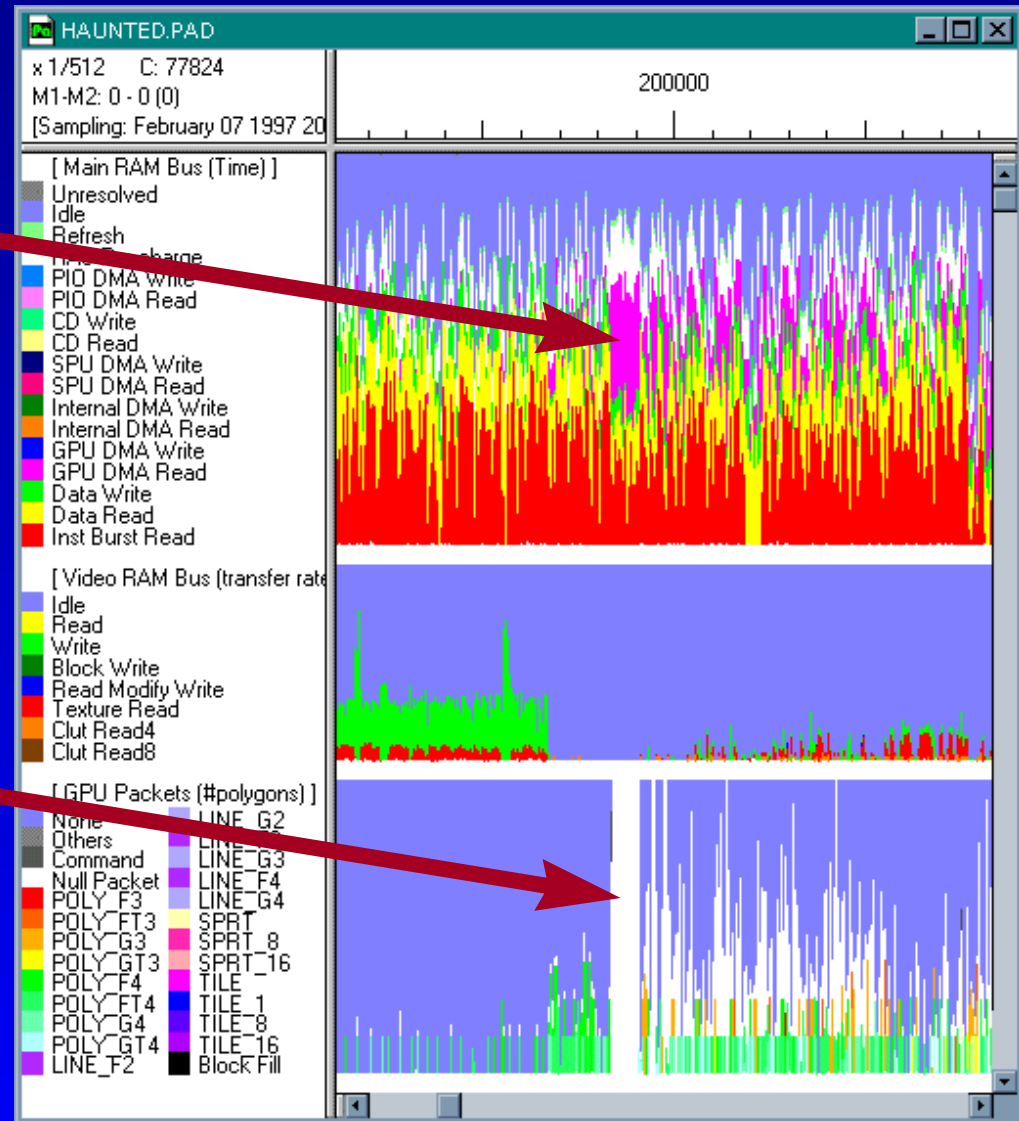
# Packets Transferred to GPU

- ▶ A pink pattern in the main RAM bus analysis indicates a GPU packet DMA transfer.



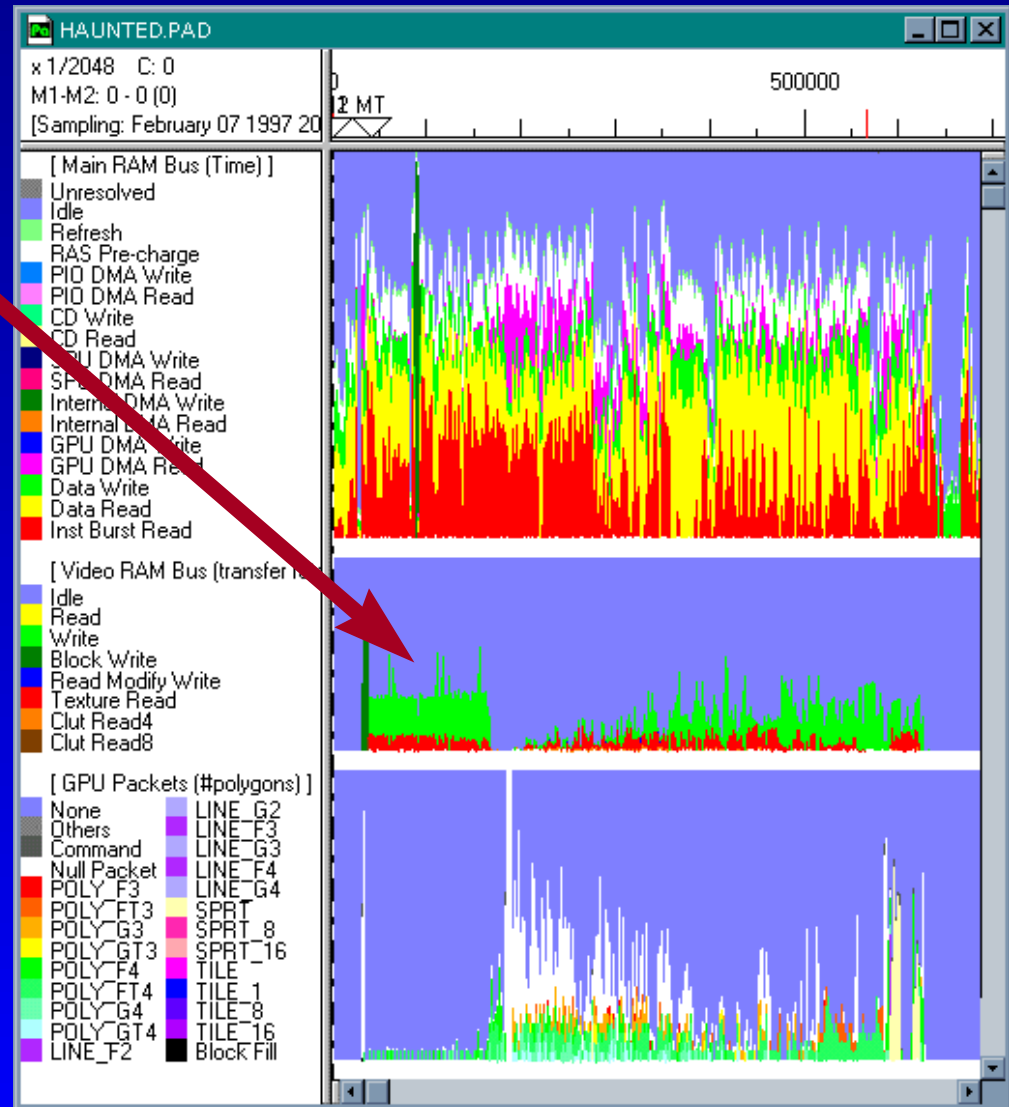
# NULL GPU Packets

- ▶ A large pink lump indicates that successive NULL packets are being transferred
- ▶ Blocks of successive null packets are also shown as a continuous white pattern in the video RAM bus analysis



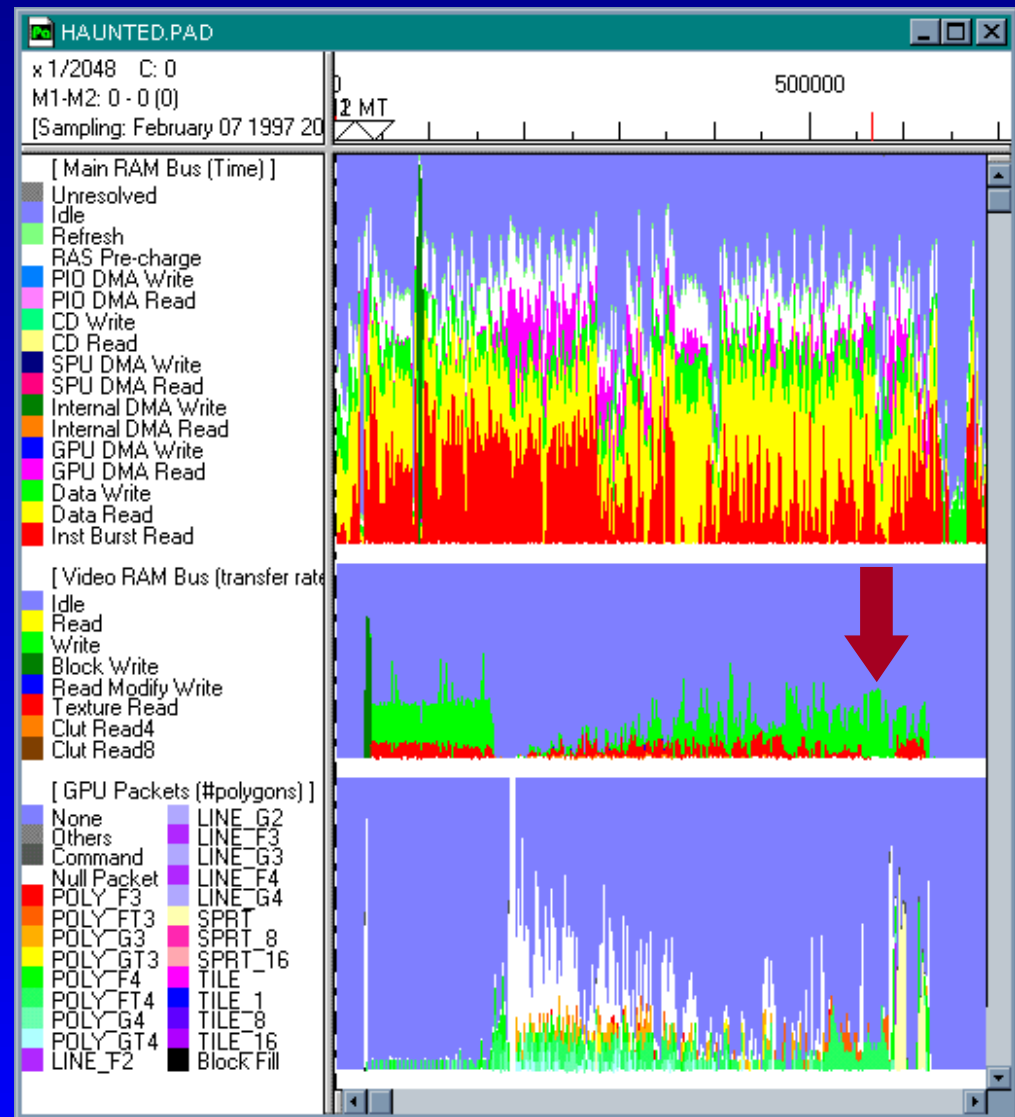
# Drawing a Background

- ▶ The first drawing pattern.
- ▶ The height (indicating transfer rate) is usually consistent.



# Efficient Drawing Pattern

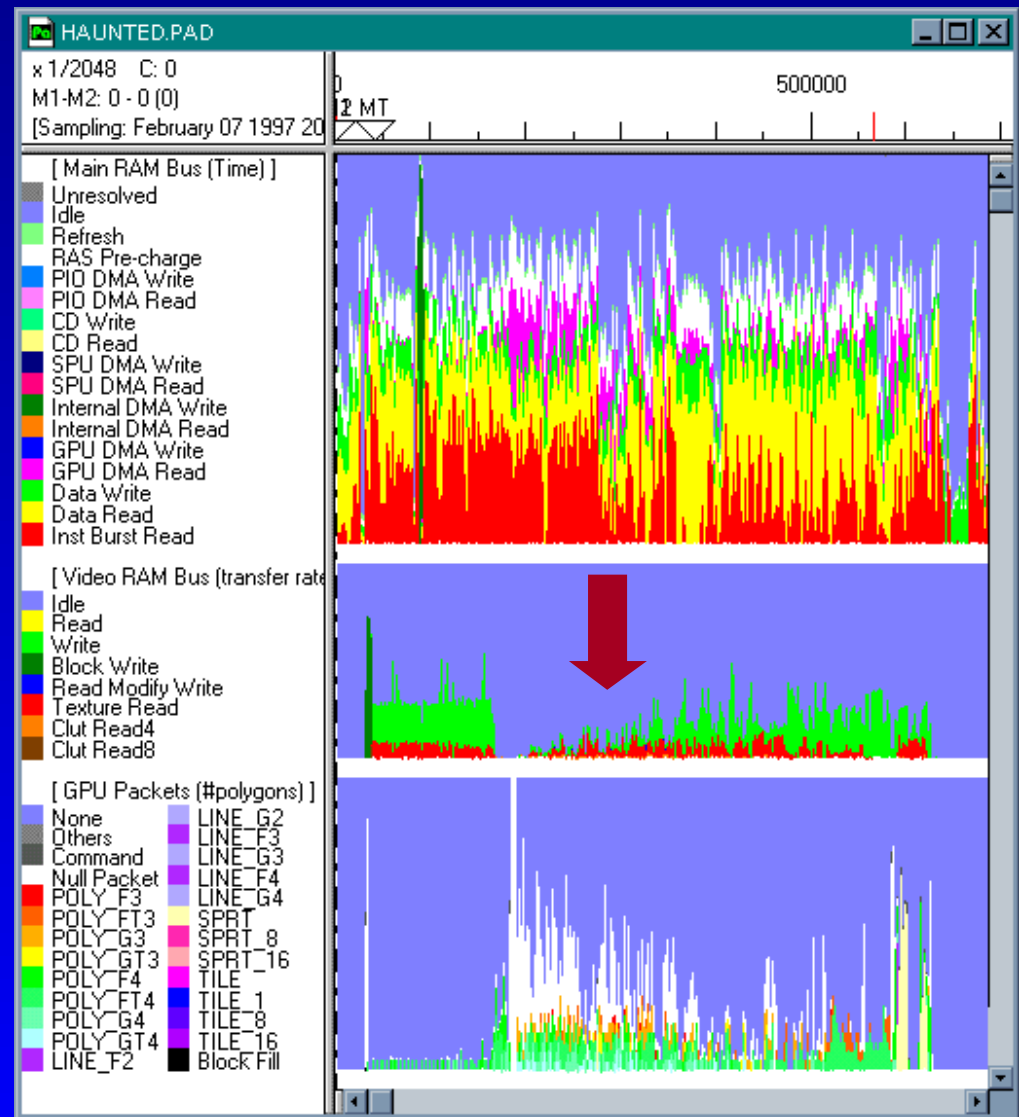
- ▶ A high green pattern
- ▶ Good green-to-red ratio
  - Red indicates texture cache reads





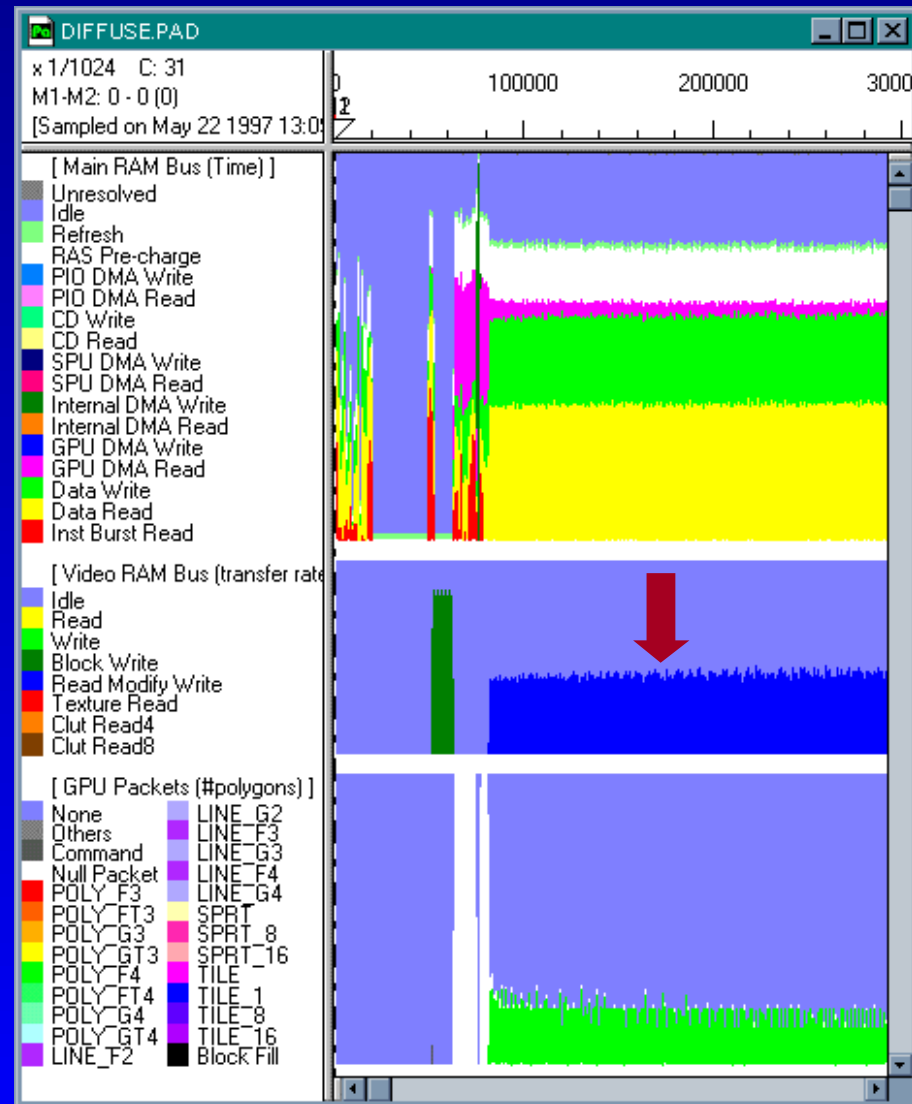
# Inefficient Drawing Pattern

- ▶ A low green pattern
- ▶ Poor green-to-red ratio
  - Red indicates texture cache reads



# Drawing Semi-transparent Colors

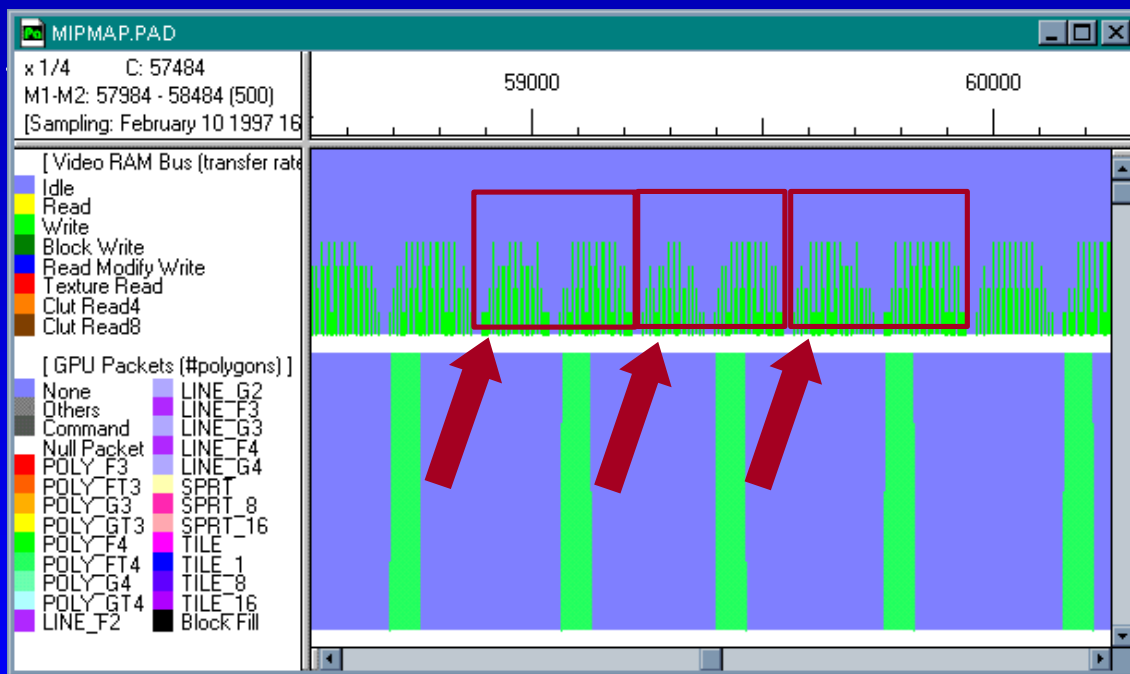
- ▶ Drawing of semi-transparent pixels requires a read-modify-write cycle, shown in blue.
  - Shown with texture on cache
- ▶ The overall colored area indicates the sum of all reads and write accesses.



