graphics hardware 06

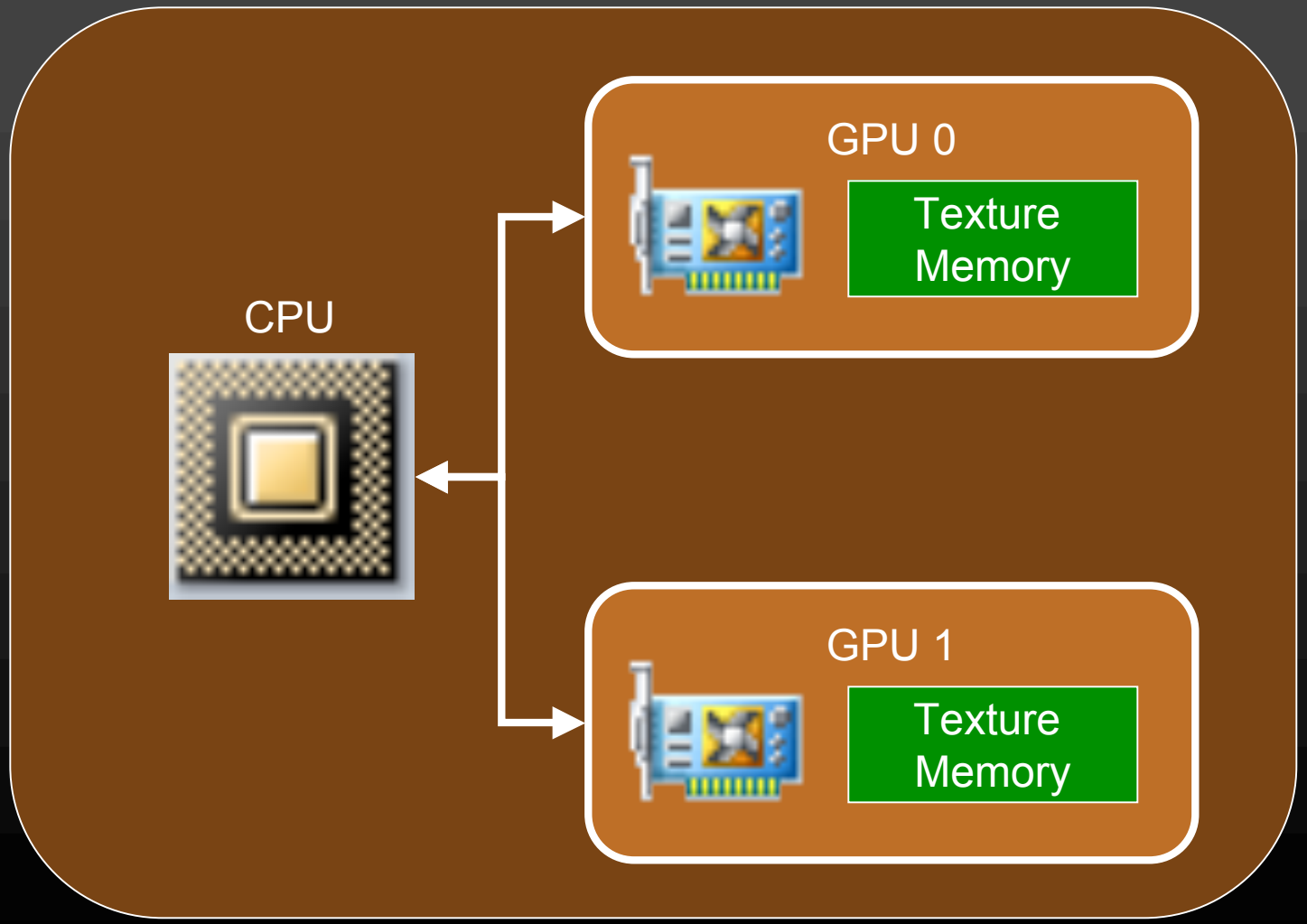
Distributed Texture Memory in a Multi-GPU Environment

Adam Moerschell

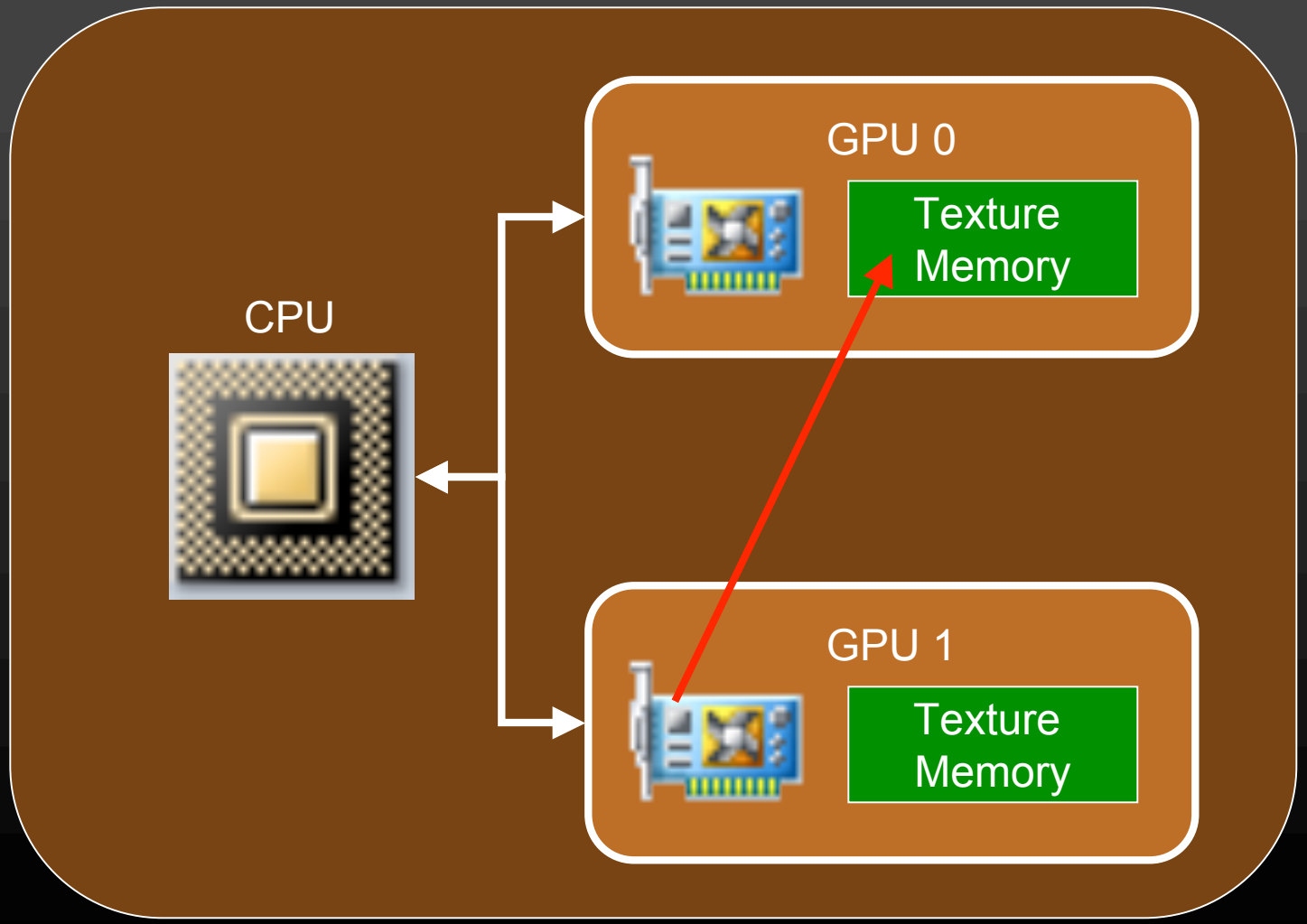
John Owens

University of California, Davis

Introduction



Introduction



Introduction

Trends

- GPU is now a programmable parallel processor
- More general purpose computation
- Multi-CPU/Multi-Core => Multi-GPU

Applications

- Large Data Set Visualization
- High End Interactive Graphics
- Physical Simulation
- General Purpose Computation

Background

Current Multi-GPU Graphics Systems

- Crossfire and SLI
  - ✓ Increased Triangle Rate
  - ✓ Increased Pixel Rate
  - x Texture Memory does not scale as more GPUs are added
- Chromium
  - ✓ Provides scalability
  - x Only works on streams of graphics commands
    - No access to stages of graphics pipeline not exposed to programmer

Implementation

Goal: Distributing Texture Memory

- Scalable to Many GPUs
  - Single Node
  - Clustered Nodes
- Globally Accessible
  - All memory blocks visible to all GPUs
- System Transparent to Programmer
  - No need to manage memory by hand

Implementation

Goal: Distributing Texture Memory

- Our goal is to show important mechanisms for accomplishing this
- Performance is secondary
- We provide insights as to what can change to help make this system work better

Implementation

Memory System

- Based on Distributed Shared Memory (DSM)
  - Scalable
  - Creates a global memory space for all GPUs to operate in
  - Able to create a definable and enforceable memory consistency model
- Sequential Consistency Model
  - Texture accesses occur in program order
  - Writes complete in order issued
  - Reads only complete after previous writes complete

Implementation

Data Structures

- Global + Distributed + Consistent
- Need some way to keep track of memory
  - Directory
    - Stored in CPU memory
    - Holds state information for each memory block
    - Keeps a copy of the most recent version of a block
    - Scalable in a multi-node environment
    - Works “under the hood” -> transparent