# Relationship between GPU Packets Processing & Video RAM Bus Accesses

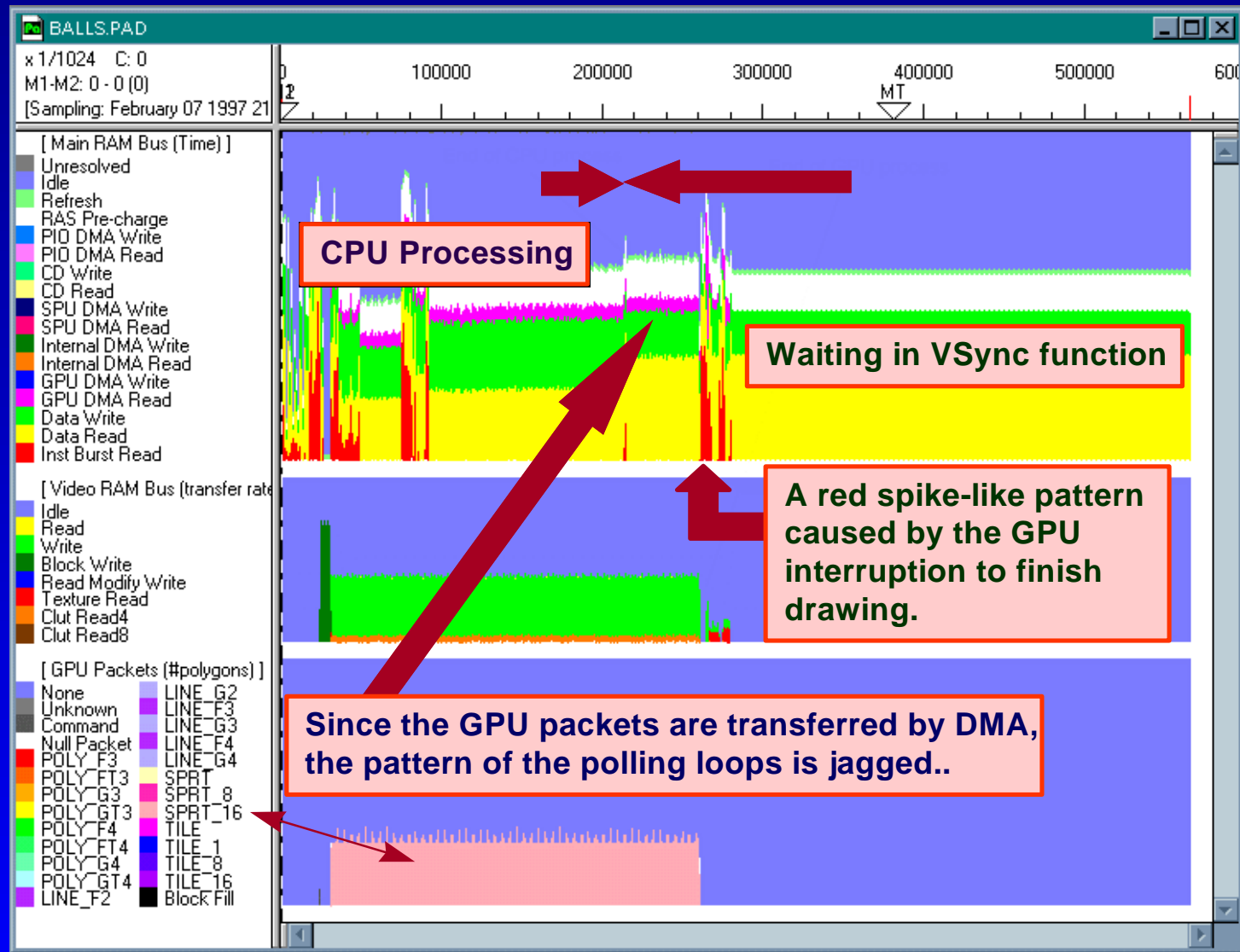
- ▶ In the video RAM bus analysis, drawing patterns appear after the corresponding GPU packet shown in the GPU packet analysis display.

- Delay



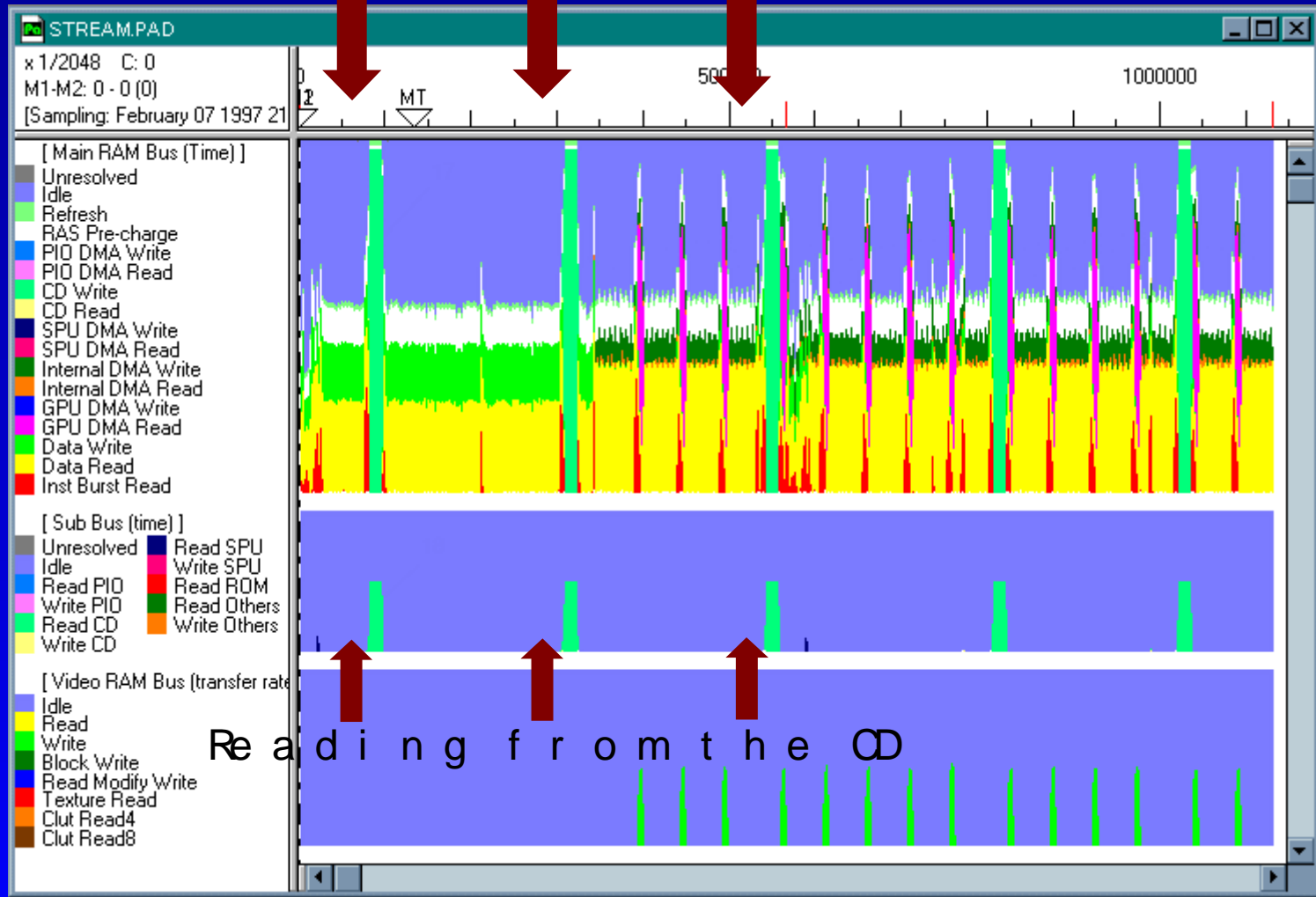
GPU

# GPU Processing -vs- CPU Processing

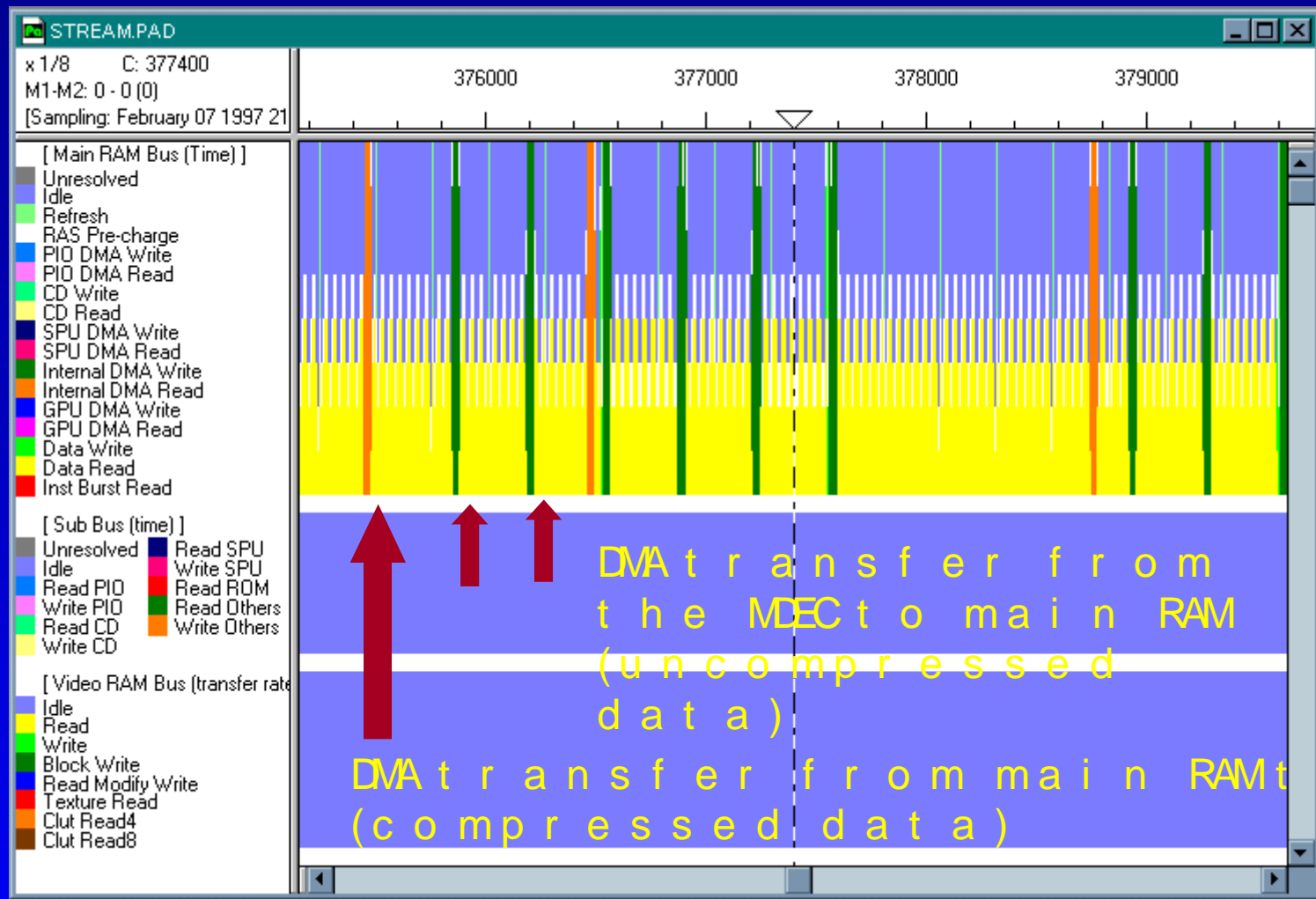


# Accessing the CD

DMA transfer from the CD to the main



# Accessing the MDEC



Note: An access to the MDEC is not shown in

# Data Dump

Instructions

Means a long-word access

Coordinates in the video RAM

Function names

Instruction burst read

Address

Data value

Means texture reads

Prev	Next	Go	Marker	Page:																
P	MR	N	N	M	S	O	MMMM	S	S	V	V	V	V	V	V	V	V	V	G	RDC
B/	W	B	A		Y	F	WWWW	B	A	B	M	M	M		H	M	H	U	XST	
S	O	Y	D		F		EEEE		L									N	DRS	
S	T	D			L	S	NNNN	S		N	M	A	X		A	X	Y	I	111	
T	D	R			O	E	3210	T		K	O	C	O		C	1	1	N	***	
A					L		****	A		D	C				C			T		
T								T		E	O				1			*		
196105:	IR			80016CE0	_exeque+0001FC	AC22C	----			0	TXR							0	111	
196106:	IR			80016CE4	_exeque+000200	3C0380	----			0	TXR	R(960, 268)	R(961, 268)					0	111	
196107:	IR			80016CE8	_exeque+000204	8C63CE	----			0	TXR								0	111
196108:	IR			80016CEC	_exeque+000208	3C0280	----			0	TXR	R(962, 268)	R(963, 268)					0	111	
196109:	H						----			0	TXR								0	111
196110:	DW	1	4				----			0	TXR								0	111
196111:	DW						----			0									0	111
196112:	DW						----			0									0	111
196113:	DW					00000001	www			0	R								0	111
196114:	DW			8002CB00	_gout+000000		www			0	R	W(114, 44)	W(115, 44)					0	111	
196115:	H						----			0	R								0	111
196116:	IR	4					----			0	R	W(118, 44)						0	111	
196117:	IR						----			0	R								0	111
196118:	IR						----			0	R								0	111
196119:	IR			80016CF0	_exeque+00020C	8C42CB00	----			0	R								0	111
196120:	IR			80016CF4	_exeque+000210	00000000	----			0									0	111
196121:	IR			80016CF8	_exeque+000214	10620009	----			0									0	111
196122:	IR			80016CFC	_exeque+000218	00000000	----			0									0	111
196123:	H						----			0									0	111
196124:	DR	1					----			0									0	111
196125:	DR						----			0									0	111
196126:	DR						----			0									0	111
196127:	DR			8002CBCC	_qin+000000	00000001	----			0									0	111
196128:	H						----			0									0	111
196129:	R						----			0									0	111
196130:	R						----			0									0	111

# Statistics — Main RAM Bus

- ▶ Get Statistics for the duration of a 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

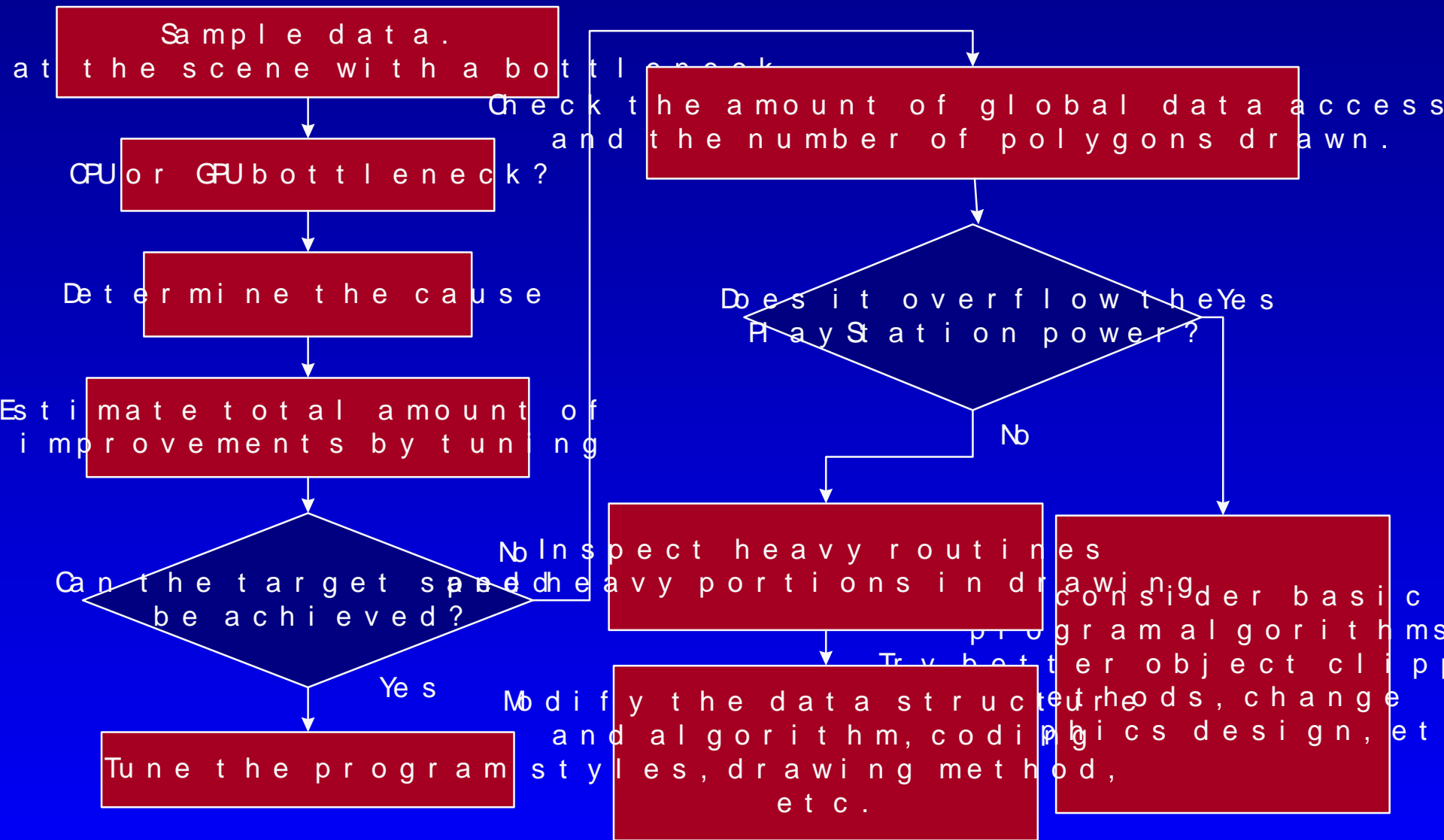


# Statistics — GPU Packets

- ▶ Get statistics for the duration of a GPU process.
- ▶ Look for NULL packets
- ▶ Check the number of polygons for each type.

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

# Flow of Analysis



# *Optimization Strategy*

- ▶ Assign a priority to each method of optimization, and apply only those which are necessary.
  - Start with the optimization which should provide the most significant improvement.
    - It is enough to get the target speedup?
- ▶ Put emphasis on portions where great improvements are expected with less work.
  - A small loop with many iterations, which takes long time to be processed.

# *Optimization Strategy*

- ▶ Check to see if the target speed is achievable after performing each improvement.
  - For example, the CPU can not run faster than the level where no cache miss occurs, so figure out potential improvement gained just by eliminating cache miss.
  - If the CPU process takes significantly longer than the GPU process, then optimizing the GPU process won't have much effect on your game's framerate. Or vice versa.



**Estimation of speedup is important!**

# *How to Sample Program Execution*

- ▶ Push the trigger button connected to the back of the DTL-H2700
  - Can also use the **Trigger** function.
- ▶ Resetting the Sample Range
  - Once you've captured data and determined which range makes up a complete game frame, change the sampling range in the **Options** dialog box
  - Download the data again without re-triggering by using the **Read Data Only** menu option
    - Eliminates unnecessary sample data.
    - The PA software will run faster with less data to examine.

# Using Set Trigger Condition

- Specify trigger address, function name, data value, and how many video frames BEFORE trigger condition to sample.

Set Trigger Condition

A Kind of Trigger Condition

Condition	Invert	Device Signal	Invert
<input checked="" type="checkbox"/> Trigger Switch		<input type="checkbox"/> GUNINT (Gun)	<input type="checkbox"/>
<input checked="" type="checkbox"/> Main RAM Address Bus		<input type="checkbox"/> RXD1 (Link Cable)	<input type="checkbox"/>
<input type="checkbox"/> Main RAM Data Bus	<input type="checkbox"/>	<input type="checkbox"/> DSR1 (Link Cable)	<input type="checkbox"/>

OK  
Cancel  
Save  
Default

Main RAM Address Bus

Symbol List

Address (in HEX) 80014F68  
Mask (in HEX) FFFFFFFF

Main RAM Data Bus

Value (in HEX) 00000000  
Mask (in HEX) FFFFFFFF

the number of V-blanks prior to trigger

1

Read/Write, Size

by Data Type

by Data Type

Read or Write  Read  Write

Byte  Half Word  Long Word

by Hardware Signal

by Hardware Signal

3210(byte)     uncheck: Write  
Mask     check: Read

★ Setting the trigger address to a function such as **DrawOTag** enables sampling to synchronize with the start of a game frame.

Symbol List

Begin Address 80014F68 Stop Address 80014FDB

OK  
Cancel

Symbol List (Address Order)

Symbol List (Alphabetical Order)

Symbol List (Address Order)	Symbol List (Alphabetical Order)
DrawOTag	DrawOTag
DrawOTag	DrawOTag
PutDrawEnv	DrawOTagEnv
DrawOTagEnv	DrawPrim
GetDrawEnv	DrawSync
PutDispEnv	DrawSyncCallback
GetDispEnv	DSTACK
GetODE	DumpClut
SetTexWindow	DumpDispEnv
SetDrawArea	DumpDrawEnv

# Capturing Where the Frame Rate Drops

- ▶ Set the trigger address to a variable named *ReqTrigger*
  - If *ReqTrigger* is accessed, PA will trigger capture of data
- ▶ Many types of phenomena can be captured under program control

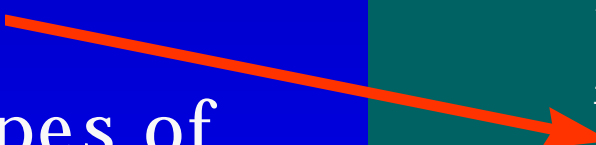
```
/* frame length in H count
   (needs an adjustment) */

#define HCountThreshold  525
volatile long ReqTrigger;

main()
{
    for(;;)          /* main loop */
    {
        /* compare H-counts since */
        /* the last Vsync exited */

        if( VSync(1) > HCountThreshold )
            ReqTrigger = 0;

        VSync(0);
        ...
    }
}
```



# *Examples of Analysis*



# *How to Use Statistics*

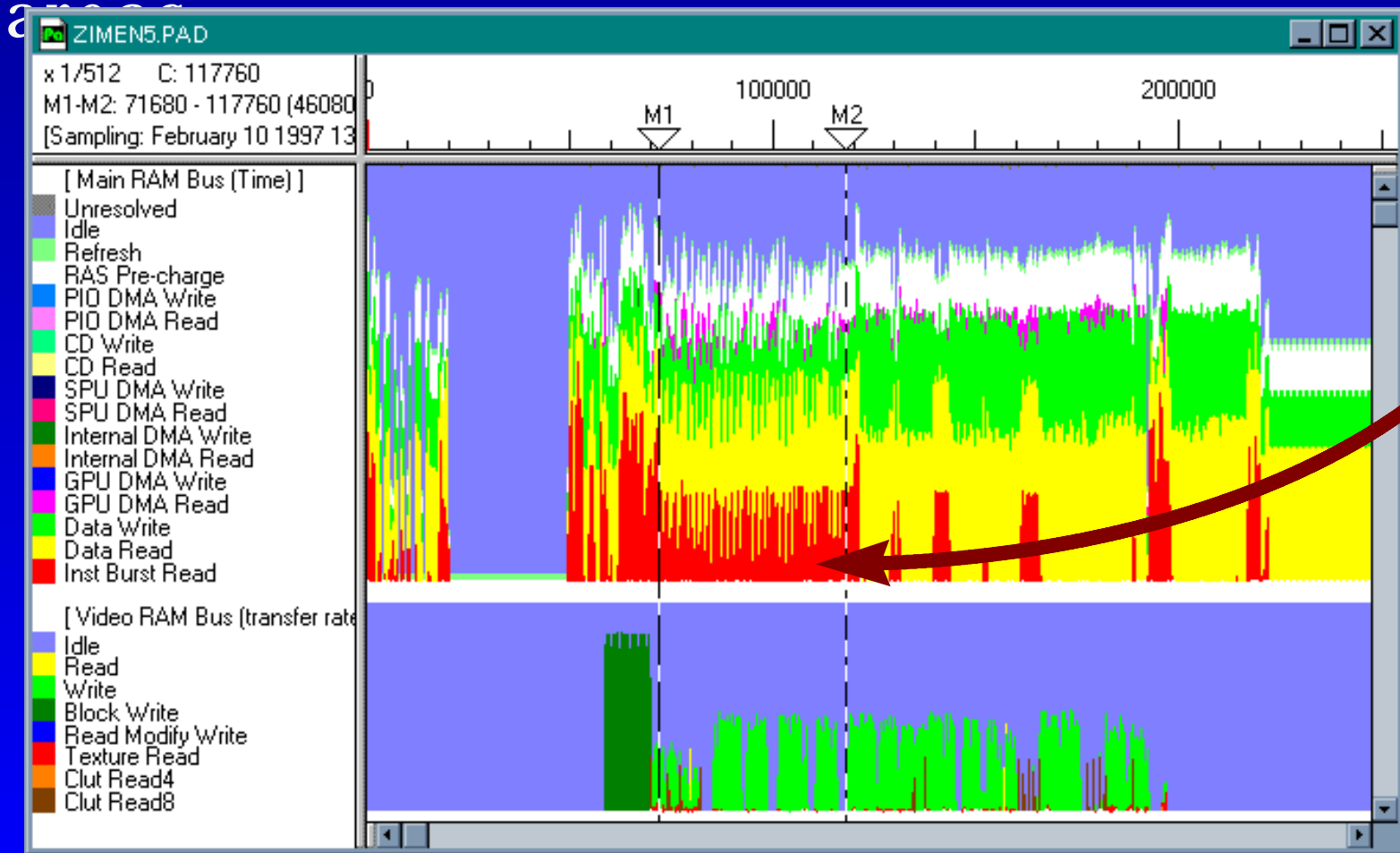
- ▶ Get total statistical amount for each of CPU and GPU processes.
- ▶ Estimate total amount of CPU stall cycles.
  - How much improvement can be achieved at most by eliminating cache miss, etc.
  - Check the number of GPU packets being processed.
    - Is it overloading the GPU?
    - Too many small polygons with large textures which take long time in preprocessing?
    - Too many backface, offscreen, or partially-clipped polygons which take long time in preprocessing?

# *How to Use Statistics*

- ▶ Get a statistical amount in a particular range.
  - Statistics at particular iterations.
    - This will give how much improvement can be achieved eliminating cache miss, etc.
  - Statistics at a portion where drawing is inefficient.
    - Because of the number of polygons?

# Finding Code Causing I-cache Misses

- ▶ Check loop processing patterns with red



# Finding Code Causing I-cache Misses

- ▶ Determine estimate of CPU stall cycles caused by I-cache burst reads

ZIMEN5.PAD [Sampling: February 10 1997 13:01:21]  
Range: 71680 - 117760

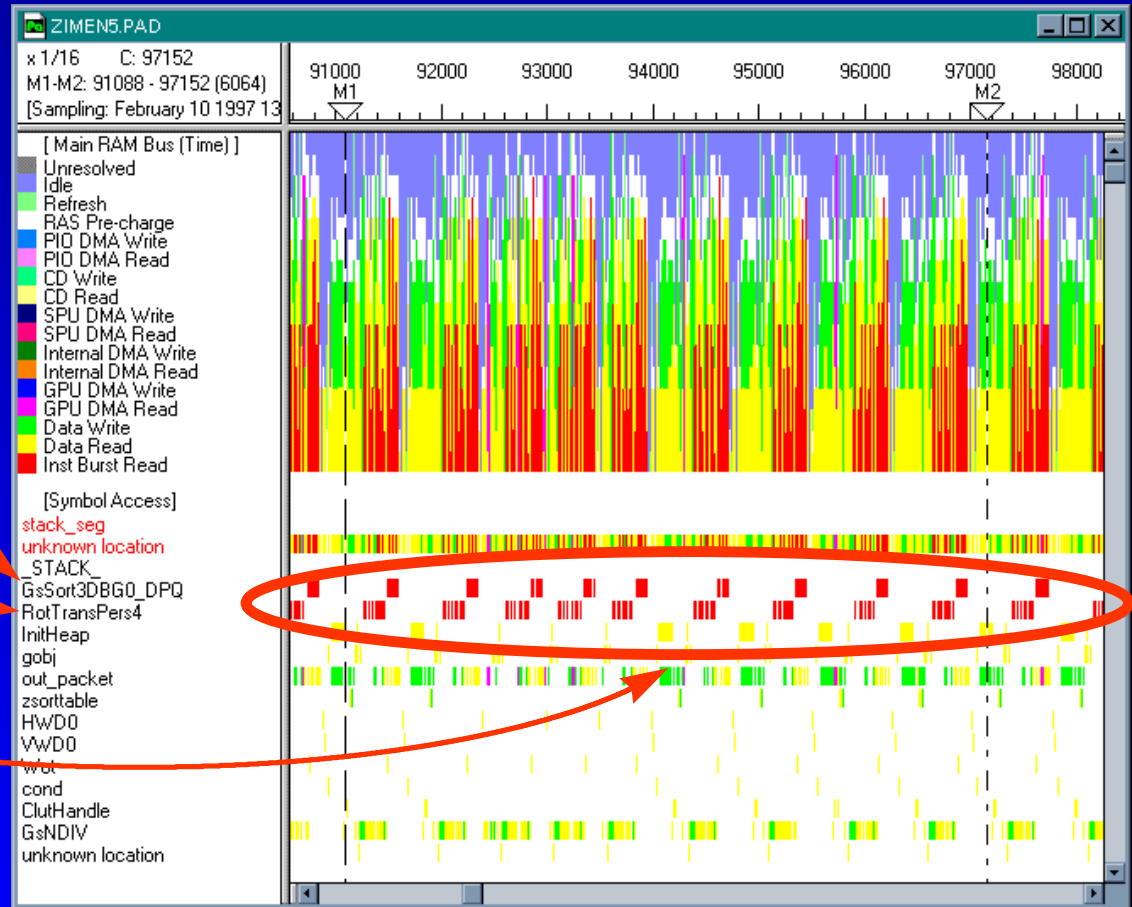
Main Memory Bus:

	Time(%)	Bytes	Speed(MB/sec)	Clock	Cycles/word	Estimated CPU Stall Cycles
unresolved	0.0	----	----		----	-----
IDLE	24.9	----	----		----	-----
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	2764	83.68		----	-----
DATA WRITE	14.6	4828	24.37		----	-----
DATA READ	27.4	12636	34.01		4.0	12633
I-BURST READ	16.7	17392	76.69		1.8	4406

# Finding Code Causing I-cache Misses

- ▶ Read the symbol file & show symbol accesses

Two functions cause cache miss alternately.



# Finding Code Causing I-cache Misses

- ▶ Cache line = 2nd & 3rd least significant hex digits of instruction address

DUMP ZIMEN5.PAD Page: 927

Prev	Next	Go to Marker	Page: 927				
P	MR	N	N	M	S	O	D
O	B/	W	B	A	Y	F	A
S	W	O	Y	D	M	F	T
S	R	T	D	E	B		
T	D	E	R				
A							
T							

92620:	DW	1	4				
92621:	DW						
92622:	DW						
92623:	DW						
92624:	DW			807FFE64	GsNDIV+798C34	807FFF14	
92625:	H						
92626:	IR		4				
92627:	IR						
92628:	IR						
* 92629:	IR			80012550	RotTransPers4+000008	C8A20000	
92630:	IR			80012554	RotTransPers4+00000C	C8A30004	
92631:	IR			80012558	RotTransPers4+000010	C8C40000	
92632:	IR			8001255C	RotTransPers4+000014	C8C50004	
92633:	H						
92634:	DR		1				
92635:	DR						
92636:	DR						
92637:	DR			807FFE68	GsNDIV+798C38	0C000000	
92638:	H						
92639:	H						
92640:	H						
92641:	DR		1				
92642:	DR						
92643:	DR						
92644:	DR			807FFE6C	GsNDIV+798C3C	E3180000	

DUMP ZIMEN5.PAD Page: 929

Prev	Next	Go to Marker	Page: 929				
P	MR	N	N	M	S	O	D
O	B/	W	B	A	Y	F	A
S	W	O	Y	D	M	F	T
S	R	T	D	E	B	S	A
T	D	E	R		O	E	
A					L	T	
T							

92855:	DR						
92856:	DR						
92857:	DR			800658C4	Wot+000018	8006572C	
92858:	DR						
92859:	DR						
92860:	DR						
92861:	DR						
92862:	DR						
92863:	DR						
92864:	IR						
92865:	IR						
92866:	IR						
* 92867:	IR			80012550	GsSort3DBG0_DPQ+000540	0086102A	
92868:	IR			80012554	GsSort3DBG0_DPQ+000544	10400002	
92869:	IR			80012558	GsSort3DBG0_DPQ+000548	00000000	
92870:	IR			8001255C	GsSort3DBG0_DPQ+00054C	00C02021	
92871:	H						
92872:	H						
92873:	IR		4				
92874:	IR						
92875:	IR						
92876:	IR			80012560	GsSort3DBG0_DPQ+000550	96620008	
92877:	IR			80012564	GsSort3DBG0_DPQ+000554	86680010	
92878:	IR			80012568	GsSort3DBG0_DPQ+000558	00026400	
92879:	IR			8001256C	GsSort3DBG0_DPQ+00055C	000C4C03	
92880:	H						
92881:	DR		1				
92882:	DR						

# Identifying Functions Running on Cache

- ▶ Function names not shown in symbol list
- ▶ Use *Search* function to look backwards for I-cache burst read

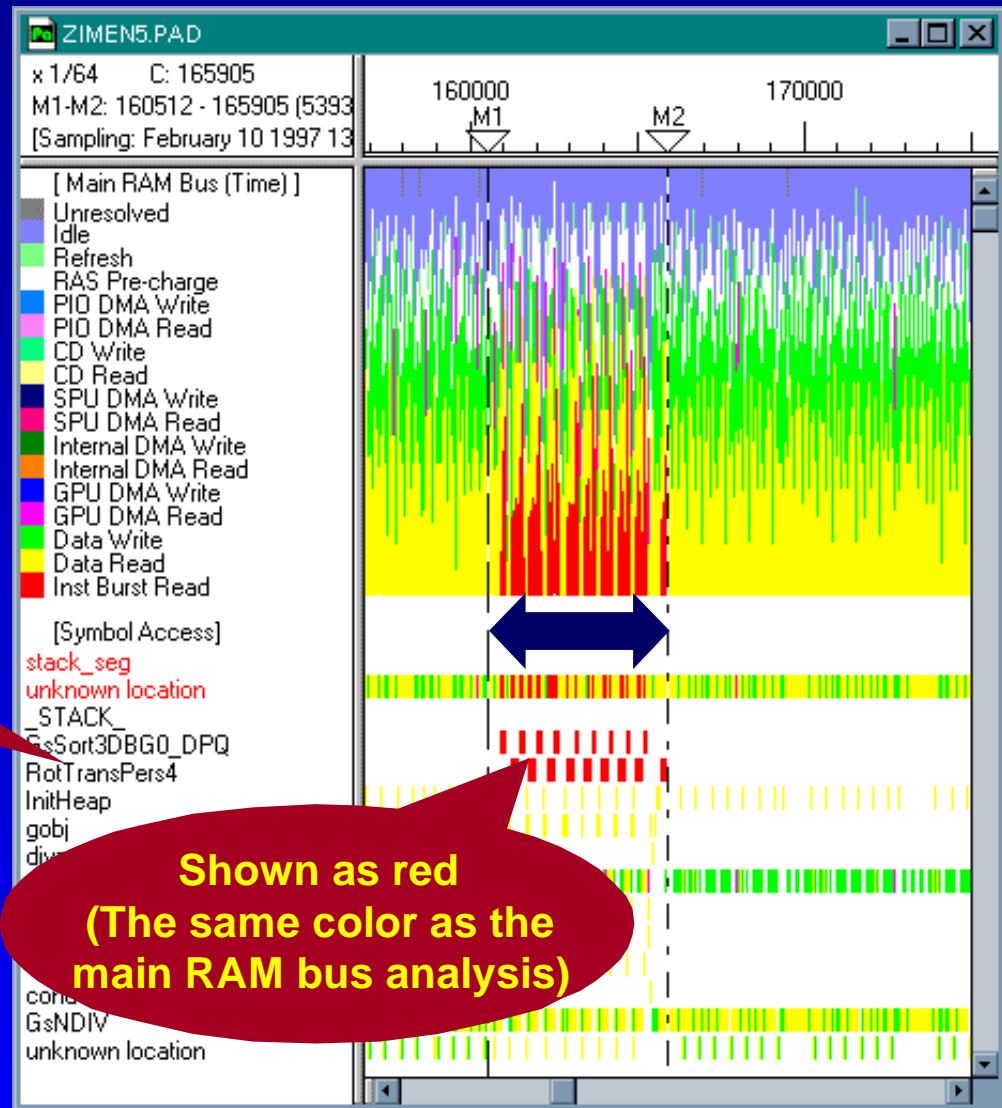
The screenshot displays the ZIMEN5.PAD memory analyzer interface. On the left, a legend lists various bus activities such as Unresolved, Idle, Refresh, RAS Pre-charge, PIO DMA Write, PIO DMA Read, CD Write, CD Read, SPU DMA Write, SPU DMA Read, Internal DMA Write, Internal DMA Read, GPU DMA Write, GPU DMA Read, Data Write, Data Read, and Inst Burst Read. The main window shows a colorful bus activity graph with a vertical cursor at address 160000. A red speech bubble points to the graph with the text "Which function is running?". On the right, the "Search Window - ZIMEN5.PAD" is open, showing search parameters for the "Main RAM Bus" with the mode set to "Inst Burst Read". A red speech bubble points to this dropdown menu with the text "Choose Inst Burst Read". Below the search parameters, there are buttons for "Previous Search", "Next Search", "Stop", and "Close". A red speech bubble points to the "Next Search" button with the text "Search backward".

# Identifying Functions Running on Cache

- ▶ Enclose red portion with markers, and extract function names which caused I-cache miss.

Function names

Shown as red  
(The same color as the  
main RAM bus analysis)





# Efficient Analysis Using Search

## ► Various search methods

- Main RAM Bus
  - by transaction type, by address or symbol
- Sub Bus
  - by transaction type, by address
- Video RAM Bus
  - by transaction type, by coordinates
- Others
  - by GPU packet type, by waveform event, by penalties

Search Window - ZIMEN5.PAD

Cursor Position: 165905

**Main RAM Bus**  
Main RAM Bus - Mode: Inst Burst Read  
Symbol List...  
Begin Address:   
Stop Address (Optional):   
Main RAM Bus - Data:

**Video RAM Bus**  
Video RAM Bus - Mode: NO SELECT  
 Range of Video RAM  
X-Range from:  0 to  0  
Y-Range from:  0 to  0

**GPU Packets, Waveform**  
GPU Packets: NO SELECT  
Waveform: NO SELECT

**Sub Bus**  
Sub Bus - Mode: NO SELECT  
Begin Address:   
Stop Address (Optional):

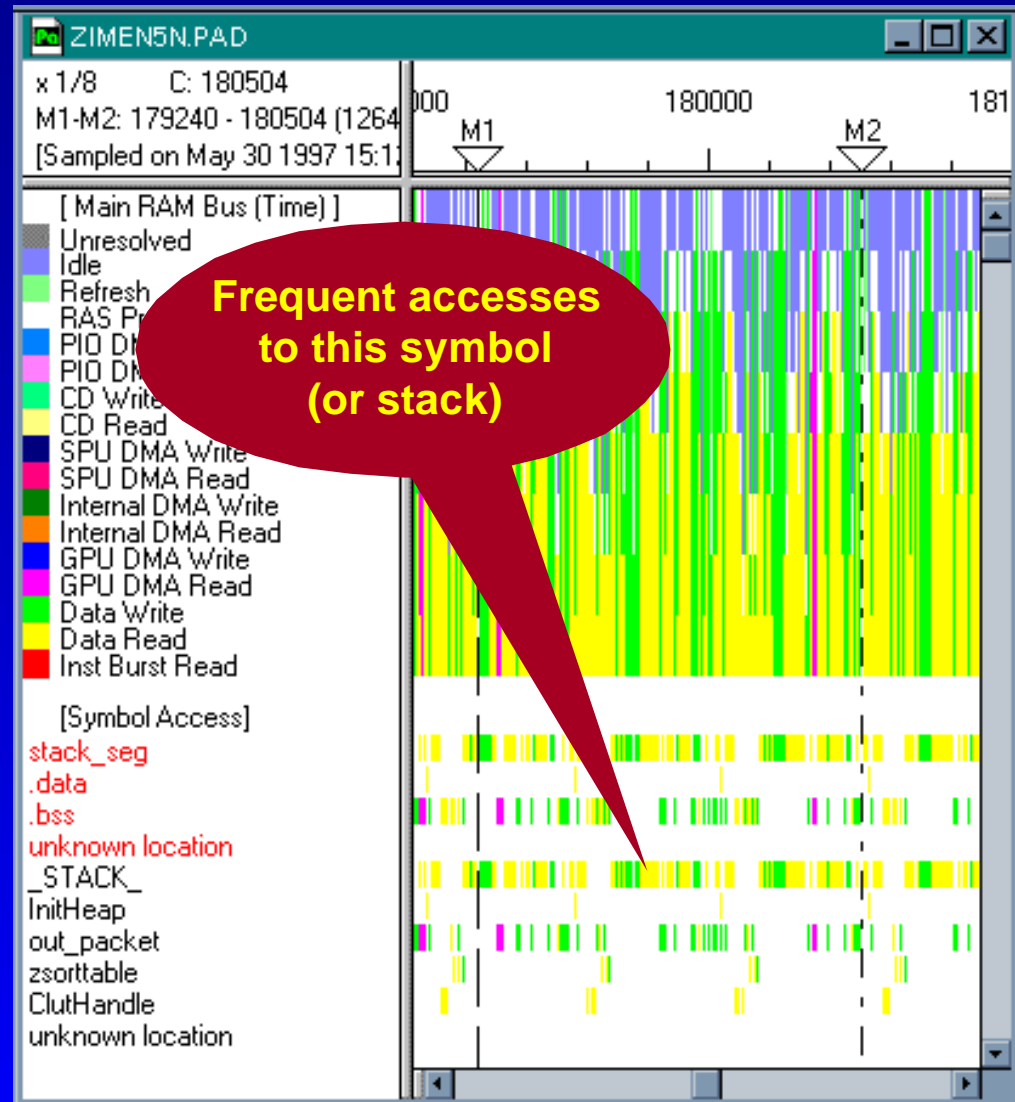
**Penalty**  
Read/Write: NO SELECT  
Write Buffer: NO SELECT  
Polygon: NO SELECT

Method of search:  AND  OR

Buttons: Previous Search, Next Search, Clear, Stop, Close

# Inspecting Read/Write Accesses

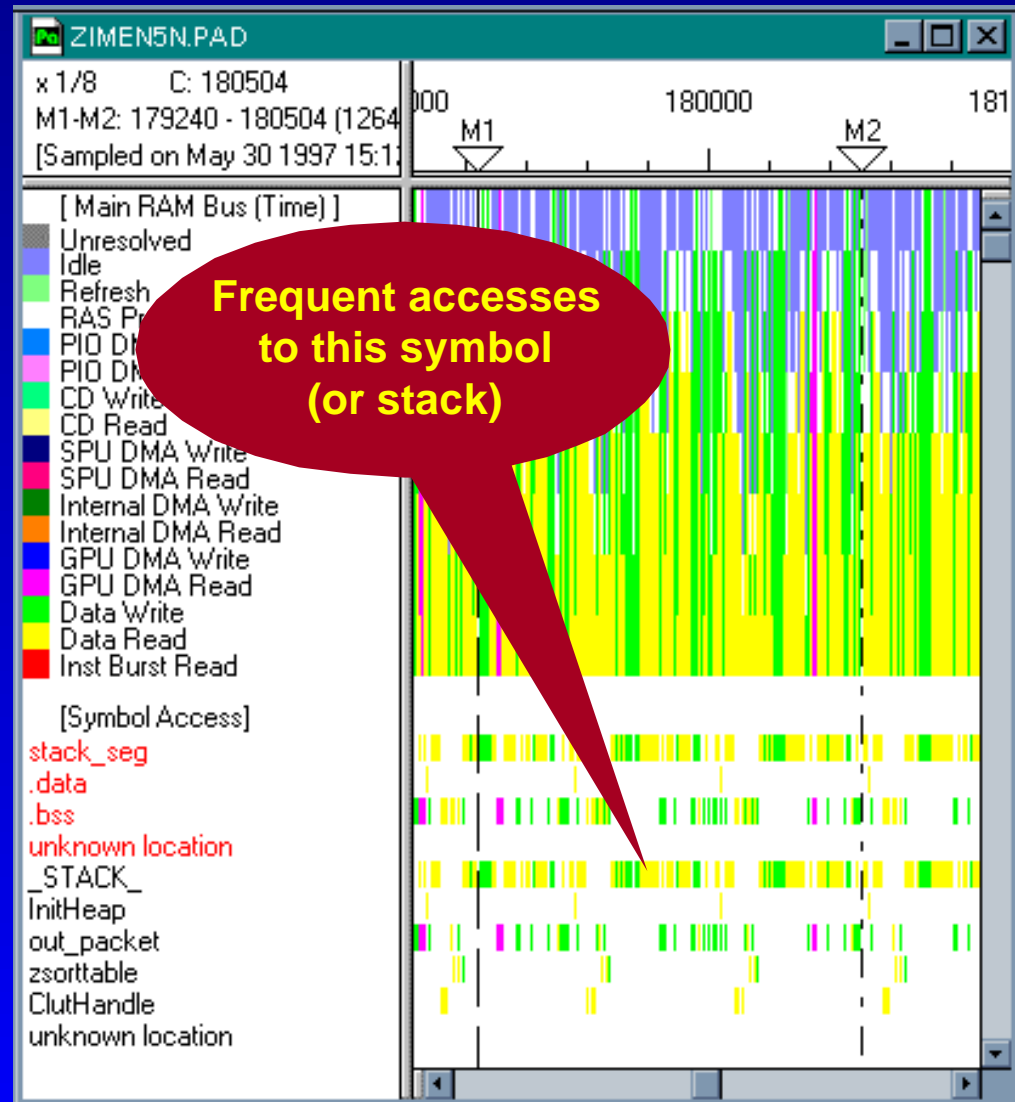
- ▶ Using the markers, enclose a portion of large red or green area, then display the only accessed symbols.
- ▶ CPU stalls for 5 cycles in each data read access to the main RAM!
  - Use coding styles such that result in few read accesses.
  - Tune stack accesses.
  - Check for duplicate reads of global data.
  - Use of joint-vertex model