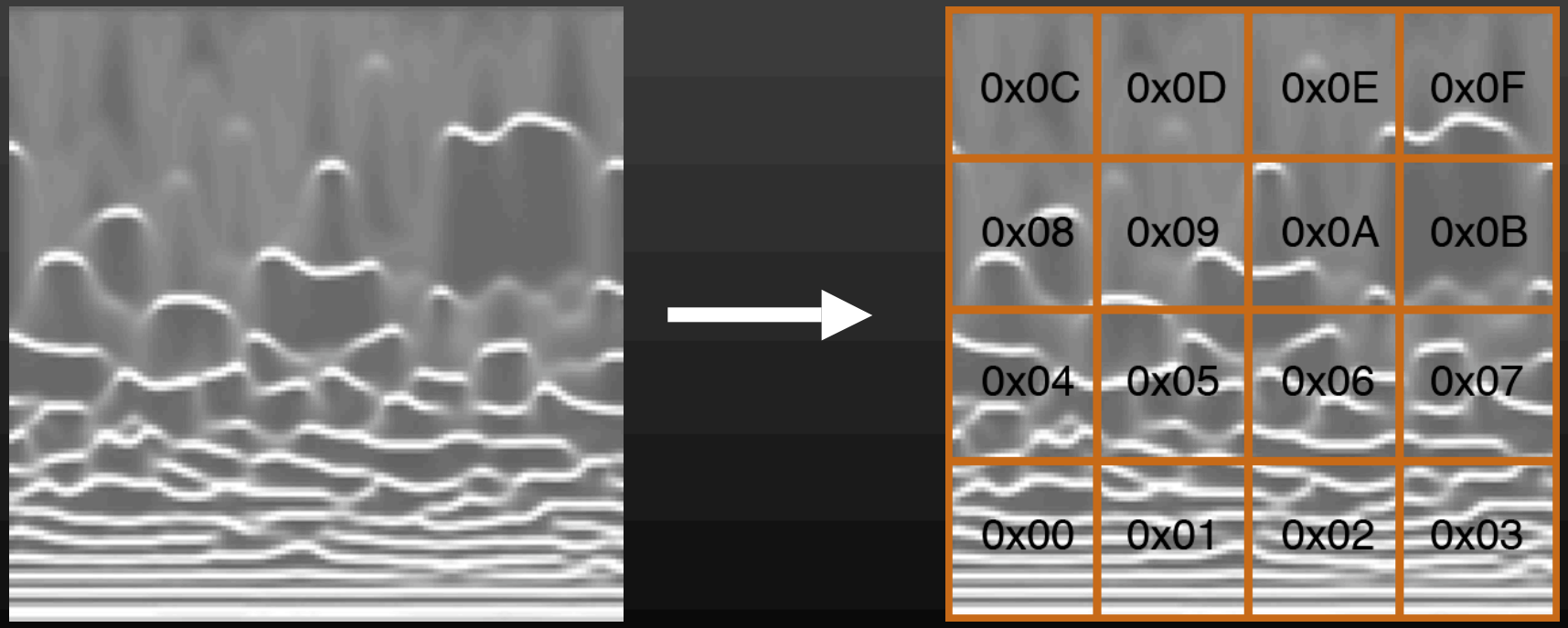
Implementation

Data Structures

- Memory Block
  - Textures are too large
  - Texels are too small
  - Page - contiguous block of texels from a single texture
- GPU Structures
  - Page Table and Physical Memory textures
    - Page Table texture stores validity and pointers to pages in the Physical Memory texture
    - Physical Memory texture stores page data
    - Texel access is an indirect lookup via the page table

Implementation

Texture Load



A texture is broken into pages as it is loaded into memory.

Implementation

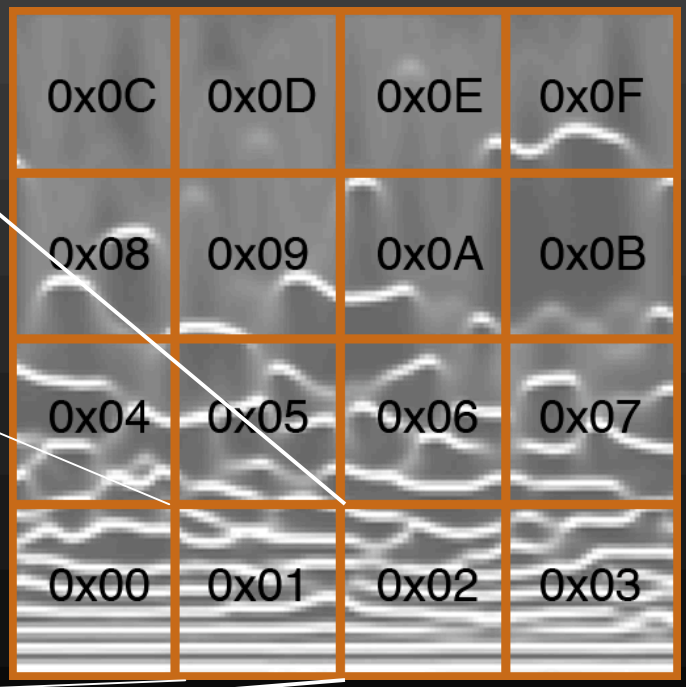
Texture Load

Directory Entry
// Global Address
page_num = 0x01;

// Dirty bit
dirty = false;

// Valid bit for each GPU
valid[num_gpus] = false;

// Pointer to data in CPU memory
*data;



A directory entry is created for each page.
The entry contains important state information.

Implementation

Texture Load

Directory Entry
// Global Address
page_num = 0x01;

// Dirty bit
dirty = false;

// Valid bit for each GPU
valid[num_gpus] = false;

// Pointer to data in CPU memory
*data;

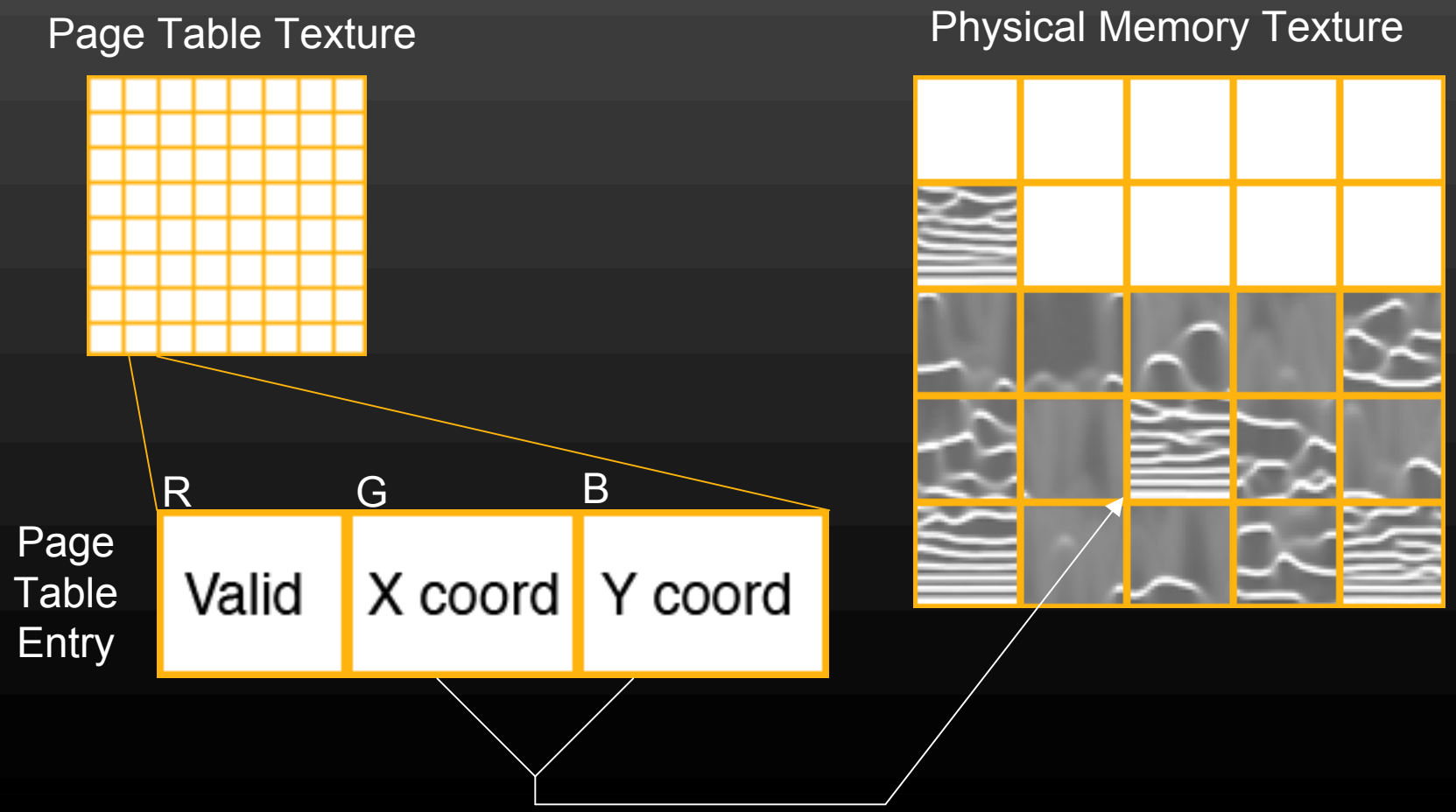
Directory

PN	D	V[0]	…	V[N-1]	data
0x00	F	F	…	F	*ptr
0x01	F	F	…	F	*ptr
…	…	…	…	…	…
0x0F	F	F	…	F	*ptr

The directory resides in CPU memory, and keeps track of the state of every page in the system.