# Inspecting Read/Write Accesses

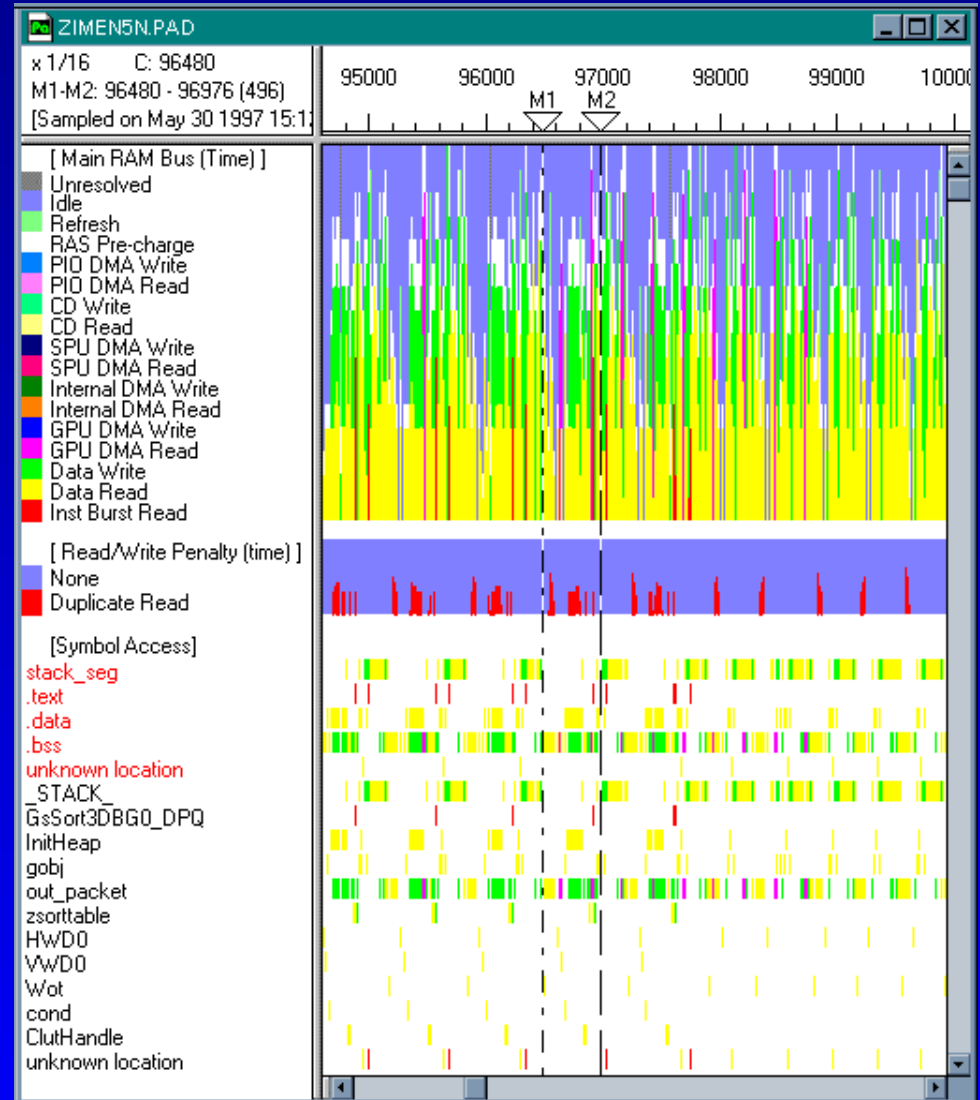
## ► Efficient use of the scratch pad RAM

- Use DMPSX together.
- Allocate a stack space on the scratch pad RAM
- Allocate heavily accessed work areas on the scratch pad RAM



# Duplicate Reads

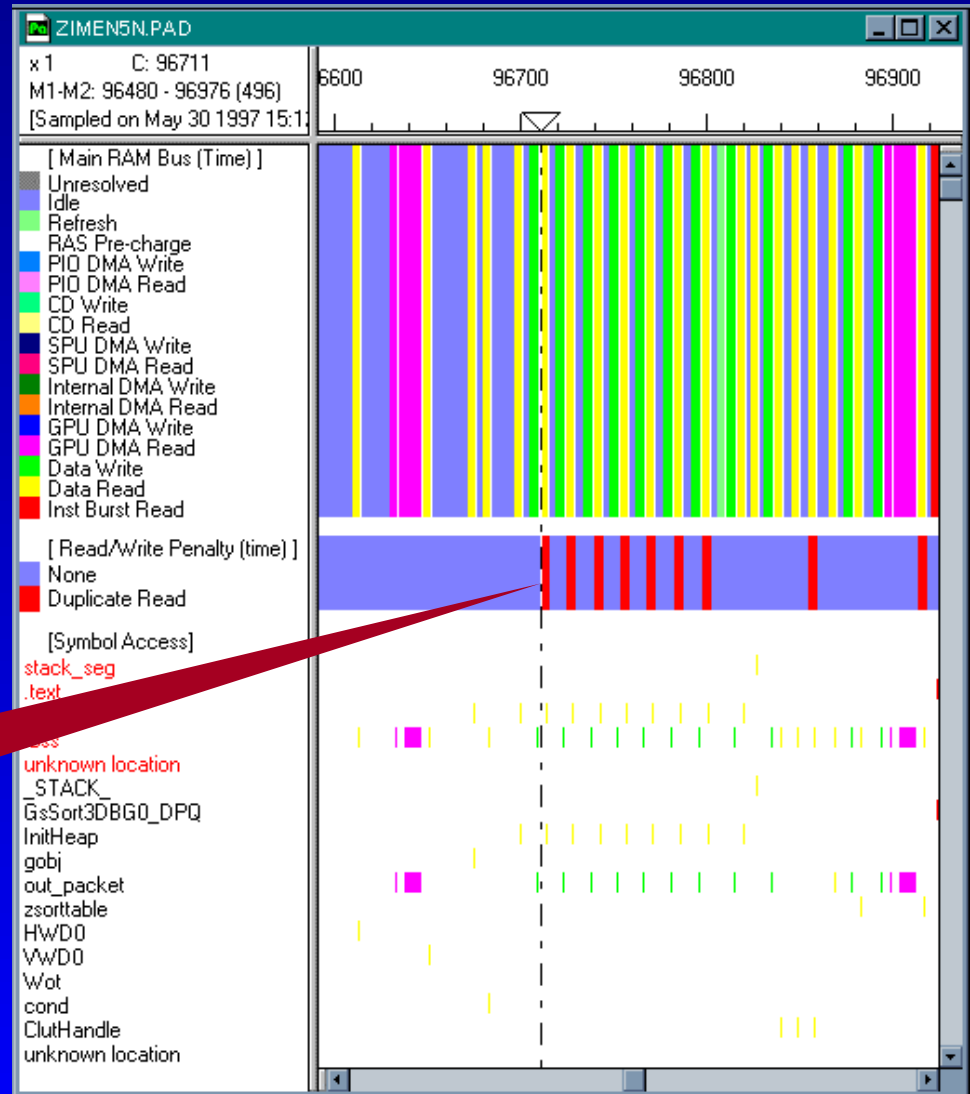
- ▶ Duplicate reads are shown as red stripes
  - Indicates CPU stall time
  - Concentrate on large areas
- ▶ Access interval threshold can be specified in *Options* dialog
  - Set lower values to highlight problem areas



# Duplicate Reads

- ▶ Using the *Search* command, search backwards for an access to same address

**Double click the red stripe and check the address**

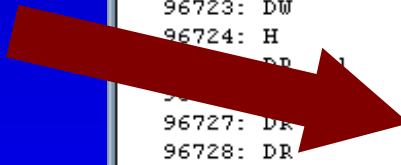
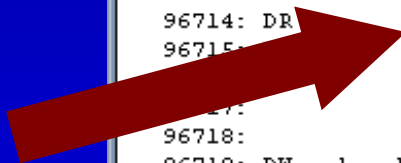


# Duplicate Reads

```
DUMP ZIMEN5N.PAD
Prev Next Go to Marker Page: 968
P MR N N M S O D MMM
O B/ W B A Y F A WWW
S W O Y D M F T EEEE
S R T D B S A NNNN
T D E R O E 3210
A L T ****
T

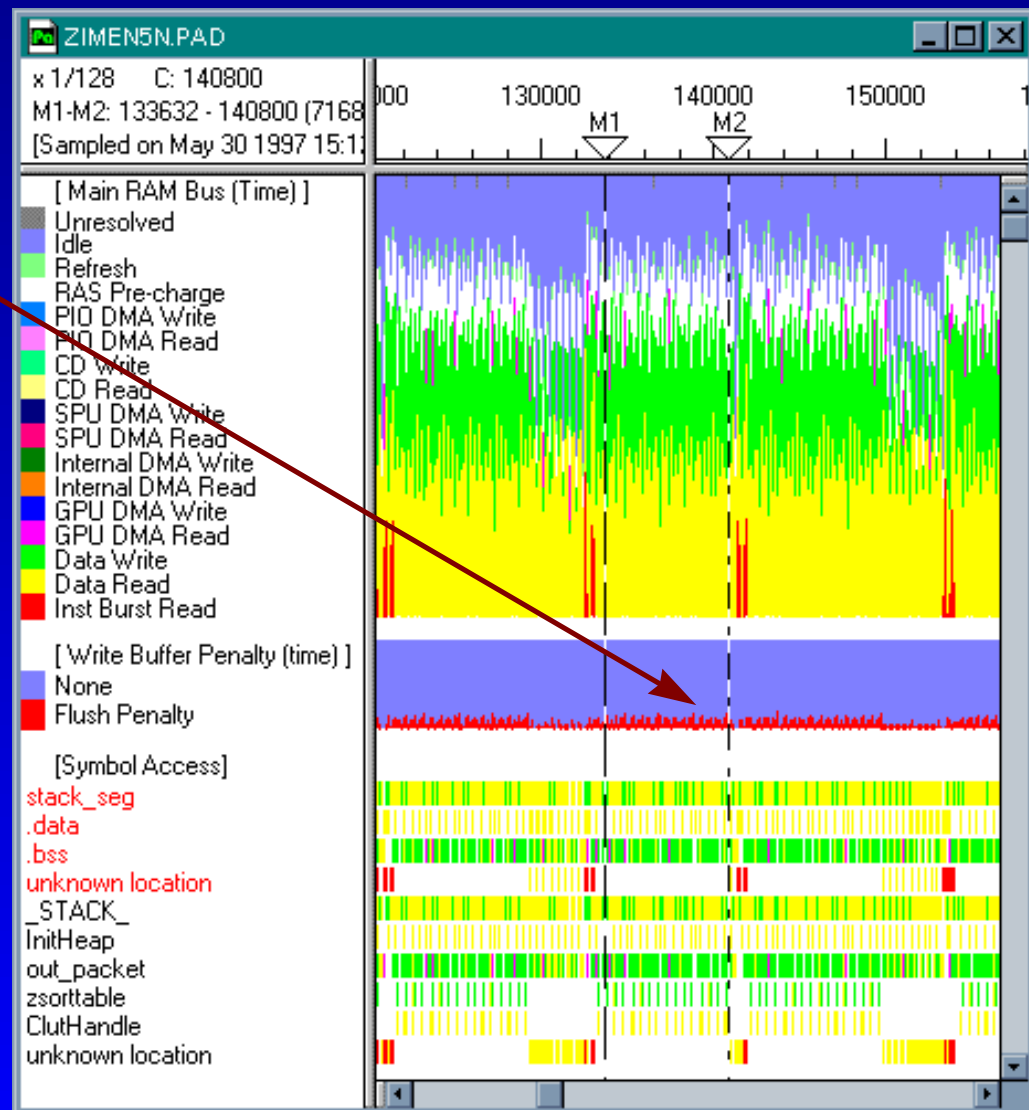
* 96711: DR 1 -----
96712: DR -----
96713: DR -----
96714: DR 8001C1F4 InitHeap+000030 20206060 -----
96715: -----
96716: -----
96717: -----
96718: -----
96719: DW 1 1 -----
96720: DW -----
96721: DW -----
96722: DW 00006000 --w-
96723: DW 8004EBDC out_packet+017C5C --w-
96724: H -----
96725: -----
96726: -----
96727: DR -----
96728: DR 8001C1F4 InitHeap+000030 20206060 -----
96729: H -----
96730: -----
96731: -----
96732: -----
96733: -----
96734: DW 1 1 -----
96735: DW -----
```

Duplicate reads



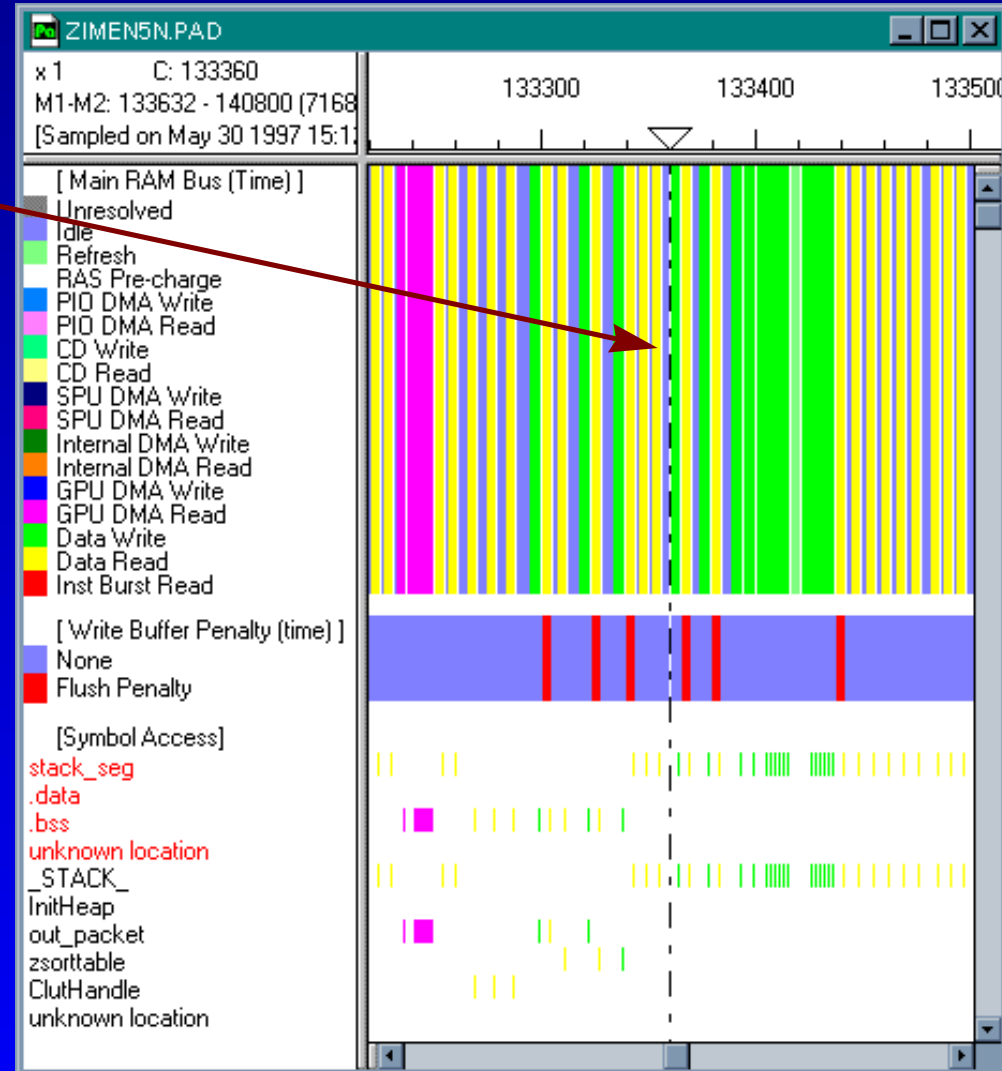
# Write Buffer Flush Penalty

- ▶ Write Buffer Penalties are shown as red stripes
- Indicates CPU stall time
- Concentrate on large areas



# Write Buffer Flush Penalty

- ▶ Particularly improve points where read and write accesses appear alternately.
  - Exchange the order of the read and write if possible.
  - Move other non-memory-access instructions to just after the store instruction.
- ▶ Note: All flush penalties don't necessarily mean the CPU is stalling.
  - In the main RAM bus analysis if an access takes place just after the write buffer has finished flushing, a penalty is displayed.
  - The CPU might be executing next non-memory-access instructions in between.





# Write Buffer Flush Penalty

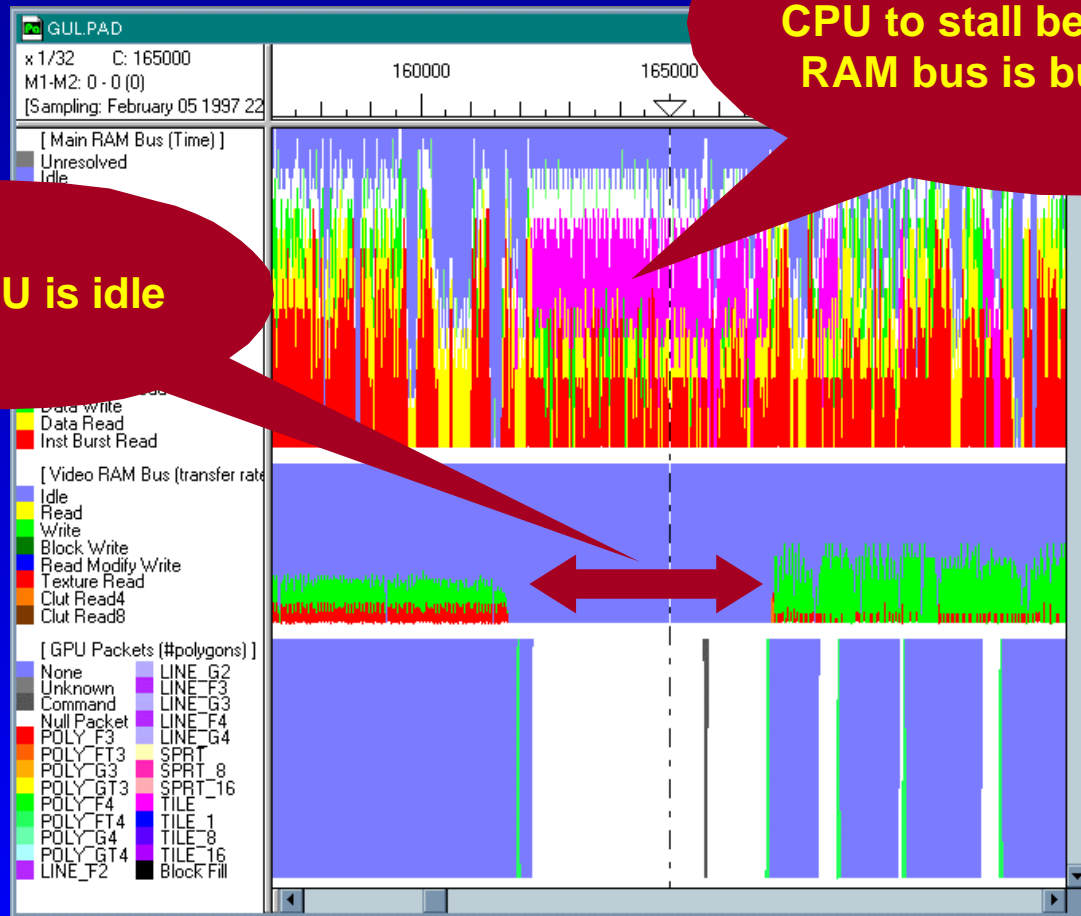
- ▶ There is a read access just after the write.
- ▶ Executing four successive store instructions results in most efficient processing

```
DUMP ZIMEN5N.PAD
Page: 1334

P MR N N M          S O D MMMM
O B/ W B A         Y F A WWWW
S W O Y D          M F T EEEE
S R T D            B S A NNNN
T D E R            O E 3210
A                  L T ****
T

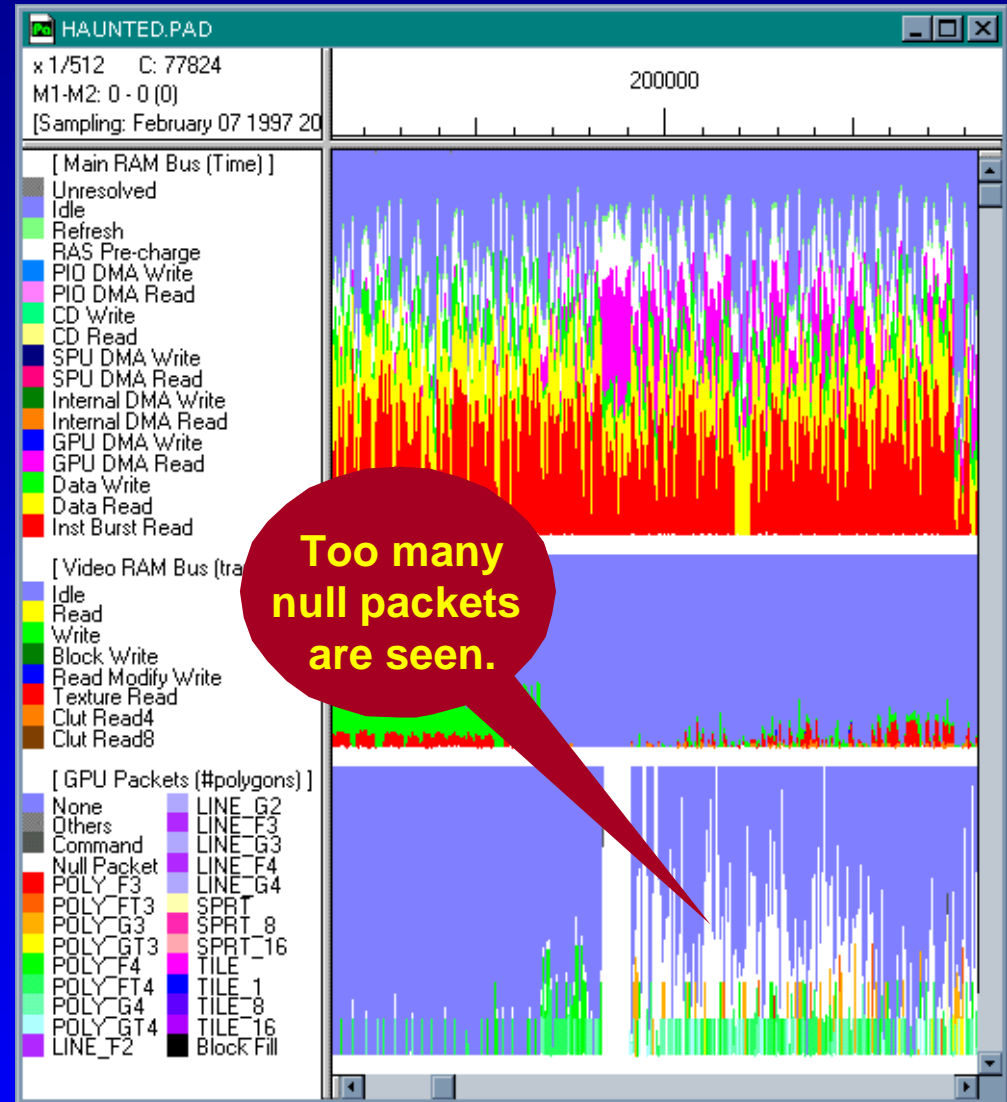
* 133360: DW 1 2
133361: DW
133362: DW
133363: DW
133364: DW          807FFDB0  _STACK_ 00001100 --ww
133365: H
133366: DR 1
133367: DR
133368: DR
133369: DR          807FFDC0  _STACK_ 21001000 ----
133370: H
133371:
133372:
133373:
133374: DW 1 2
133375: DW
133376: DW
133377: DW
133378: DW          807FFDB8  _STACK_ 00001200 --ww
133379: H
133380: DR 1
133381: DR
133382: DR
133383: DR          807FFDC8  _STACK_ 21001100 ----
133384: H
```

# Detecting NULL GPU Packets

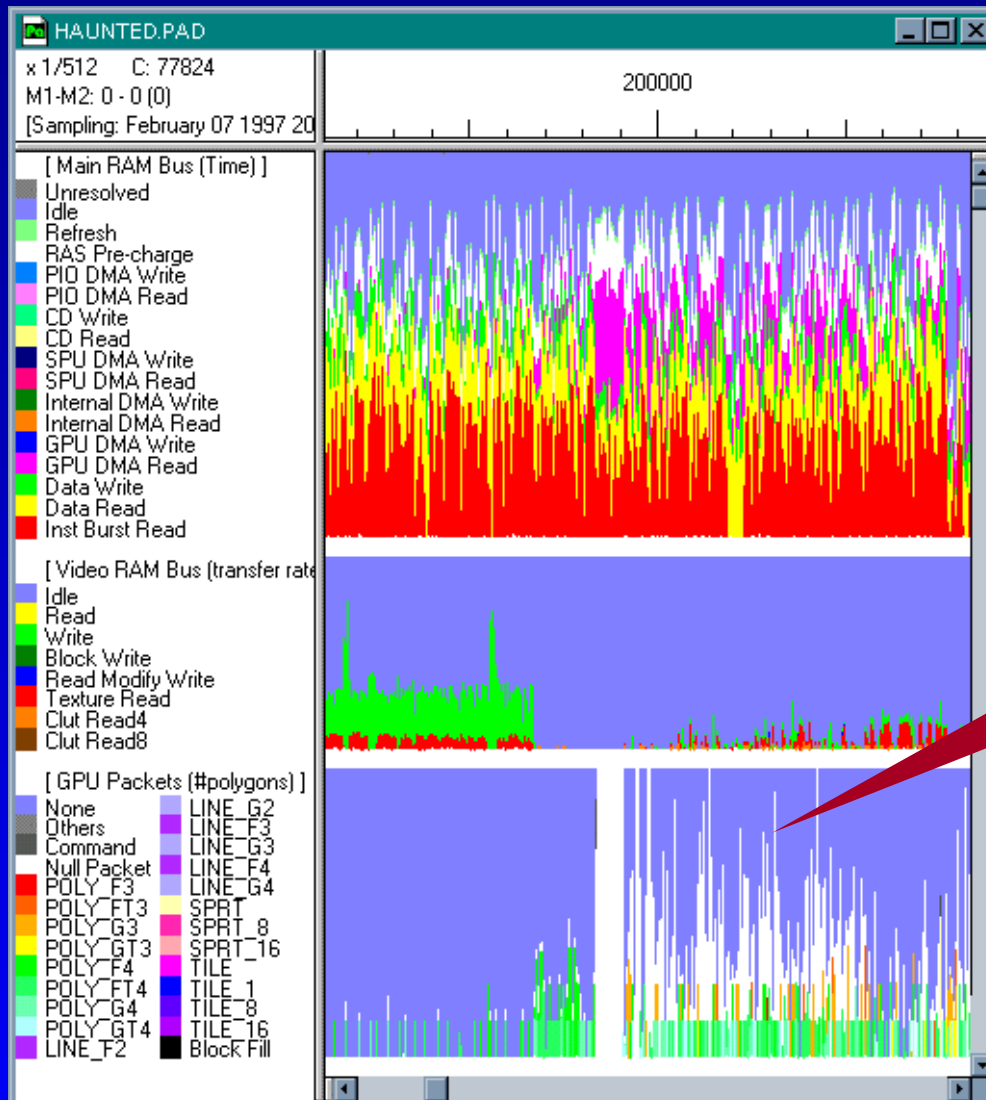


# Detecting NULL GPU Packets

- ▶ Too many NULL packets hurts performance
  - Slows down the CPU because the main RAM bus is busy
  - Wastes the GPU's time



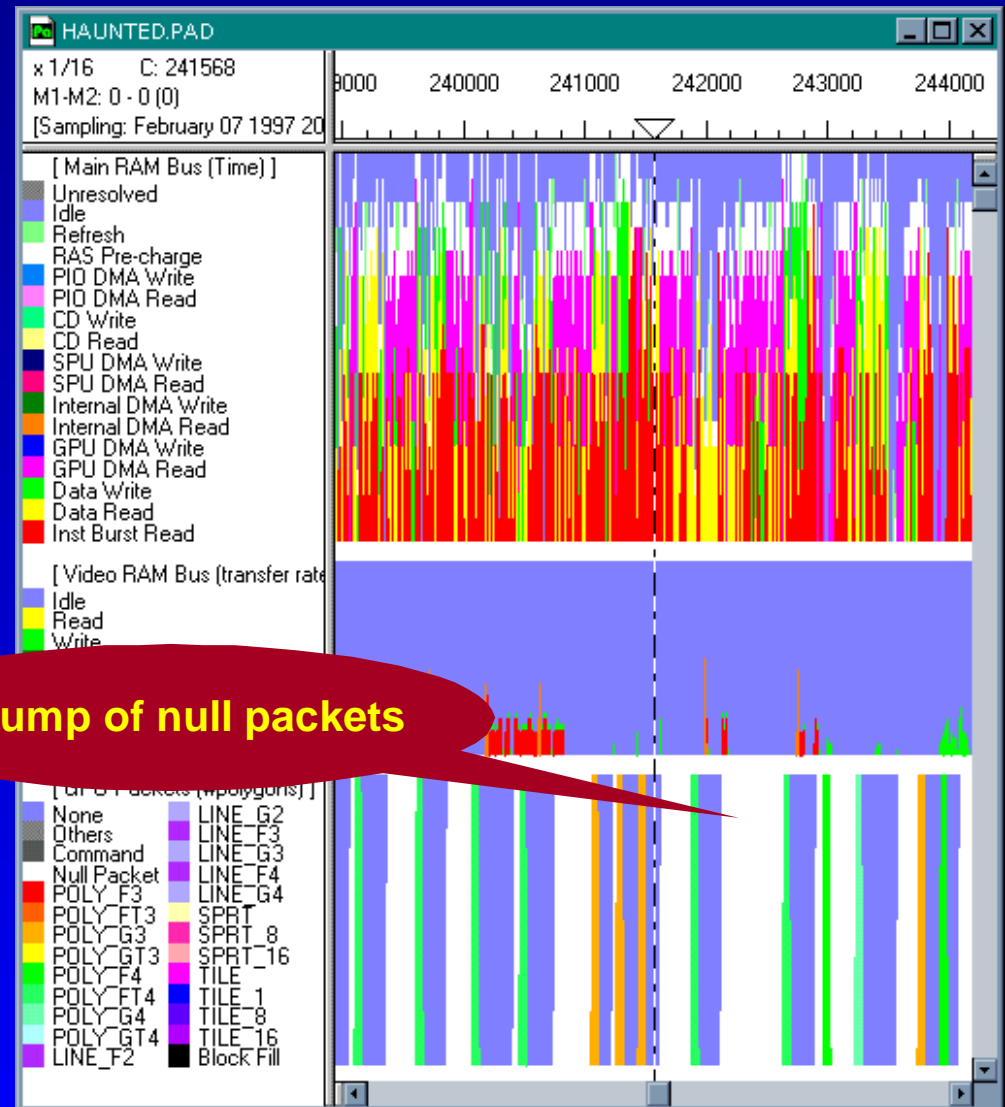
# Checking Ordering Table Resolution



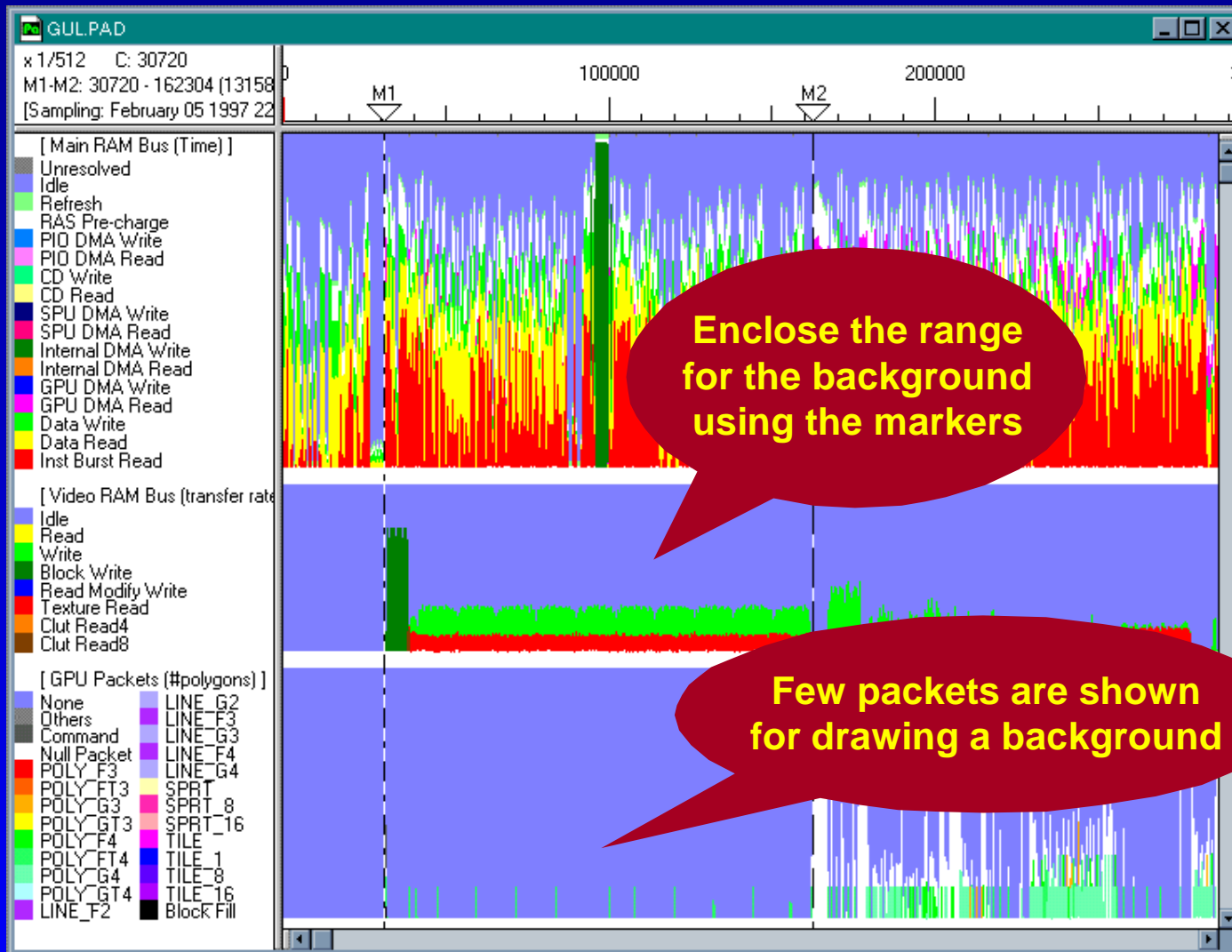
Many null packets are seen.

# Checking Ordering Table Resolution

- ▶ Processing is slowed if ordering table resolution is too high

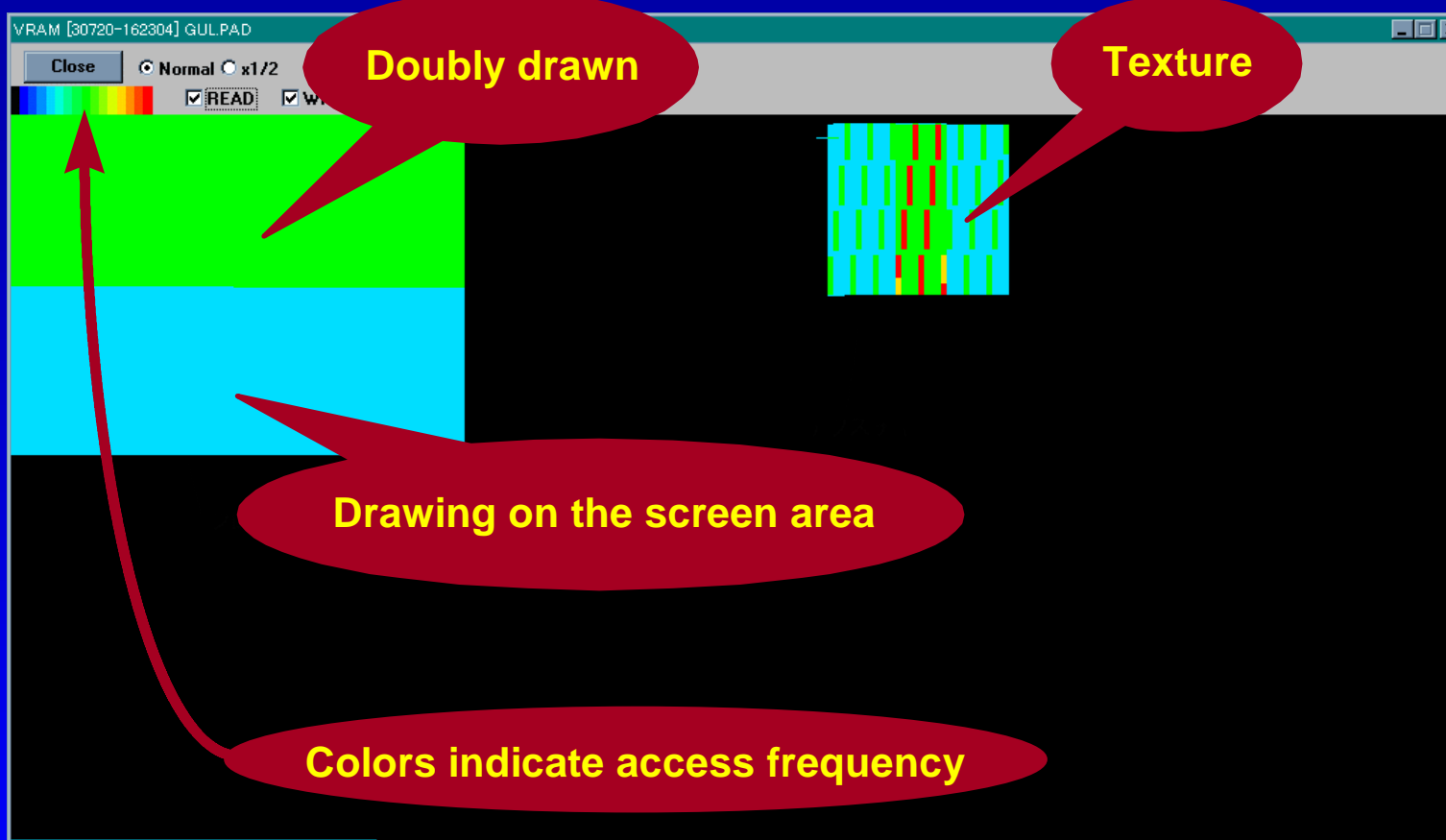


# Checking Background Drawing



# Checking Background Drawing

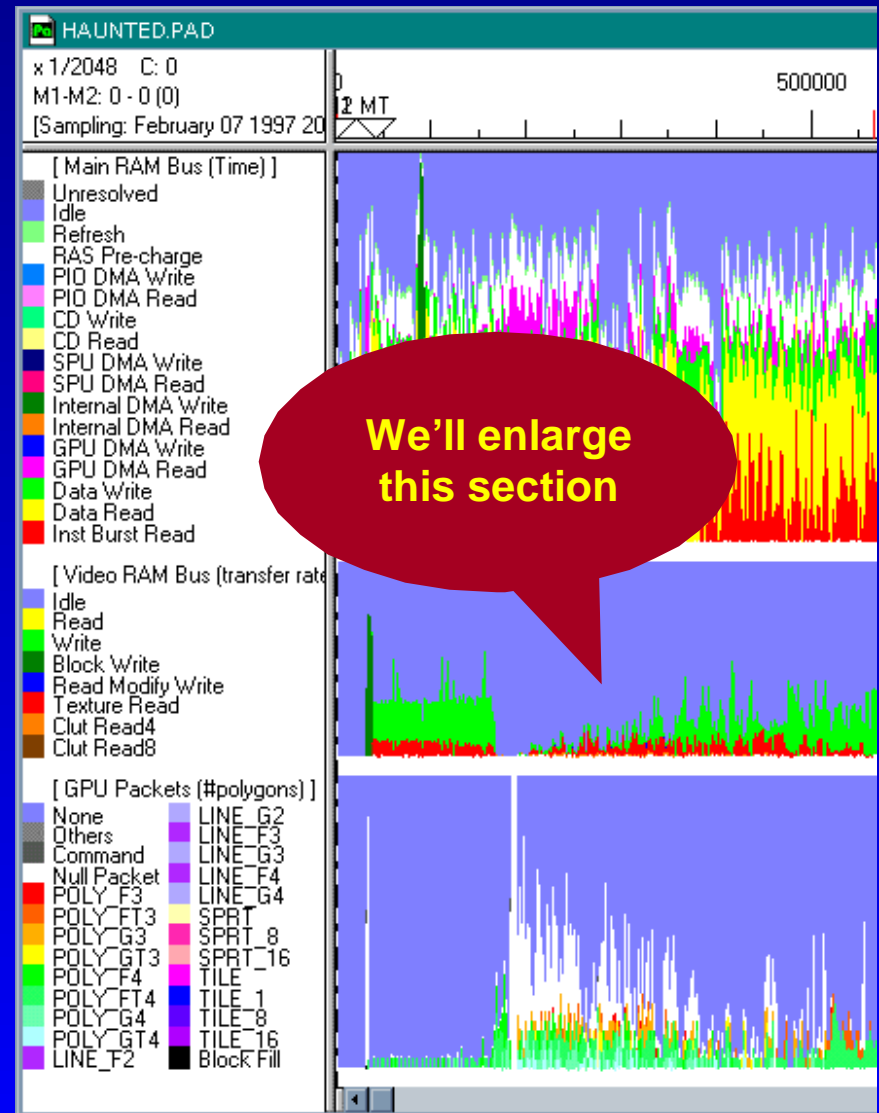
- ▶ Detect doubly-drawn areas using the video RAM viewer



# Inefficient Drawing

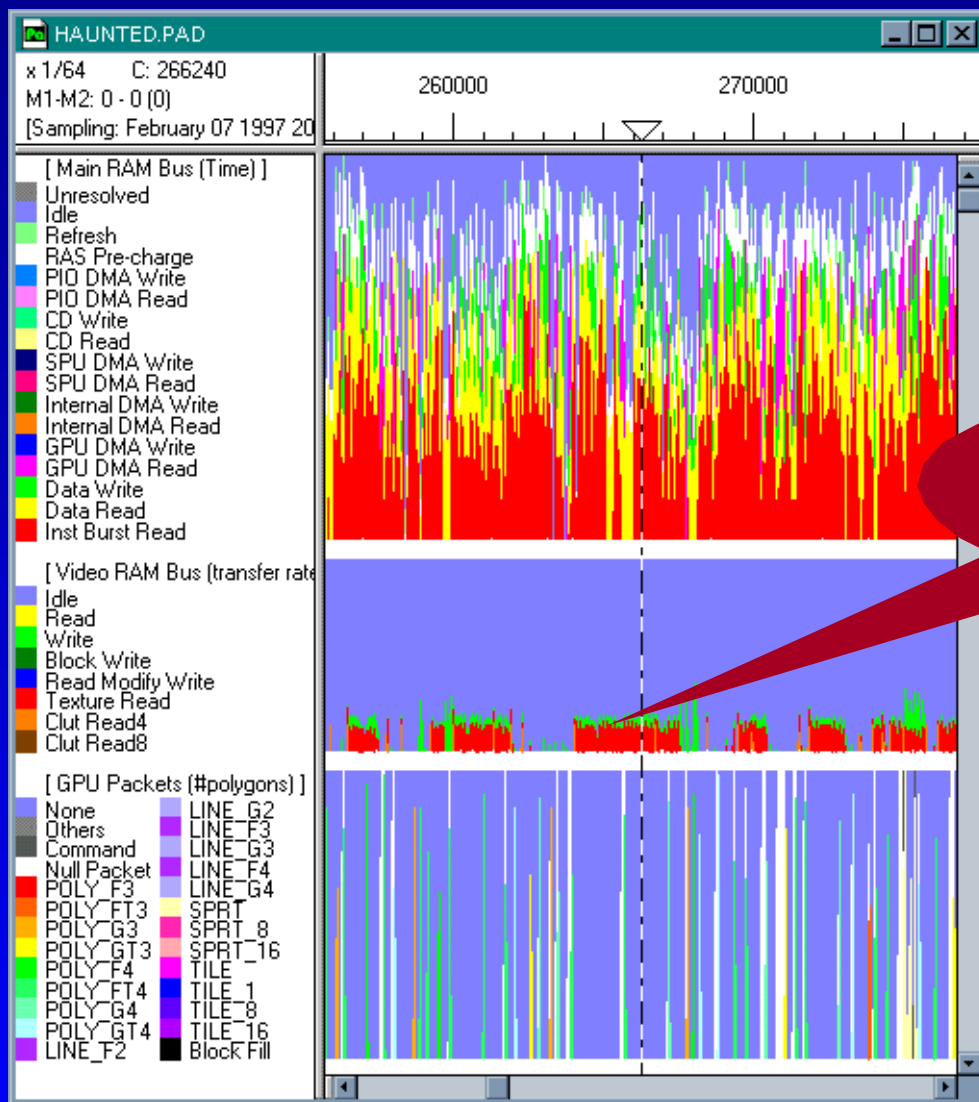
## ► Why are green patterns low?