Implementation

GPU Page Table Lookup



Implementation

Executing GL Code

- Pages loaded to the GPU PhysMem on demand
- Must take care at any texture access in a shader
  - Break texture access into two passes

Original Fragment Program

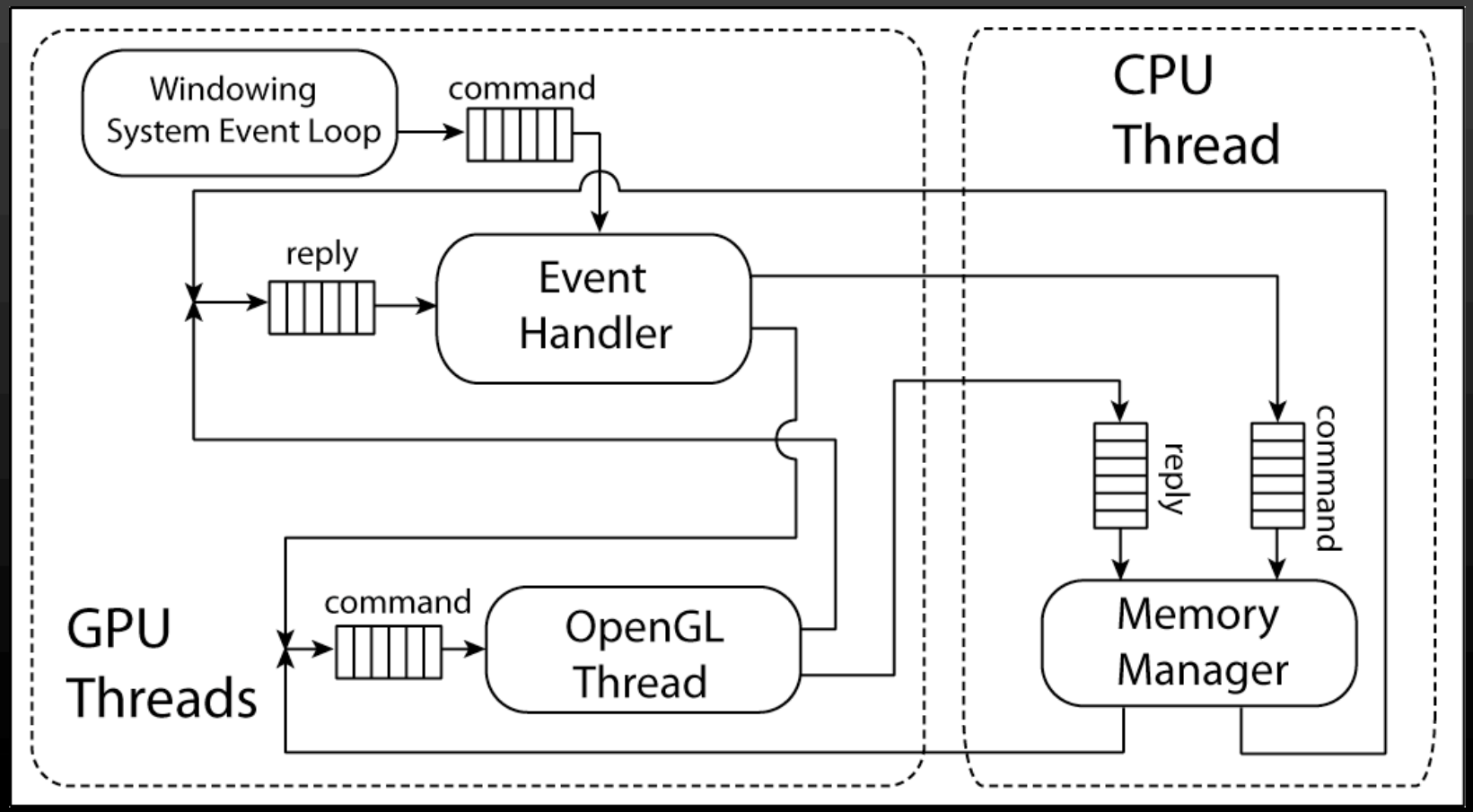
```
float4 main(float2 tc : TEXCOORD0,
  uniform sampler2D texture)
{
  …
  float4 data=tex2D(tc,texture);
  …
}
```

Fragment Program 1 (Pass 1)
- Determine if data at `tc` is resident to texture memory
- render requests to buffer

CPU Handler
- Read back request buffer
- Process requests and transfer data to GPU's texture memory
- Update page table

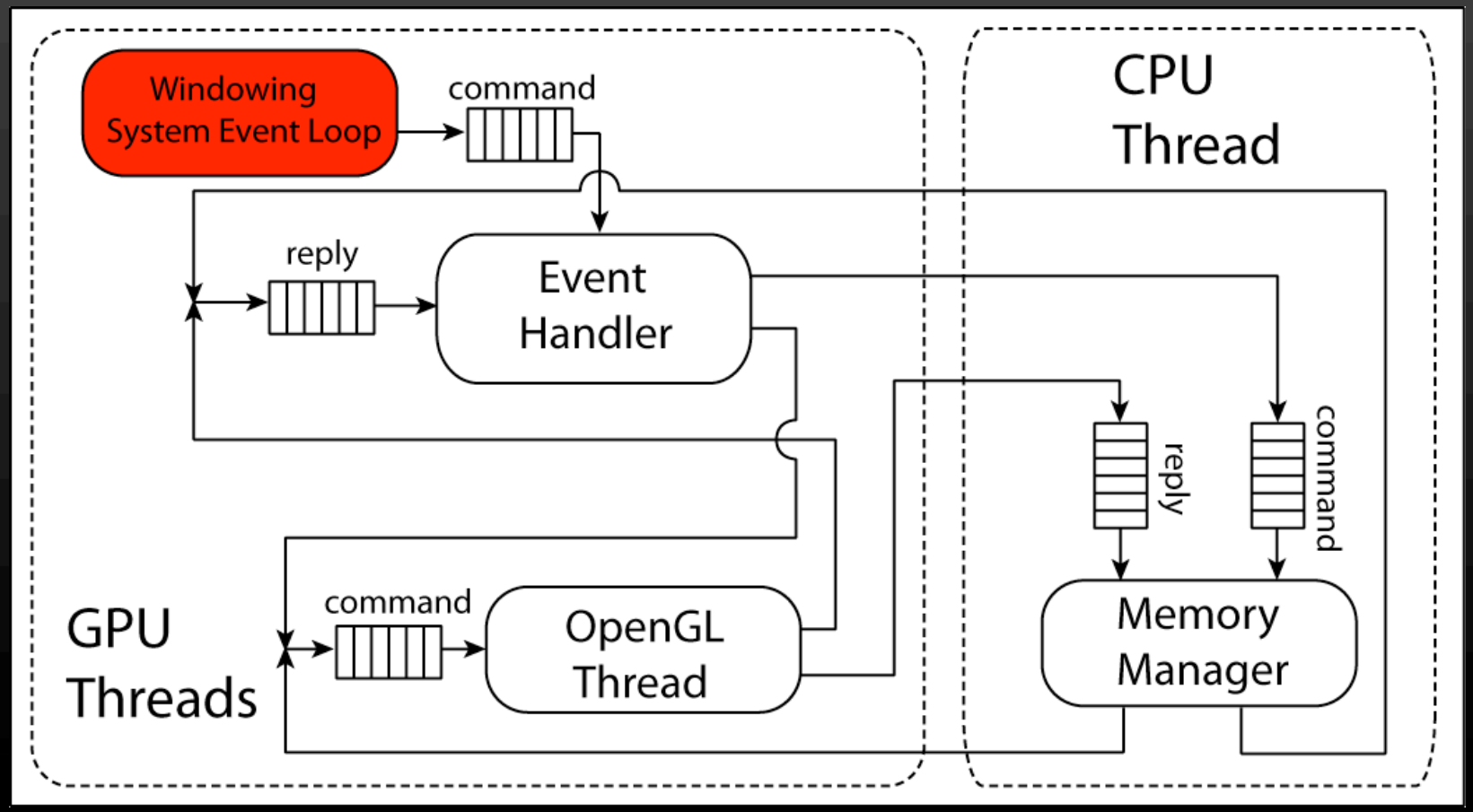
Fragment Program 2 (Pass 2)
- Dependent texture read page table ➔ physical memory