- Most of the cause
  - Texture cache miss
  - Preprocessing bottleneck
    - Small polygons
- Other causes
  - Ordering Table too large
  - Clipping
  - Transparent colors
  - Vertically long polygons
    - Page fault penalties on the video RAM





# Inefficient Drawing



Many texture cache misses

# Characteristics of the Texture Cache

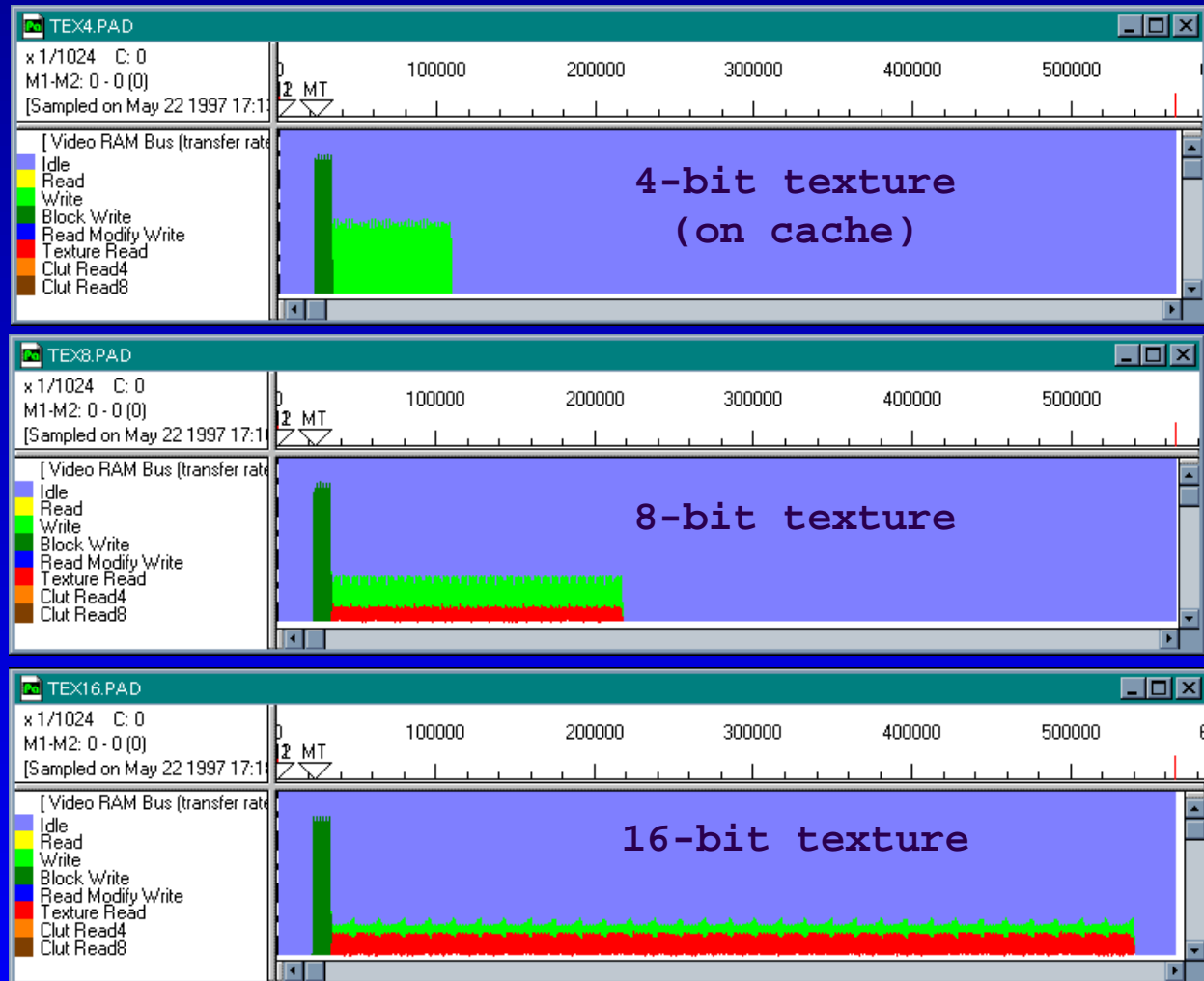
- ▶ 2KB direct-mapped cache with 6 bits in each tag
  - Cache size in pixels, the number of cache lines
    - 4-bit texture: 64x64 texels, 64x4 lines
    - 8-bit texture: 64x32 texels, 64x4 lines
    - 16-bit texture: 32x32 texels, 32x8 lines
- ▶ Line size: 64 bits
  - The number of texels (texture-pixels) loaded one at a time horizontally in the burst mode when texture cache miss occurs
    - 4-bit texture: 16 texels
    - 8-bit texture: 8 texels
    - 16-bit texture: 4 texels

# *Characteristics of the Texture Cache*

- ▶ Texture page size: 256 x 256 texels
  - The texture cache is flushed every time when a new texture page is accessed.

# Comparing Texture Modes

- ▶ Performing 16 flat-texture mappings with 64x64 texture size.



# Comparing Texture Modes

- ▶ A very low transfer rate results when texture reads occur.



# *Techniques for Fast Texture Mapping*

- ▶ Divide the ordering table if possible
  - Use a different ordering table for polygons which use the same texture.
    - The ground and floor usually use repeated texture. The ordering table is easily divided for such objects, too.
  - Divide the ordering table based on texture pages.
    - The texture cache is flushed every time when a new texture page is accessed.

# *Techniques for Fast Texture Mapping*

- ▶ If texture is mapped reducing its size, 4-bit texture mode also becomes slow.
  - Use mip-mapping
  - If mapping with reducing texture size is inevitable, using 8 or 16-bit texture mode do not slow down drawing speed a lot.

# *Techniques for Fast Texture Mapping*

- ▶ Do not place a large texture data rotating at 90 degrees if it doesn't fit in the texture cache size.
  - Divide polygons into smaller ones.
  - Prepare polygons for each rotation.



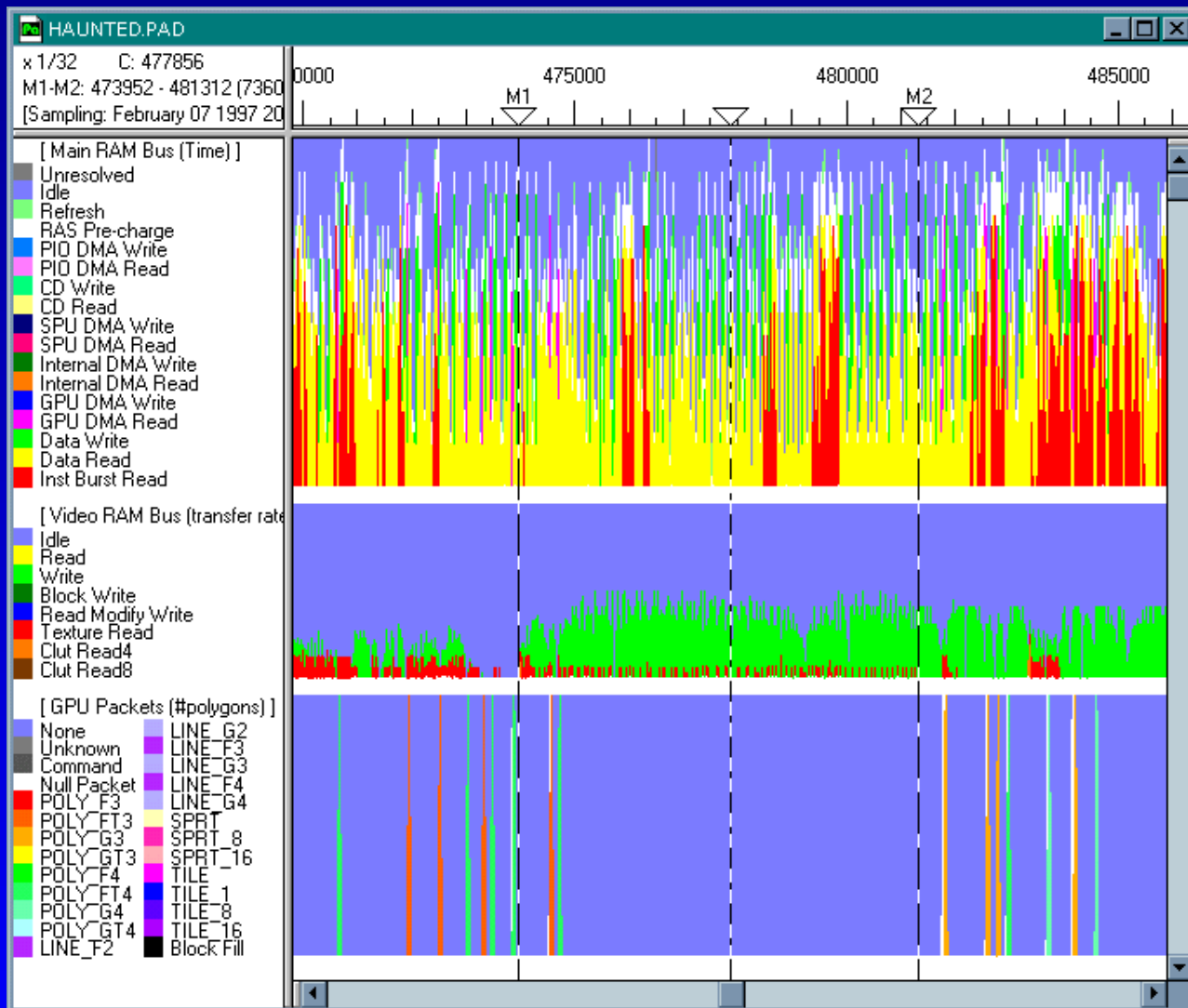
# *Inspecting Texture Cache Miss Causes*

- ▶ On the video RAM bus analysis, use the markers to enclose a portion to inspect.
- ▶ Use the video RAM viewer to inspect the cause, then check the following.
  - Ratio of texture data read to polygon pixels written
    - Reducing/expanding in texture mapping.
    - Enlarged textures drawn efficiently, but may look pixelated
    - Reduced textures not drawn efficiently
  - Ratio of texture data read to polygon pixels written, and the access frequency (shown as colors)
    - Cache-hit ratio

# *Inspecting Texture Cache Miss Causes*

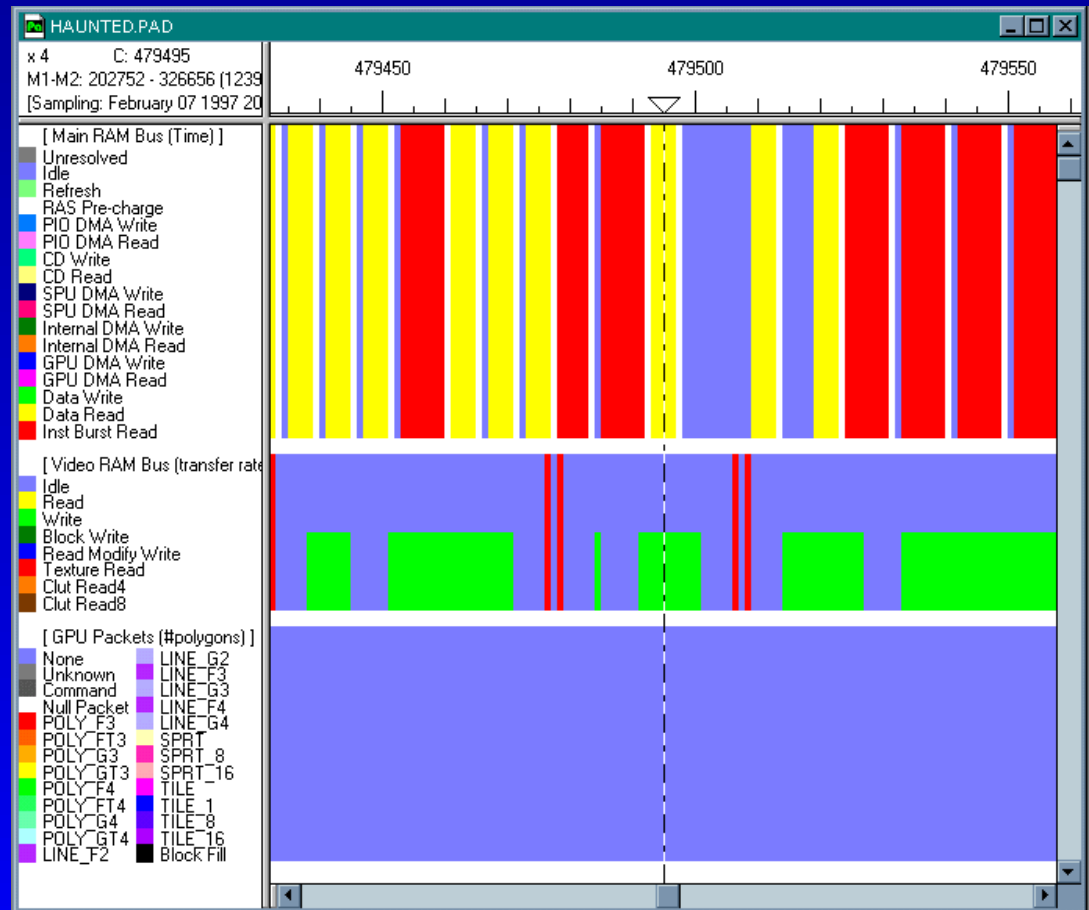
- ▶ Use the video RAM viewer to inspect the cause, then check the following.
  - Accessed texture locations, Texture pages
    - Frequency of cache flushing
  - Roughness of horizontal lines in texture area
    - Reducing factor
  - The shape of an accessed texture area pattern, reading direction
    - Shape of texture data, rotated polygons

# Texture Cache Missing not a Problem



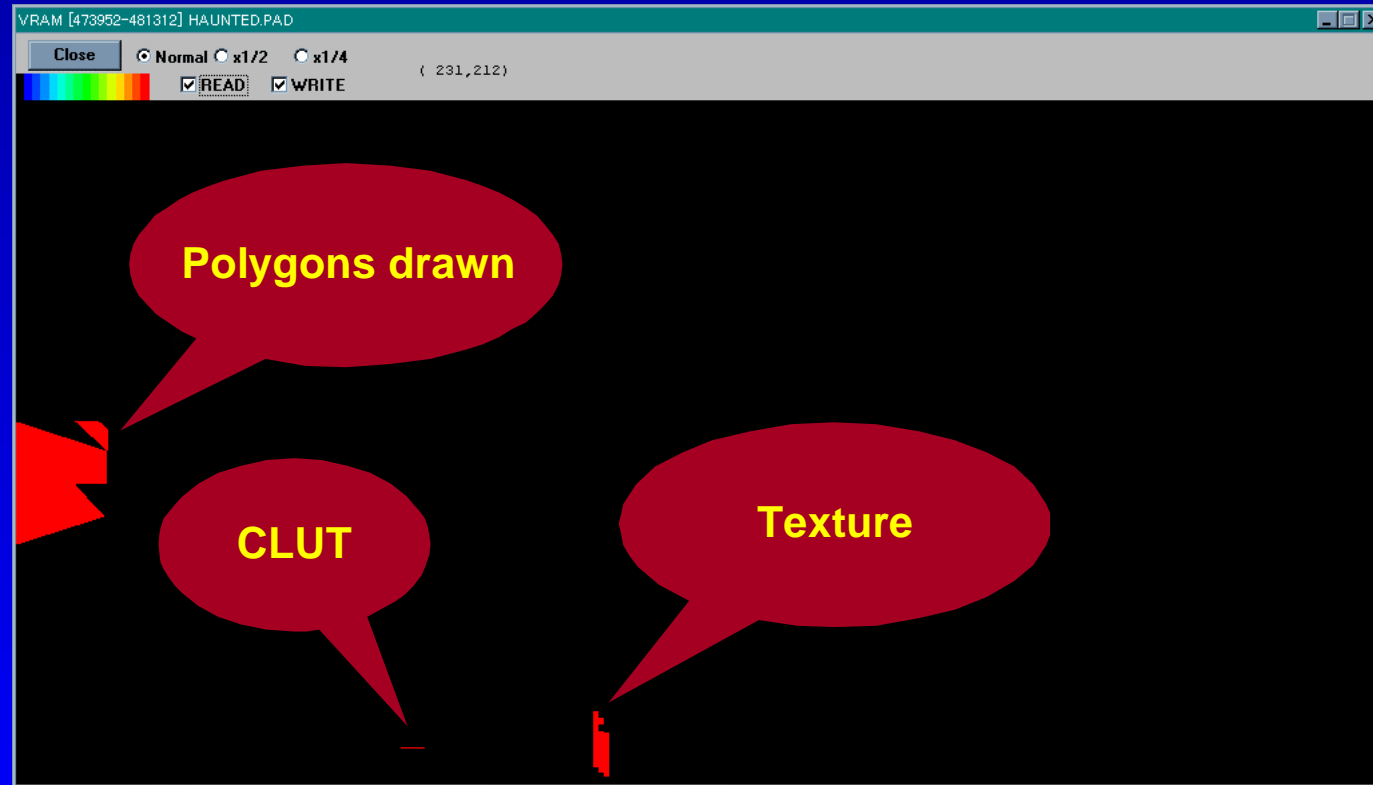
# Texture Cache Missing not a Problem

- ▶ No interruption in drawing results in high drawing speed

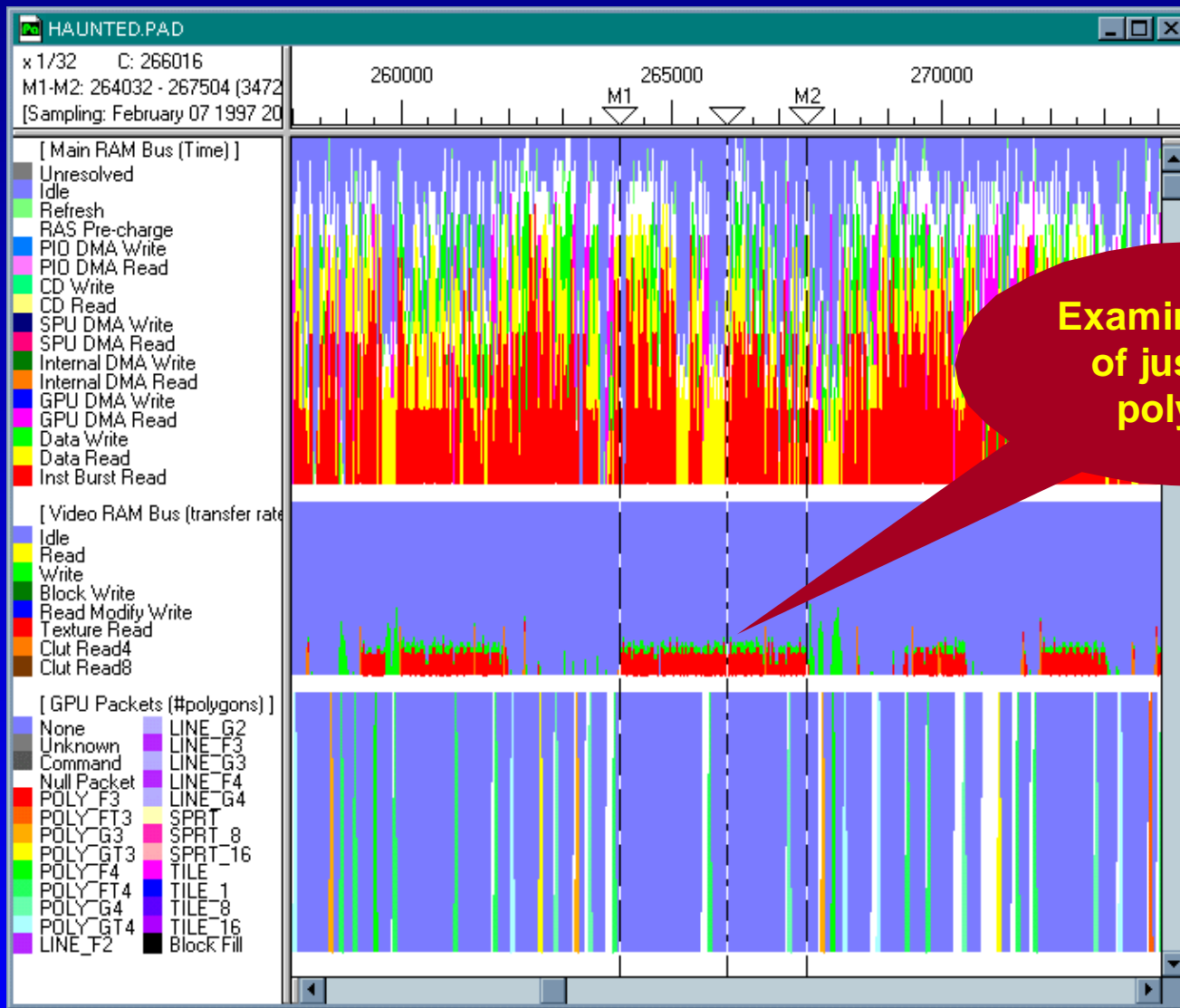


# *Texture Cache Missing not a Problem*

- ▶ Viewing in the video RAM display
  - Efficient drawing because of expanded texture

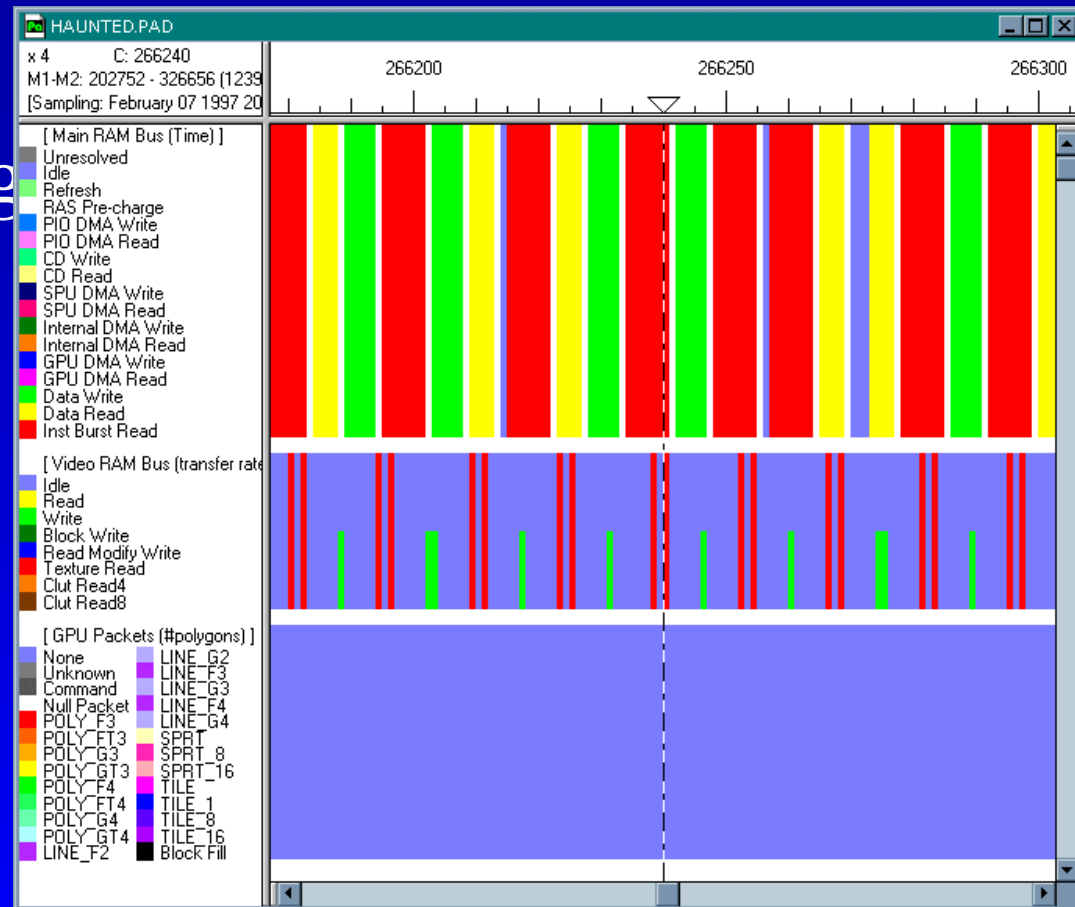


# Frequent Texture Cache Misses



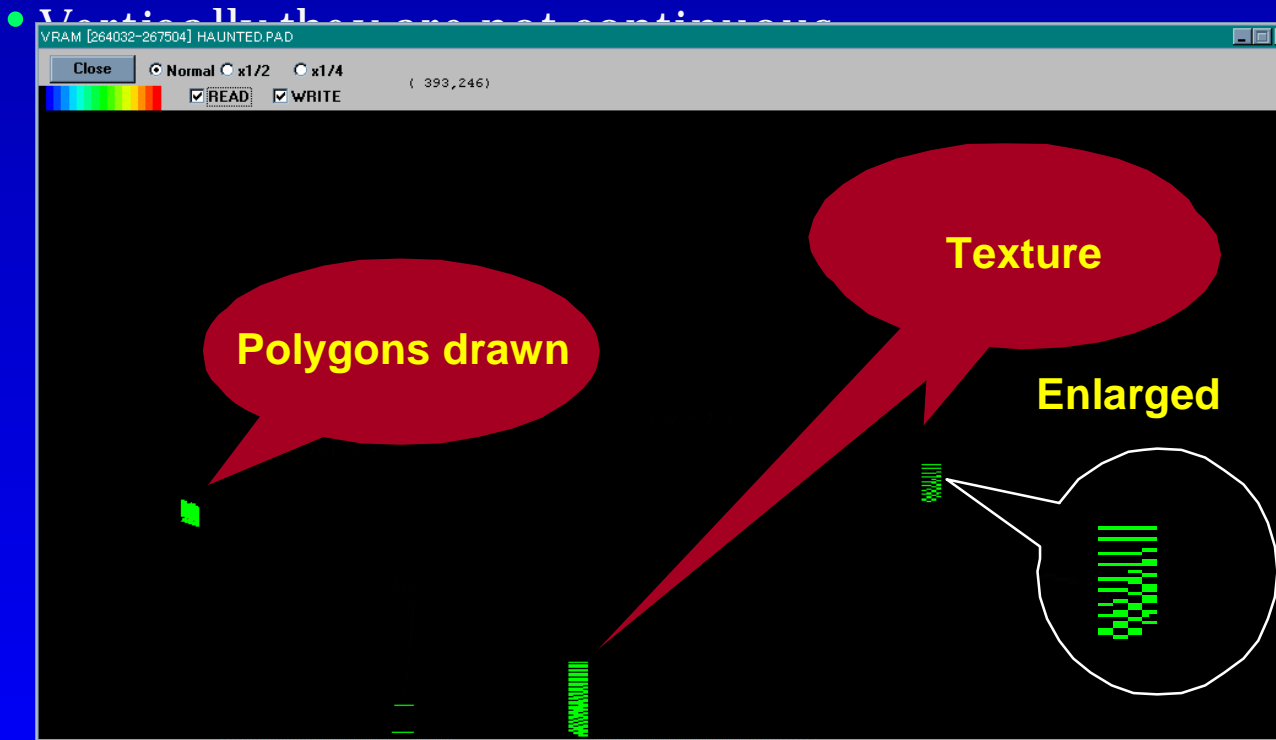
# Frequent Texture Cache Misses

- ▶ Cache misses mean many interruptions in drawing, resulting in low drawing speed



# Frequent Texture Cache Misses

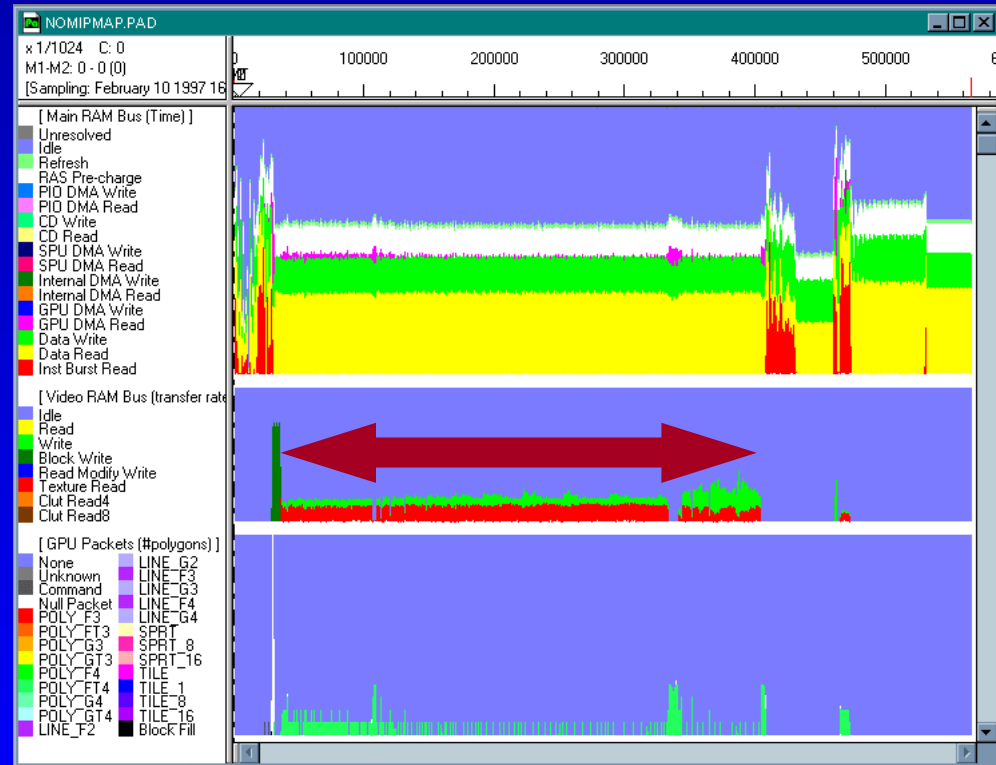
- ▶ Inspecting in the video RAM viewer
  - Inefficient drawing due to reduced magnification of texture data.
  - Read accesses appearing as horizontal stripes
  - Vertical textures are not continuous





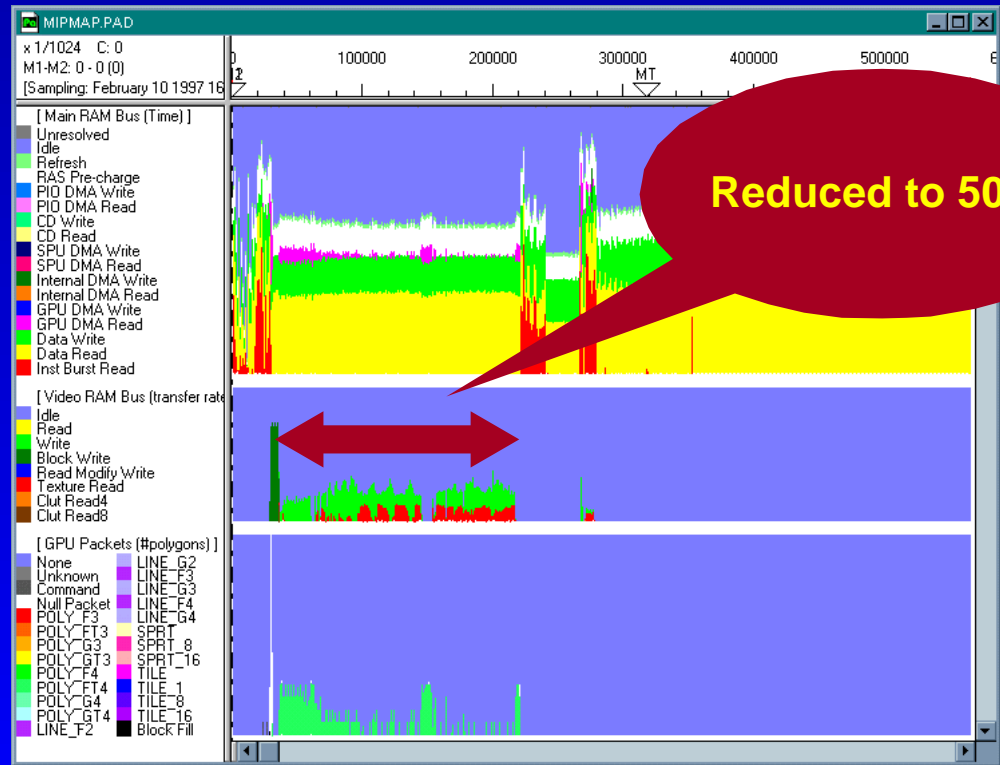
# Mip-Mapping -vs- not Mip-Mapping

- ▶ Even if 4-bit texture mode is used to increase effective texture cache size, reducing a texture when drawing a polygon results in unused texels being loaded on the cache, therefore the burst read is no longer effective.
  - For example, drawing a 20x20 polygon that uses a 64x64 texture

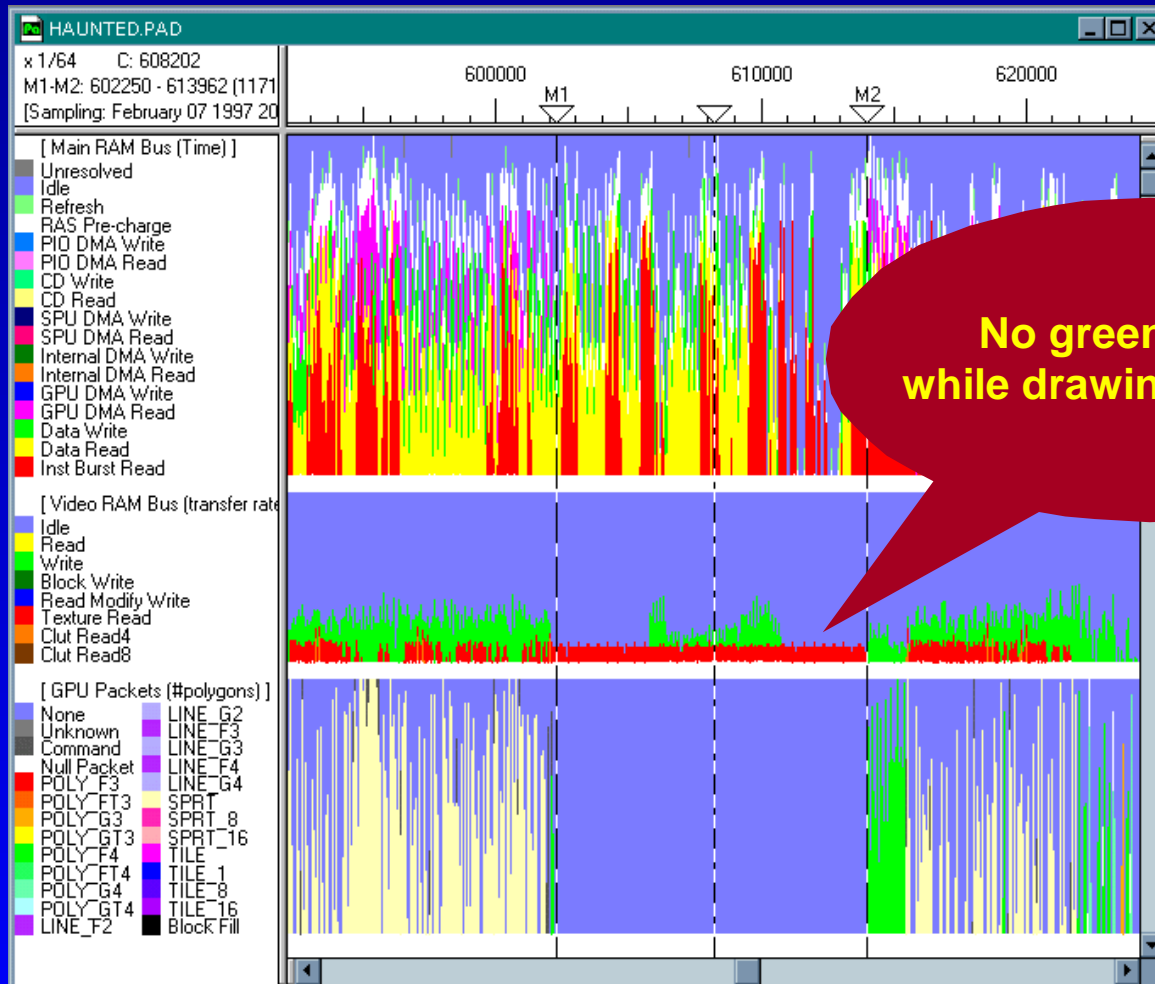


# Mip-Mapping -vs- not Mip-Mapping

- ▶ Mip-mapping can take advantage of the 4-bit texture mode avoiding shrunk texture
- ~~Mapping~~ Mapping solves an aliasing problem in texture mapping far-away polygons.
  - Reduces texture crawl and flicker.

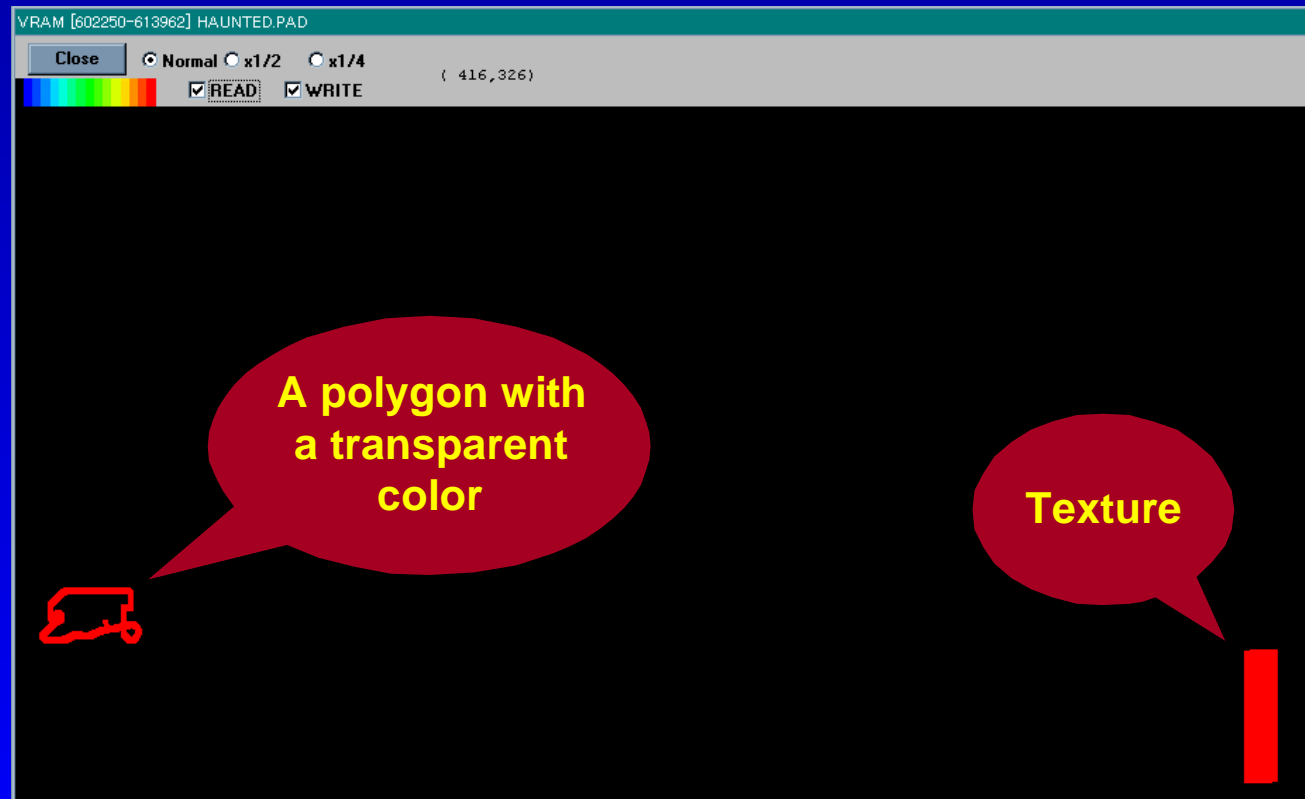


# Polygons with Transparent Areas



# *Polygons with Transparent Areas*

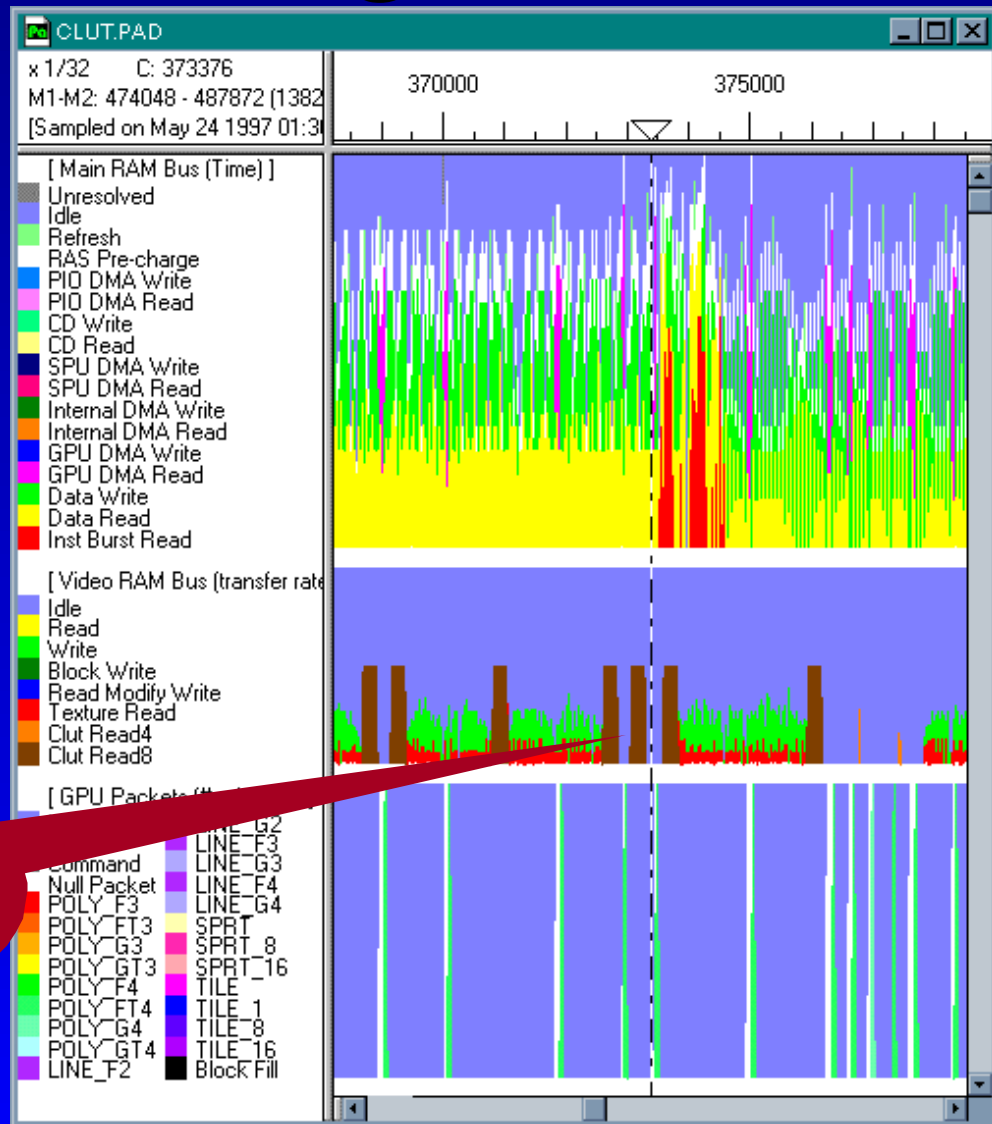
- ▶ A transparent color takes the same amount of drawing time as a non-transparent color.
- If transparent area is large, divide the polygon to increase the performance.
  - Don't draw the hole!



# Frequent CLUT Switching

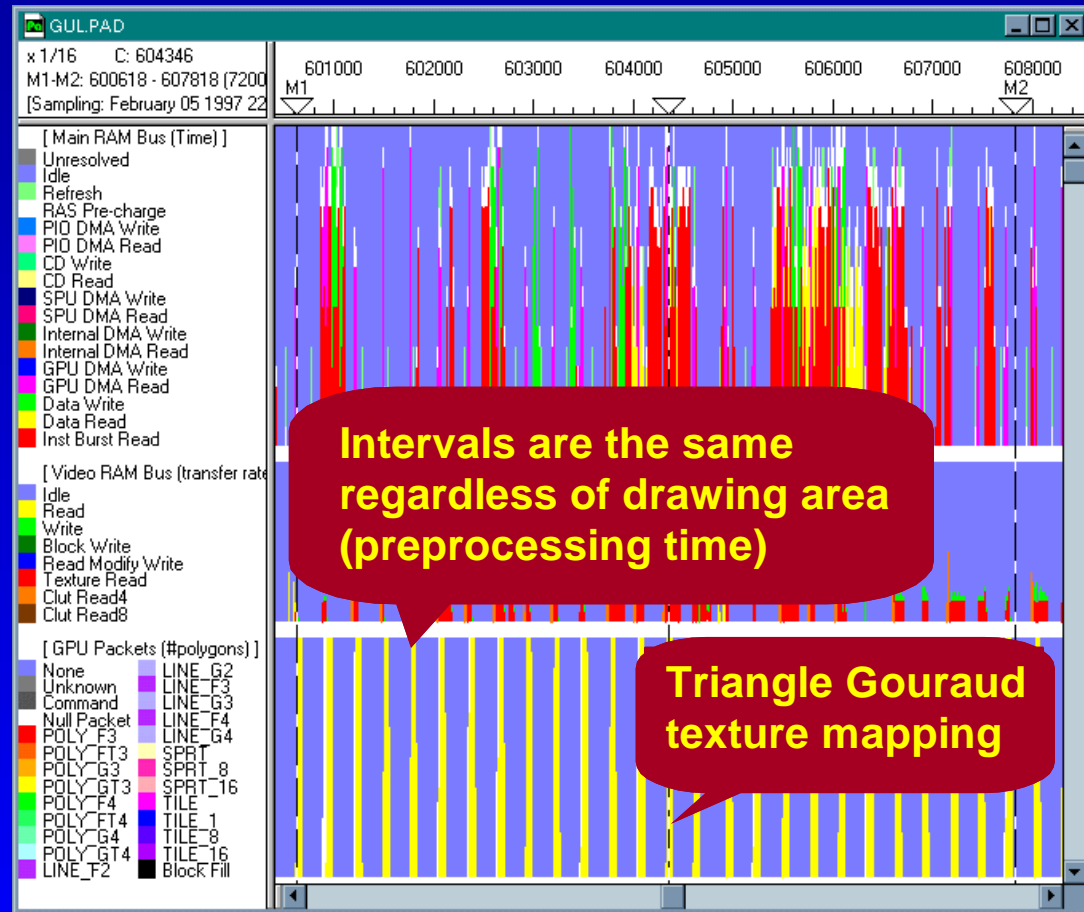
- ▶ Frequent CLUT switching slows down drawing speed.
  - Use a separate ordering table for each CLUT if possible.