Threading System



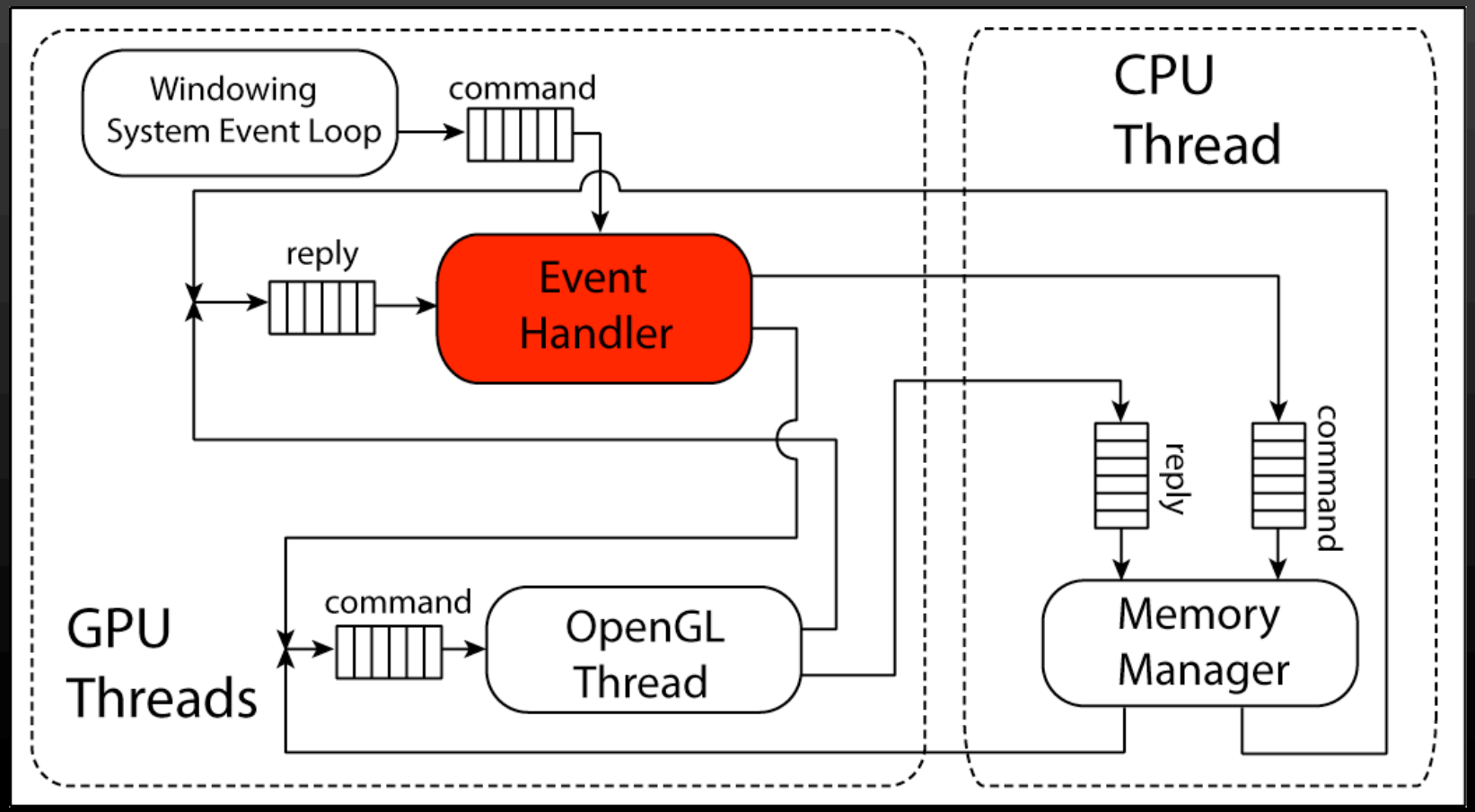
Threading System

- Windowing System Event Loop
  - Captures User and Windowing System Input



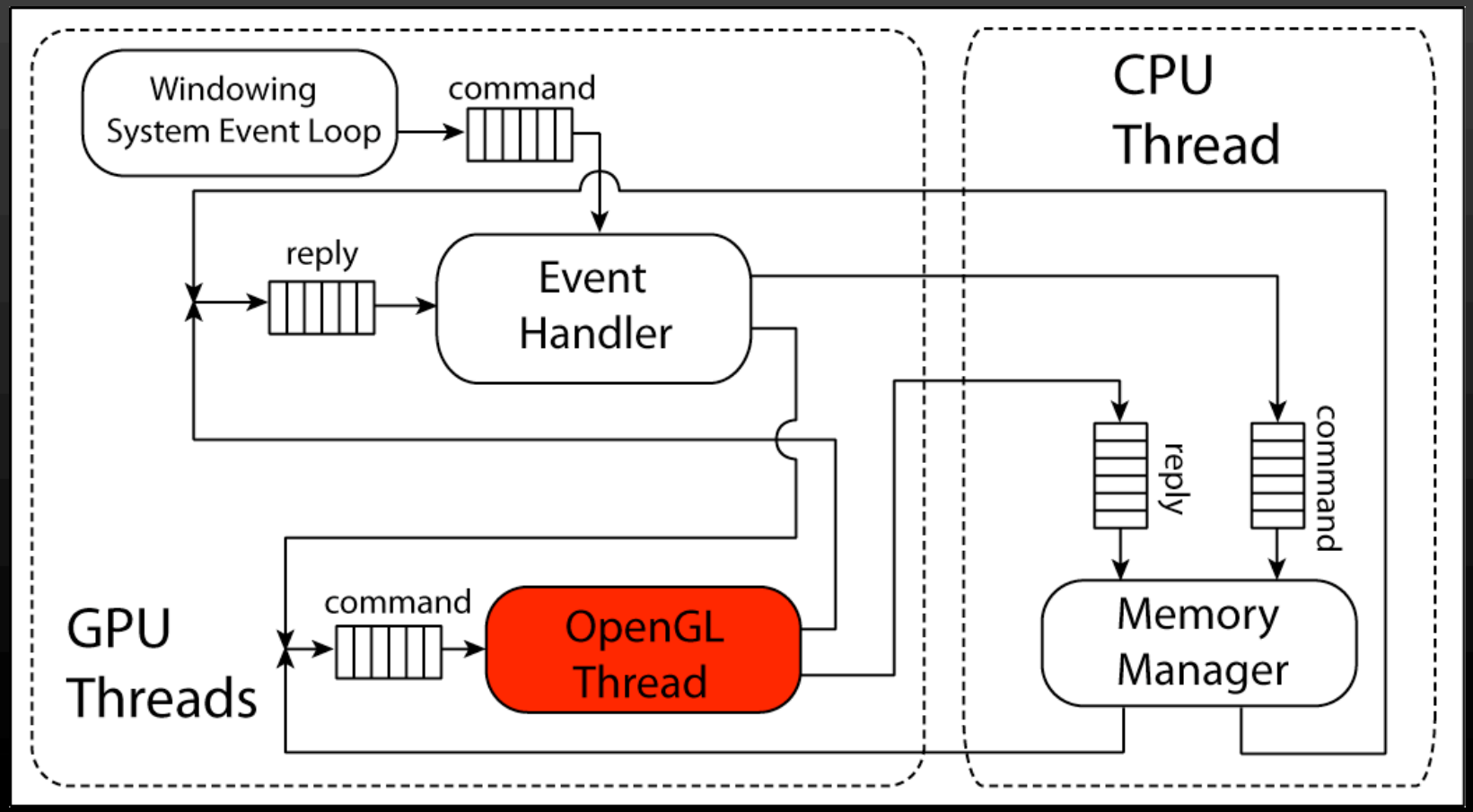
Threading System

- Event Handler
  - Responds to events and executes callbacks



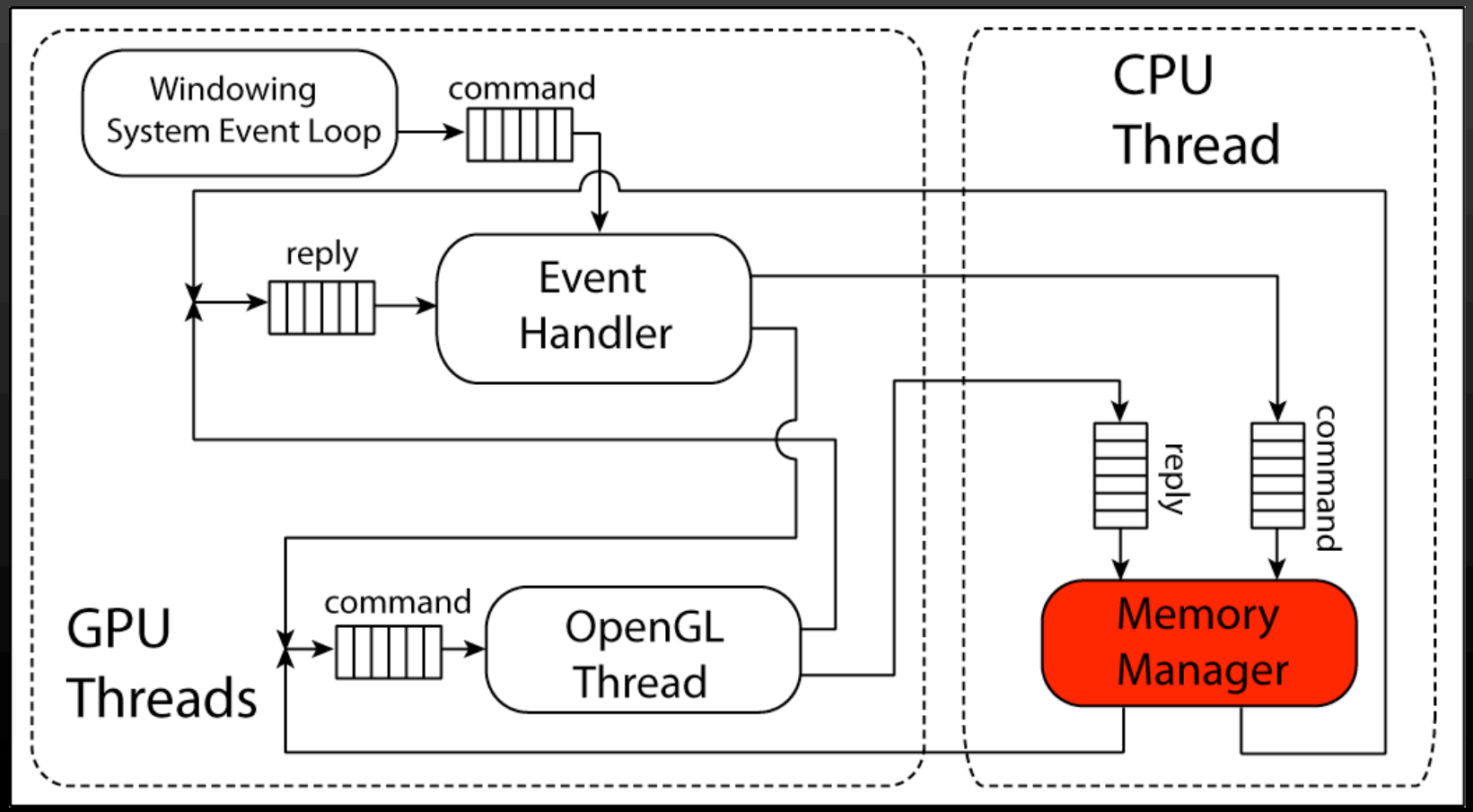
Threading System

- OpenGL Thread
  - Owns Graphics Context and executes all GL commands

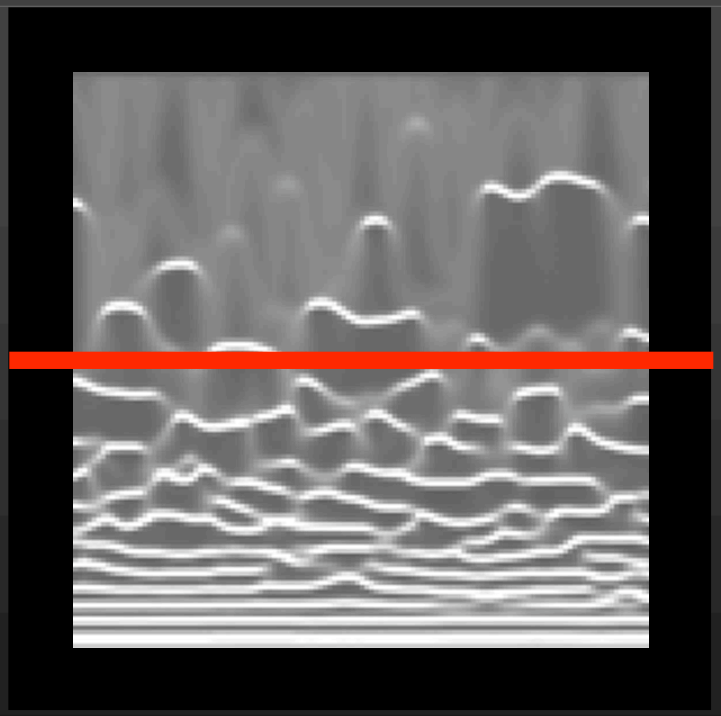


Threading System

- Memory Manager
  - Manages Directory and Memory Consistency



Texture Read

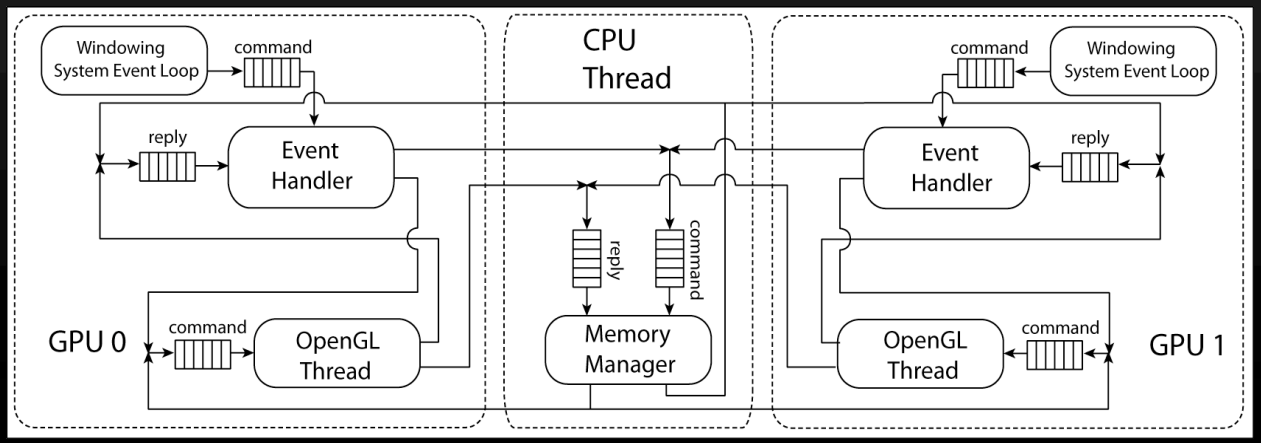


GPU 0

GPU 1

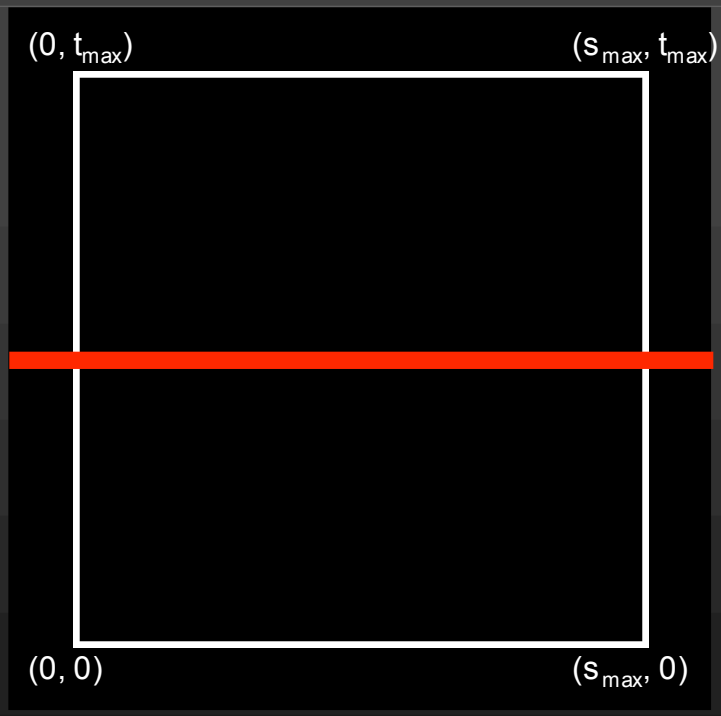
Dual GPU Configuration

- Goal: draw texture mapped quad to screen
- Only parts of texture needed on each card
- Currently no data loaded into texture memory



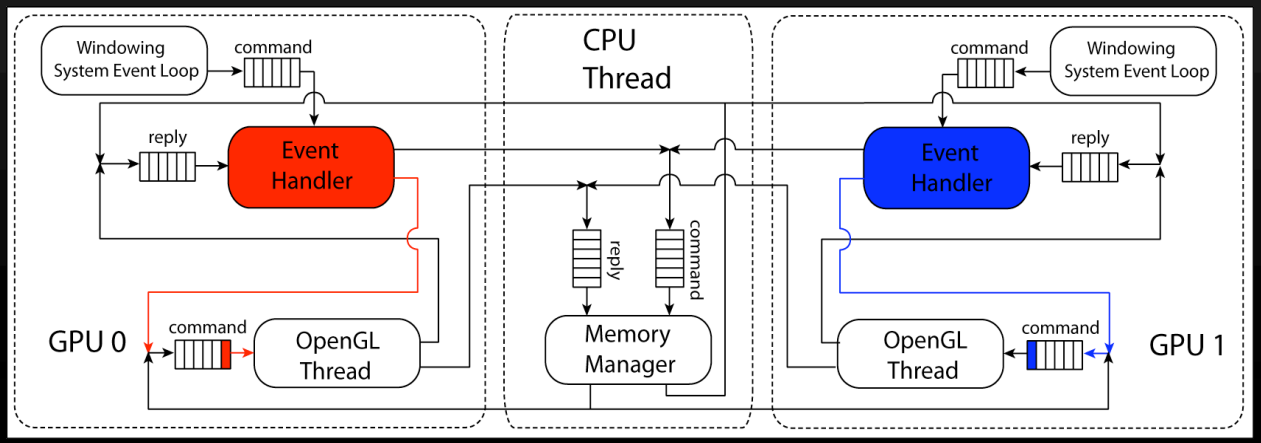
ADDR	D	V[0]	V[1]
0x00	F	F	F
0x01	F	F	F
0x02	F	F	F
0x03	F	F	F
0x04	F	F	F
0x05	F	F	F
0x06	F	F	F
0x07	F	F	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F