**8-bit CLUT switching is taking place very often**



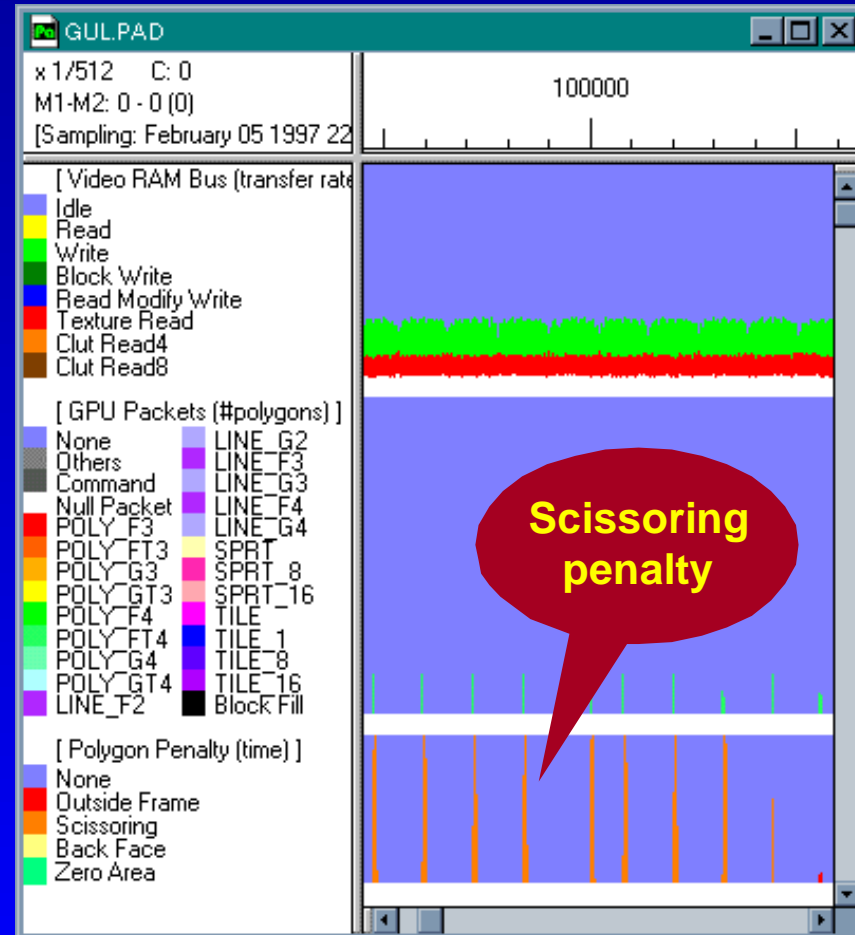
# GPU Preprocessing Bottlenecks

- ▶ Small polygons with reduced textures have long pre-processing times which decrease performance.
  - For example, POLY\_GT4
  - Investigate changing the polygon type to flat-shaded and/or non-textured if Z is large.



# Polygon Penalties

- ▶ Set the screen size and drawing offset values.
  - Set them in the option dialog box.
  - Inspect a point with many polygon penalties, then identify the polygon using the video memory viewer.
- ▶ If too many back-face polygon penalties are shown, toggle its setting in the option dialog box.
  - Some programs use both back and front face polygons
  - In the mesh data, the vertex order and the face of the polygon do not match, therefore the penalty is not shown correctly.



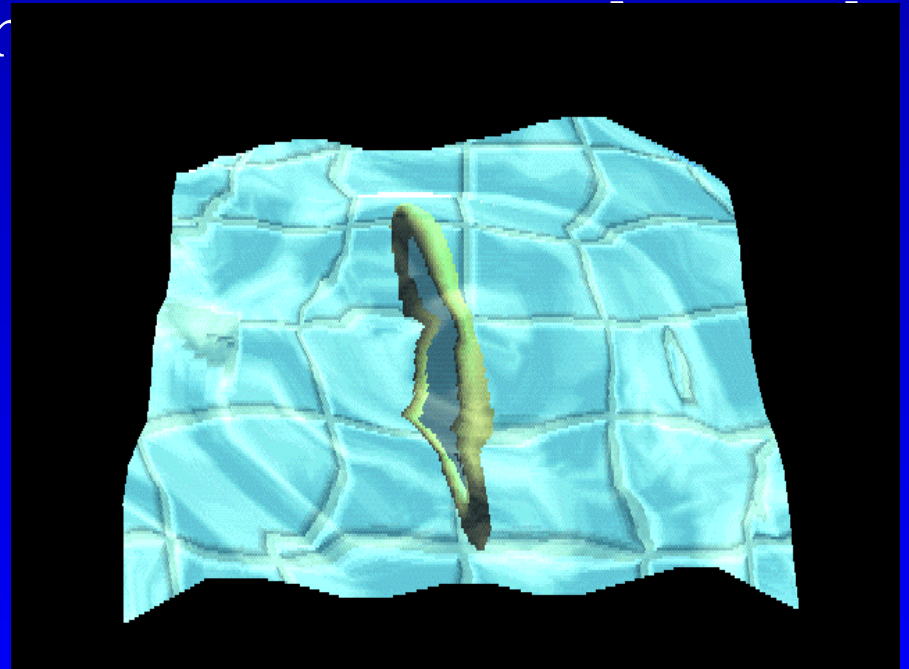
# *Polygon Penalties*

- ▶ If drawing speed is low because of polygon penalties,
  - Perform clipping if there is a margin left in a CPU process.
  - When scissoring penalties cannot be ignored, use small polygons, particularly around the screen boundaries.
  - Perform normal clipping to discard zero-area polygons and back-face polygons.



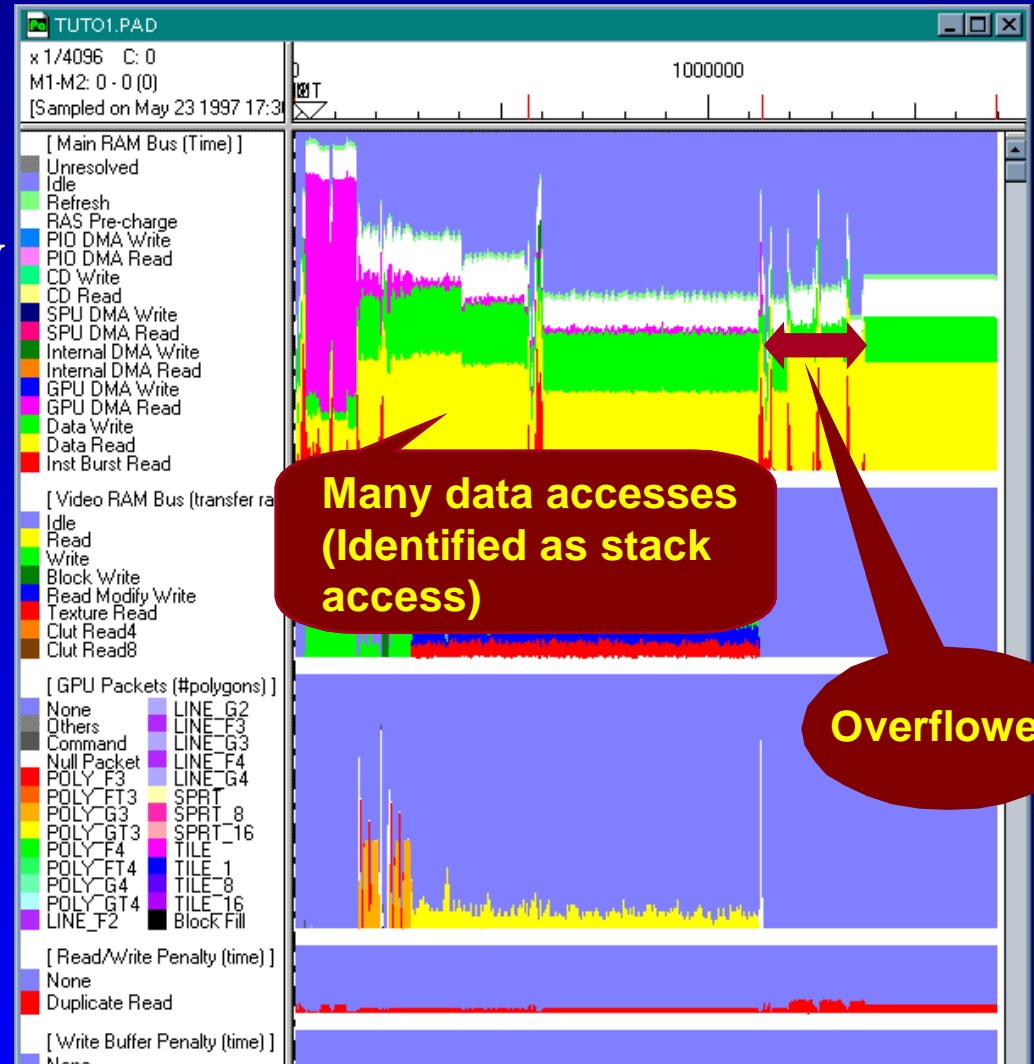
# *Example of Optimizing a Program*

- ▶ Texture address modulation
  - The texture address calculated from normal vector.
  - It must maintain 30 fps
- ▶ An improvement from 15 to 15 x 15!
  - About 50 % improvement in division ratio



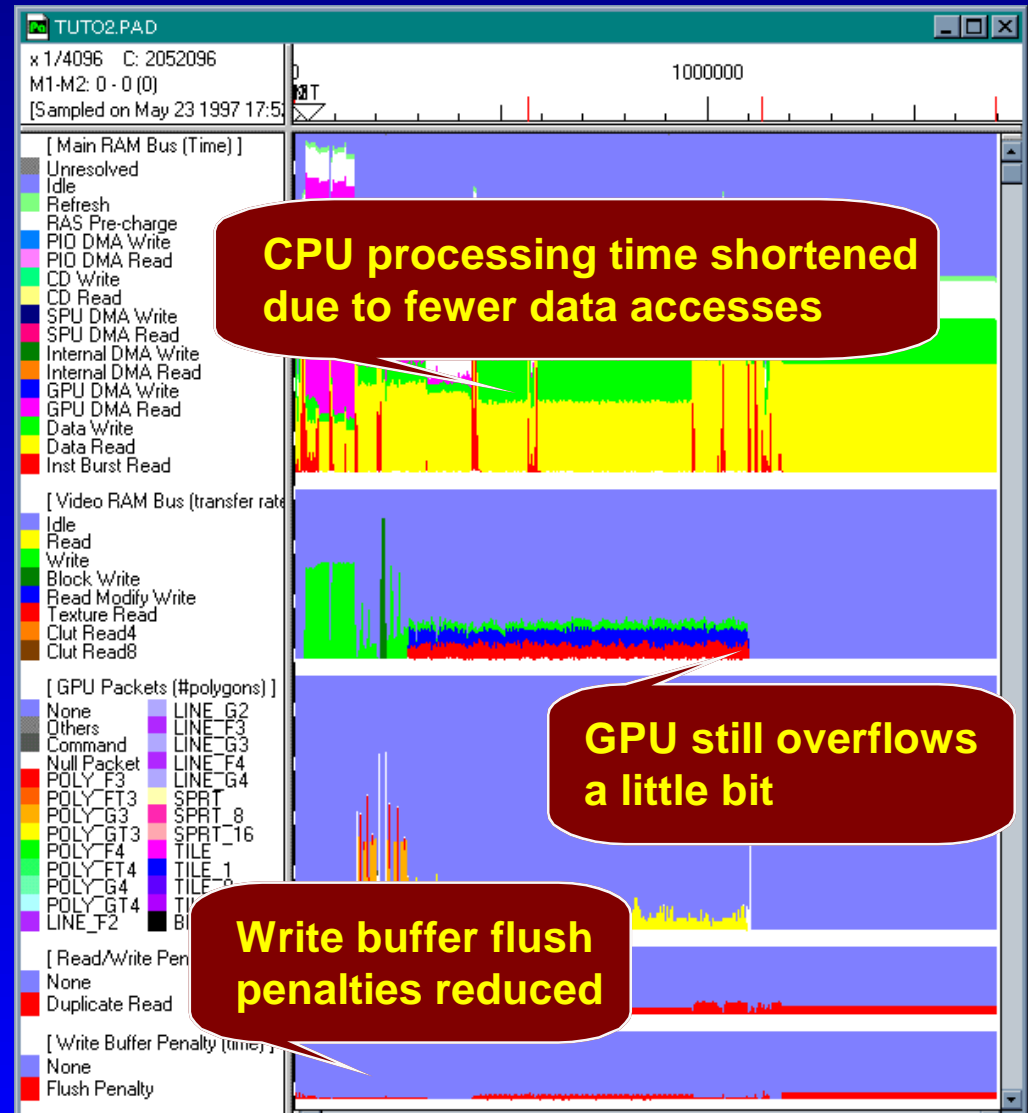
# Example of Optimizing a Program

- ▶ Using the 15x15 mesh
- ▶ DMPSX is already employed.
- ▶ Both CPU and GPU don't finish in two VBLANK periods.



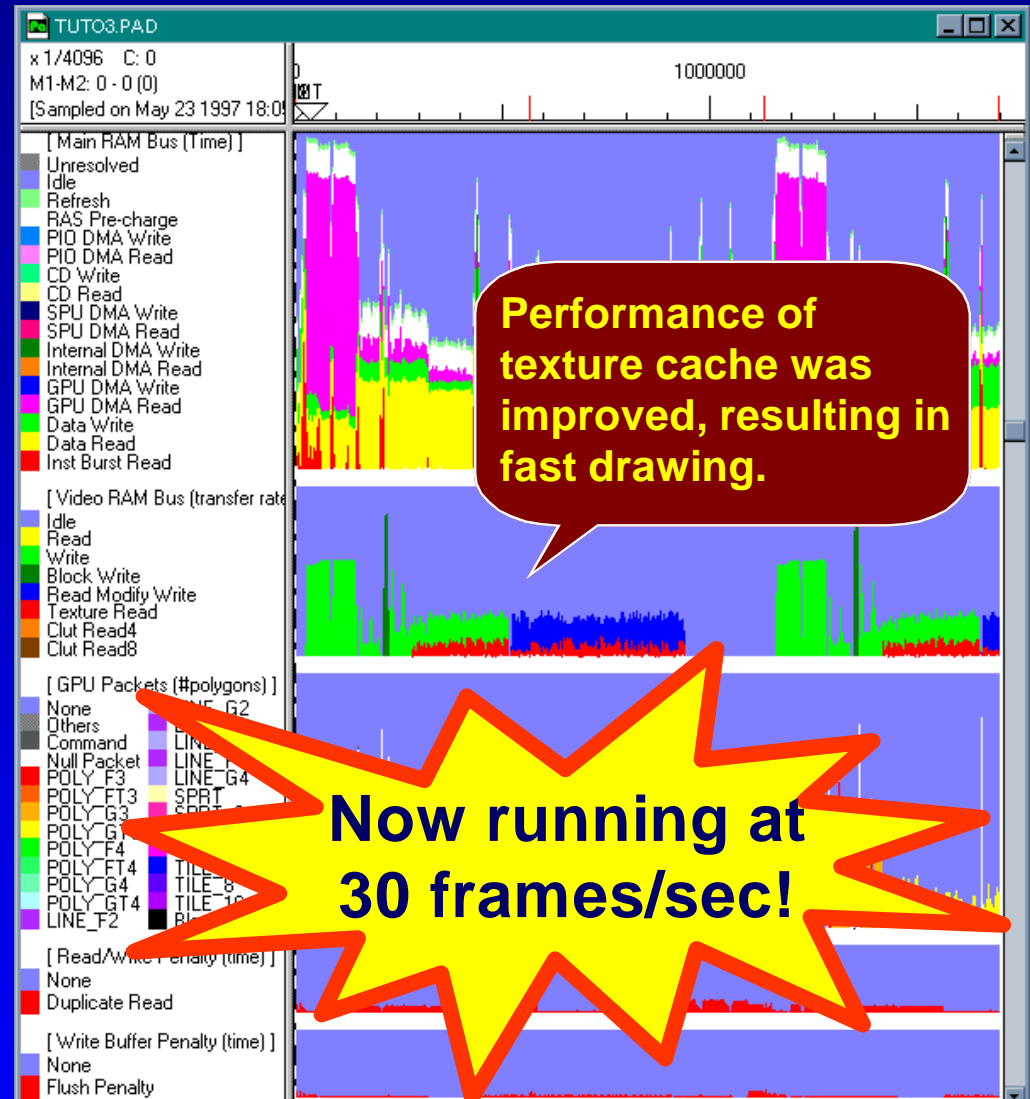
# Example of Optimizing a Program

- ▶ In the detected routine, the code was modified so that the scratch pad RAM would be used as a stack.
  - CPU now finishes in less than two VBLANK periods.



# Example of Optimizing a Program

- ▶ The ordering table was divided into two tables so that the hit ratio of the texture cache would increase.
- Both CPU and GPU finish in less than two VBLANK periods!



# Known Problems

- ▶ If the CPU is running on cache, it is hard to tell executing locations in a program.
  - Make a good guess from the patterns of data access symbols.
- ▶ The symbol file does not contain stack information.
  - Set the stack range in the option dialog box.
  - Use the `-Xm` option with `ccpsx` to generate a map file, since the DSM file output by `dumpsym` has no information about the stack segment.
  - When an I-cache miss occurs at an address in which the least four significant bits are 'C', the main RAMbus analysis identifies the transaction as a CPU data read cycle.
    - In such a case, even though an instruction cache miss occurred, only one long-word is read from the main RAM. Therefore there is no way to distinguish this instruction burst read cycle from CPU's data read cycles.
    - Care must be taken when searing for an instruction burst read transaction in the search command especially when an address is specified.

# *Tracing Unusual Phenomena*

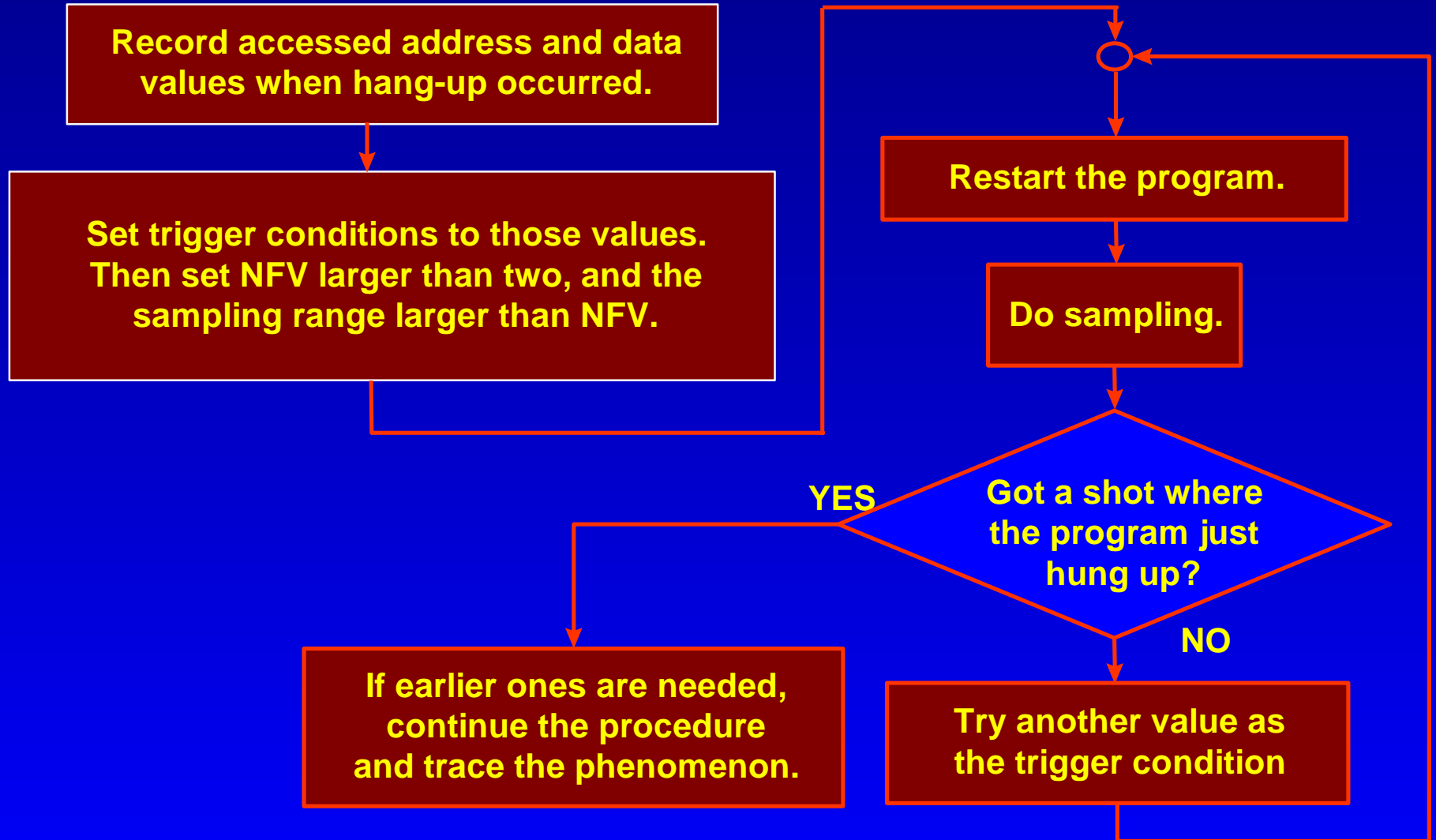
- ▶ The PA can analyze unusual phenomena such as hang-ups.
  - Trace the route to hang-up
- ▶ Very common cases
  - Forgetting initialization of library functions
    - Wrong order of calling **InitPad()** and **InitCard()**.
    - An uninitialized driver destroys the code area in a callback routine.

# *Tracing Unusual Phenomena*

## ▶ Very common cases

- Stack overflow in the scratch pad RAM.
  - Results in a bus error.
- Infinite loops & waiting for callbacks which never occur.
- Forgetting to call **ExitCriticalSection()** after calling **EnterCriticalSection()**.

# Tracing Unusual Phenomena





# *Evaluating Algorithms*

- ▶ Evaluation of a core engine, and improving its performance.
- ▶ Evaluation when introducing new data structure and algorithms
  - For example, employing joint-vertex mesh data
- ▶ Evaluation at an early stage of the design, and its use at a graphics design stage.
- ▶ Balancing the graphics design and algorithm

# Summary

- ▶ Visualization of processing by using the performance analyzer.
- ▶ Analysis by the programmer and its use as an analysis tool
- ▶ Effective use of various kinds of commands to identify the cause of processing bottlenecks
  - Various kinds of analyses
  - Statistics
  - Video memory viewer
  - Symbol access
  - Trigger function, Search command

# Summary

- ▶ Estimating improvements and determining the tuning strategy
- ▶ Optimimization
- ▶ Use for evaluation of algorithms, and its use for graphics design
- ▶ Adding more features such as improvements in graphics qualities when the CPU and GPU have more bandwidth left.
  - Introducing mip-mapping

# Summary

- ▶ Use as an experimental tool at an early stage in a design phase
- ▶ Useful data in discussion between programmers and graphics designers in a group work
- ▶ Use as a monitoring tool to examine unusual phenomena

*The End*