Texture Read

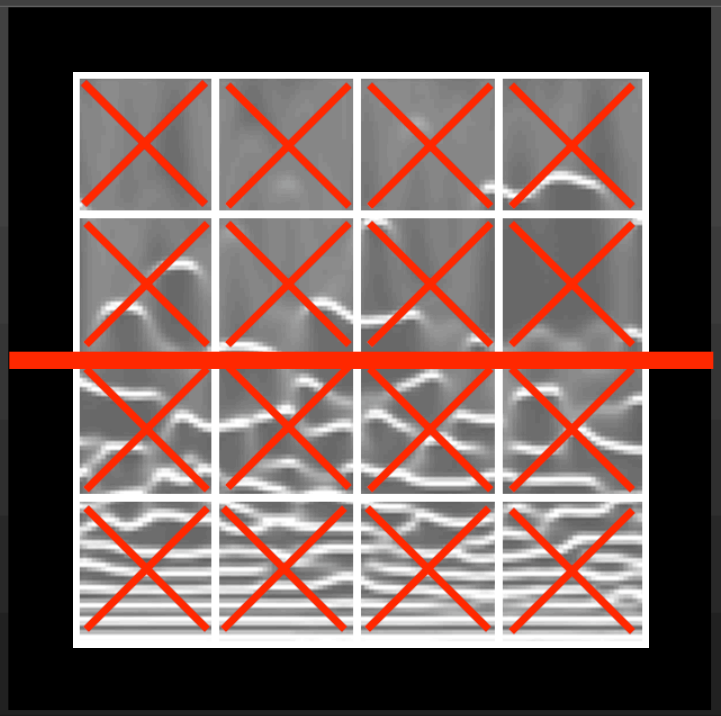


Event Handlers
- Receive screen refresh event
- Initiate First Pass
- Send Geometry and texture coordinates to their OpenGL Thread

ADDR	D	V[0]	V[1]
0x00	F	F	F
0x01	F	F	F
0x02	F	F	F
0x03	F	F	F
0x04	F	F	F
0x05	F	F	F
0x06	F	F	F
0x07	F	F	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F



Texture Read

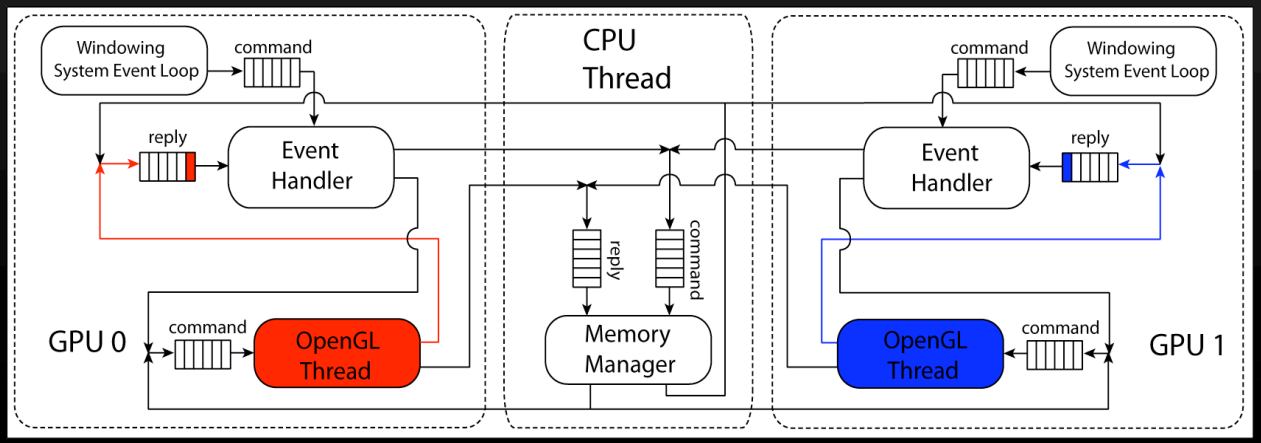


GPU 0

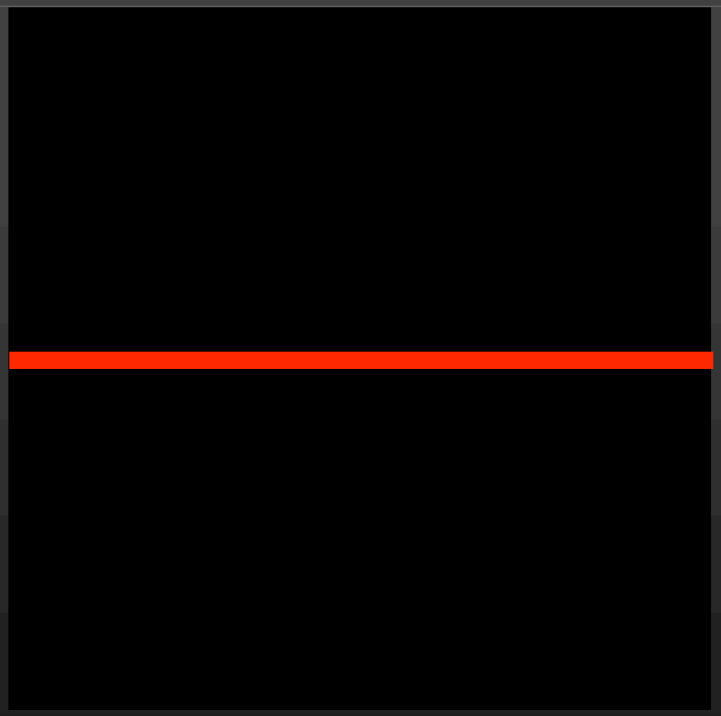
OpenGL Threads
- Receive Geometry and Tex Coords
- Look up required pages in page table and output requests for pages that are not local
- Results are read back and sent to Event Handler

GPU 1

ADDR	D	V[0]	V[1]
0x00	F	F	F
0x01	F	F	F
0x02	F	F	F
0x03	F	F	F
0x04	F	F	F
0x05	F	F	F
0x06	F	F	F
0x07	F	F	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F



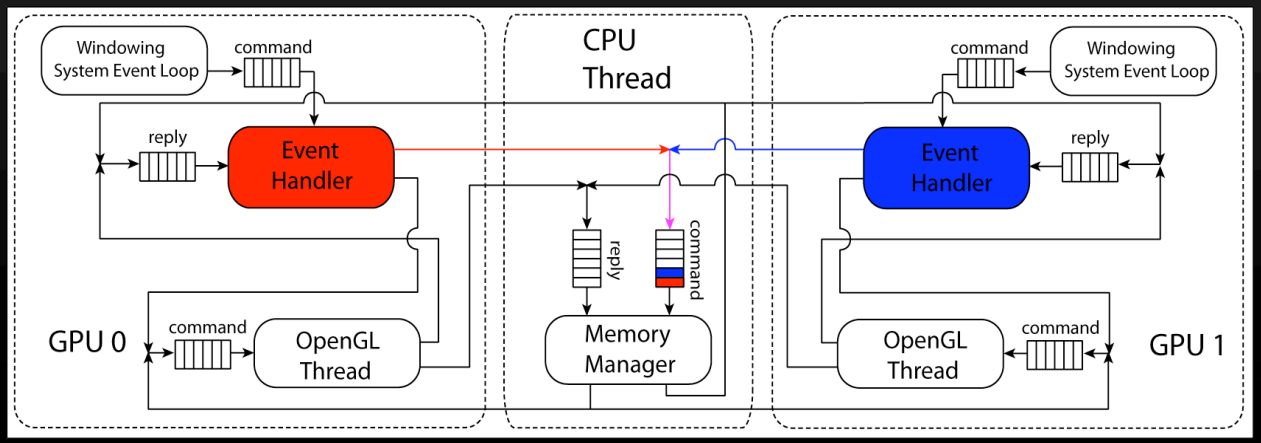
Texture Read



GPU 0

Event Handlers
- Receive Page Requests for invalid pages
- Request pages from Memory Manager

GPU 1



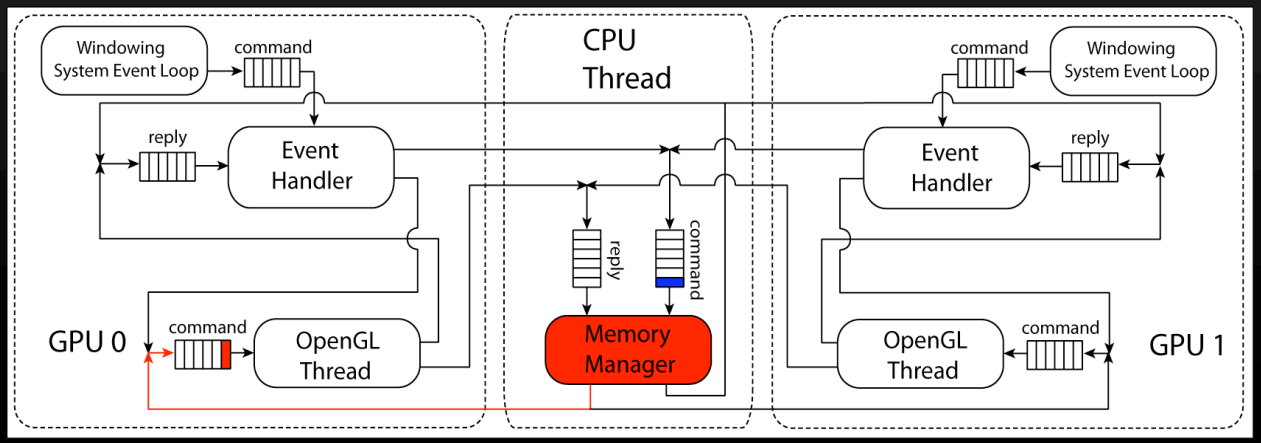
ADDR	D	V[0]	V[1]
0x00	F	F	F
0x01	F	F	F
0x02	F	F	F
0x03	F	F	F
0x04	F	F	F
0x05	F	F	F
0x06	F	F	F
0x07	F	F	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F

Texture Read

GPU 0

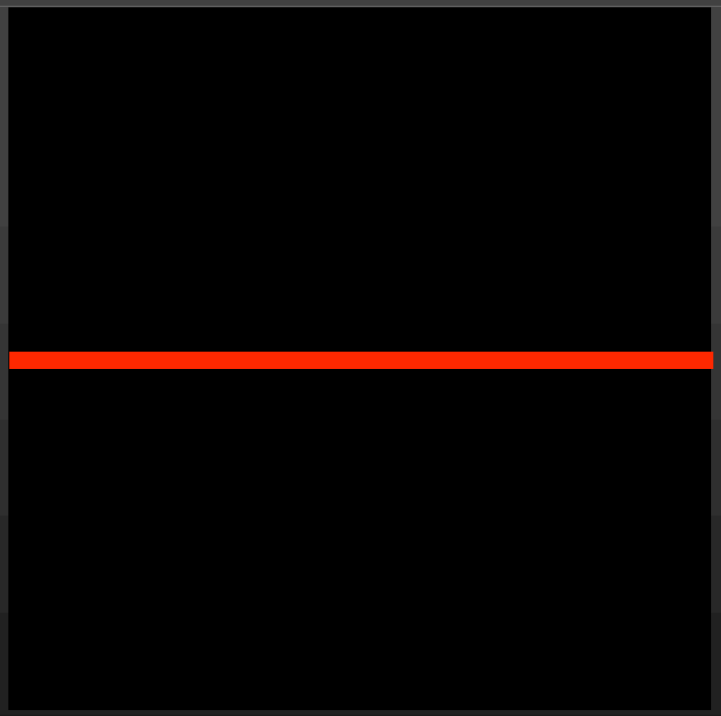
Memory Manager
- Receives first request
- Locates requested pages
- Sends pages to OGL Thread

GPU 1



ADDR	D	V[0]	V[1]
0x00	F	F	F
0x01	F	F	F
0x02	F	F	F
0x03	F	F	F
0x04	F	F	F
0x05	F	F	F
0x06	F	F	F
0x07	F	F	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F

Texture Read

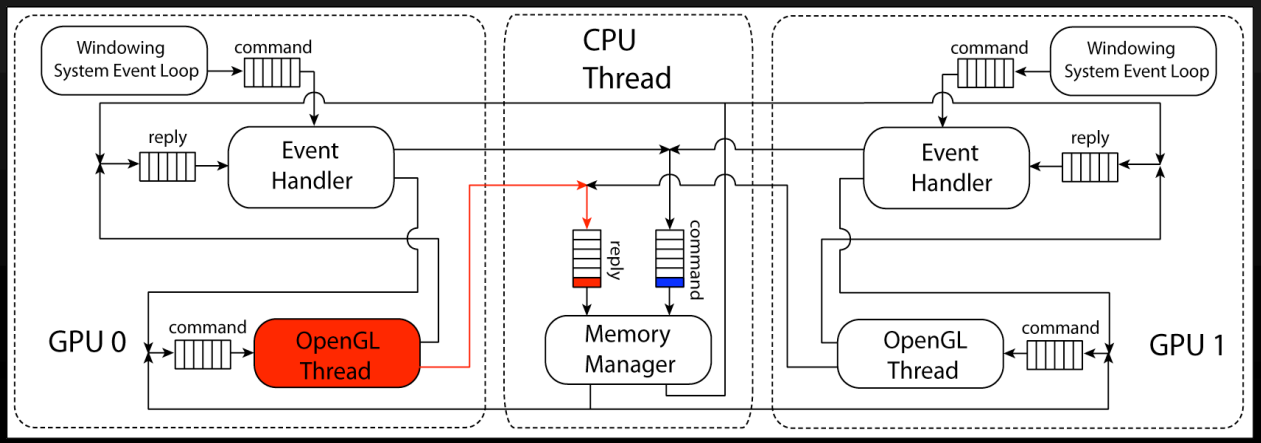


GPU 0

OpenGL Thread 0

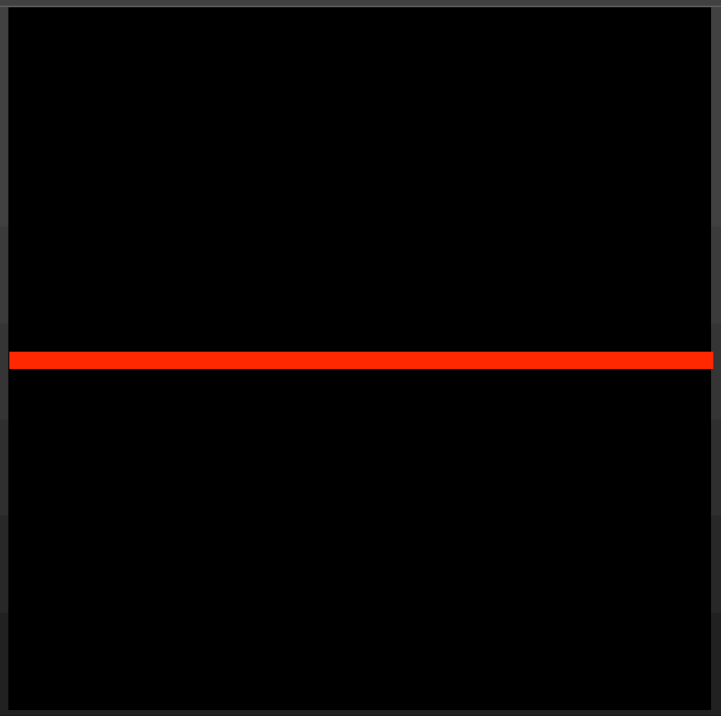
- Places Pages into Physical Memory Texture
- Updates Page Table Entries
- Replies Success to Memory Manager

GPU 1



ADDR	D	V[0]	V[1]
0x00	F	F	F
0x01	F	F	F
0x02	F	F	F
0x03	F	F	F
0x04	F	F	F
0x05	F	F	F
0x06	F	F	F
0x07	F	F	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F

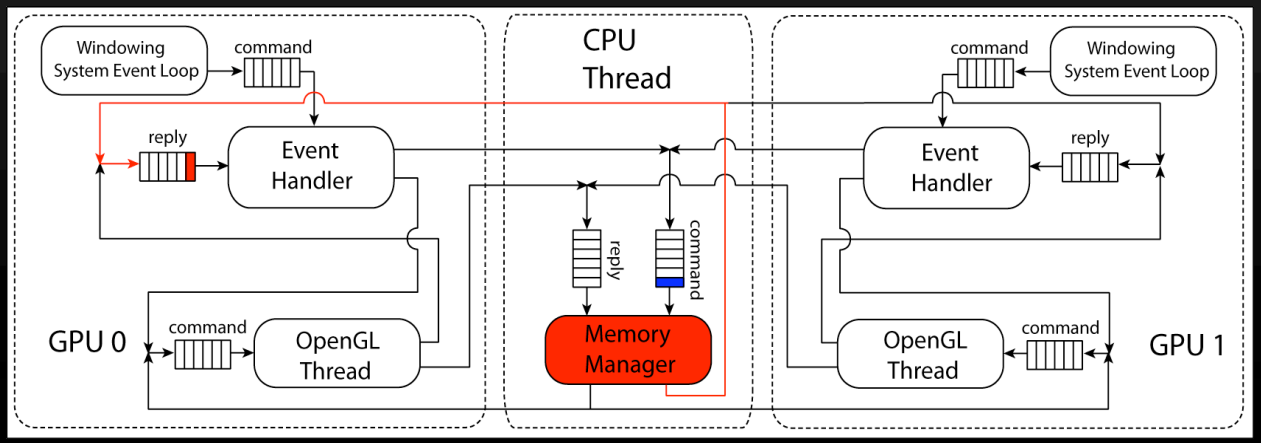
Texture Read



GPU 0

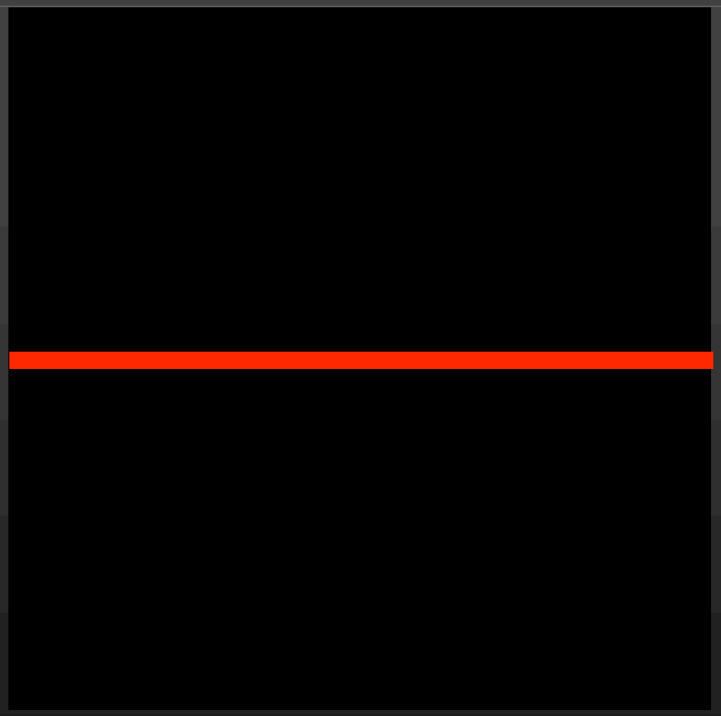
Memory Manager
- Updates Directory Entries
- Tells Event Handler it is okay to proceed

GPU 1



ADDR	D	V[0]	V[1]
0x00	F	T	F
0x01	F	T	F
0x02	F	T	F
0x03	F	T	F
0x04	F	T	F
0x05	F	T	F
0x06	F	T	F
0x07	F	T	F
0x08	F	F	F
0x09	F	F	F
0x0A	F	F	F
0x0B	F	F	F
0x0C	F	F	F
0x0D	F	F	F
0x0E	F	F	F
0x0F	F	F	F

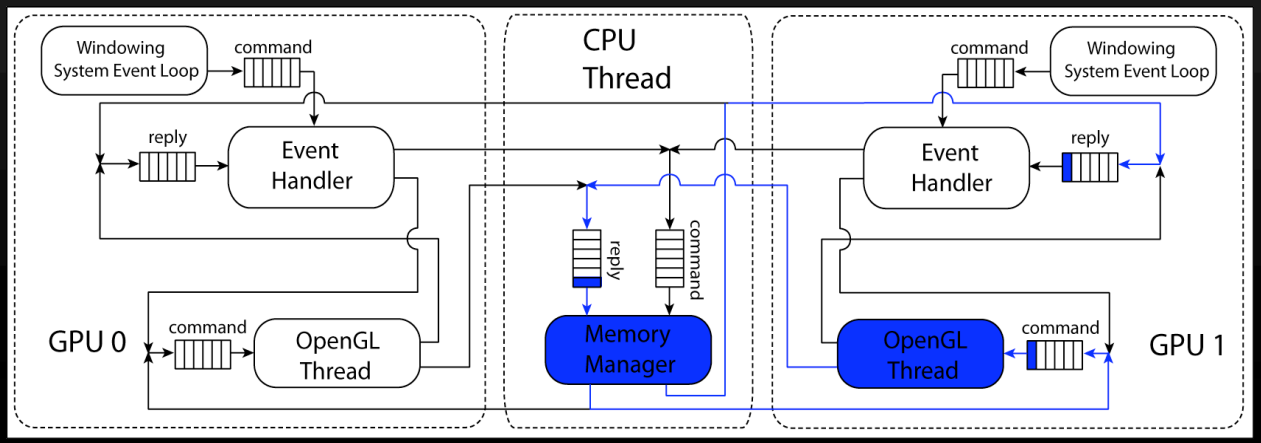
Texture Read



GPU 0

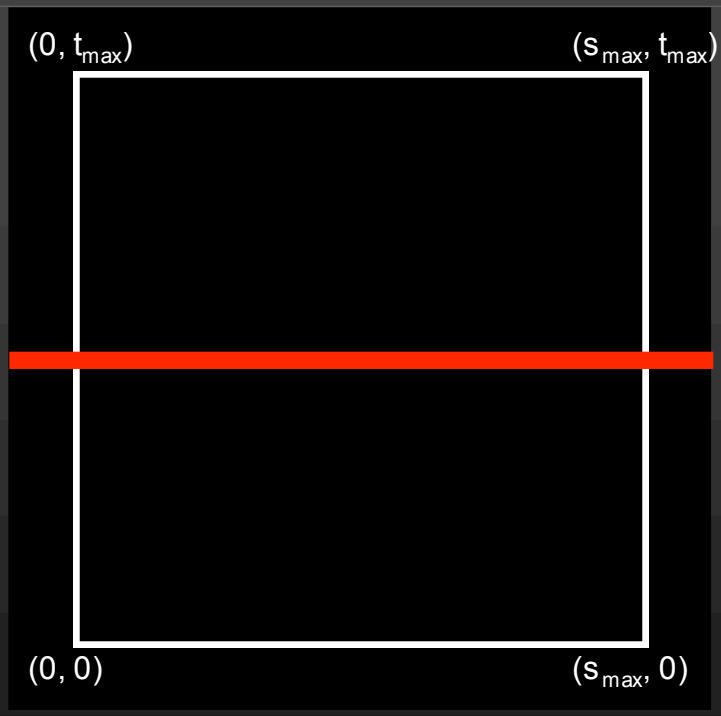
Memory Manager
- Receives second request
- Same Procedure for GPU 1

GPU 1



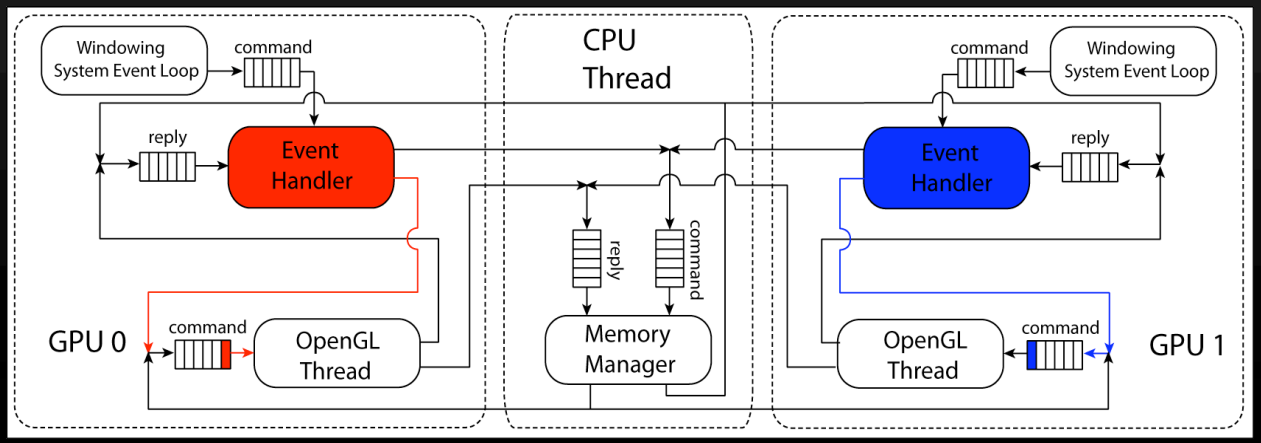
ADDR	D	V[0]	V[1]
0x00	F	T	F
0x01	F	T	F
0x02	F	T	F
0x03	F	T	F
0x04	F	T	F
0x05	F	T	F
0x06	F	T	F
0x07	F	T	F
0x08	F	F	T
0x09	F	F	T
0x0A	F	F	T
0x0B	F	F	T
0x0C	F	F	T
0x0D	F	F	T
0x0E	F	F	T
0x0F	F	F	T

Texture Read

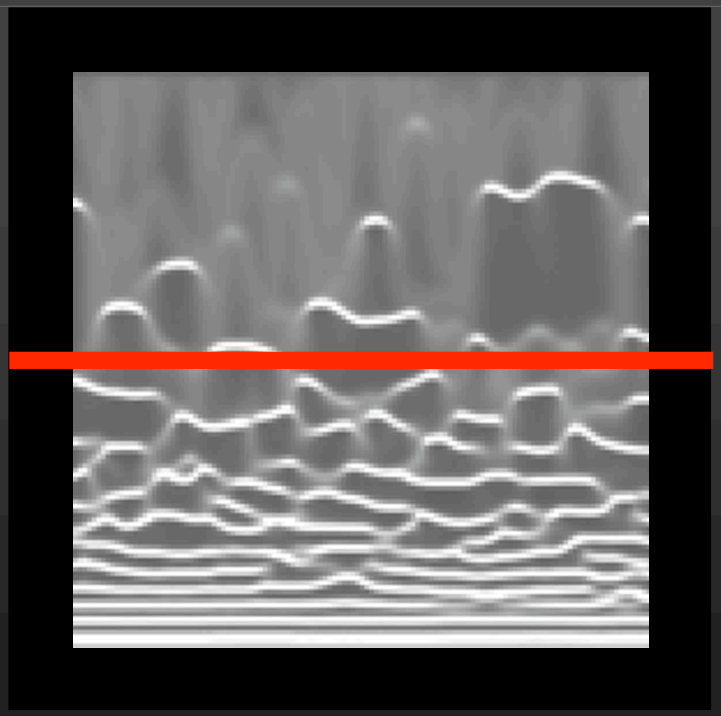


Event Handlers
- Initiate Second Pass
  - All necessary texture data is local to GPU
- Re-send Geometry and texture coordinates to their OpenGL Thread

ADDR	D	V[0]	V[1]
0x00	F	T	F
0x01	F	T	F
0x02	F	T	F
0x03	F	T	F
0x04	F	T	F
0x05	F	T	F
0x06	F	T	F
0x07	F	T	F
0x08	F	F	T
0x09	F	F	T
0x0A	F	F	T
0x0B	F	F	T
0x0C	F	F	T
0x0D	F	F	T
0x0E	F	F	T
0x0F	F	F	T



Texture Read

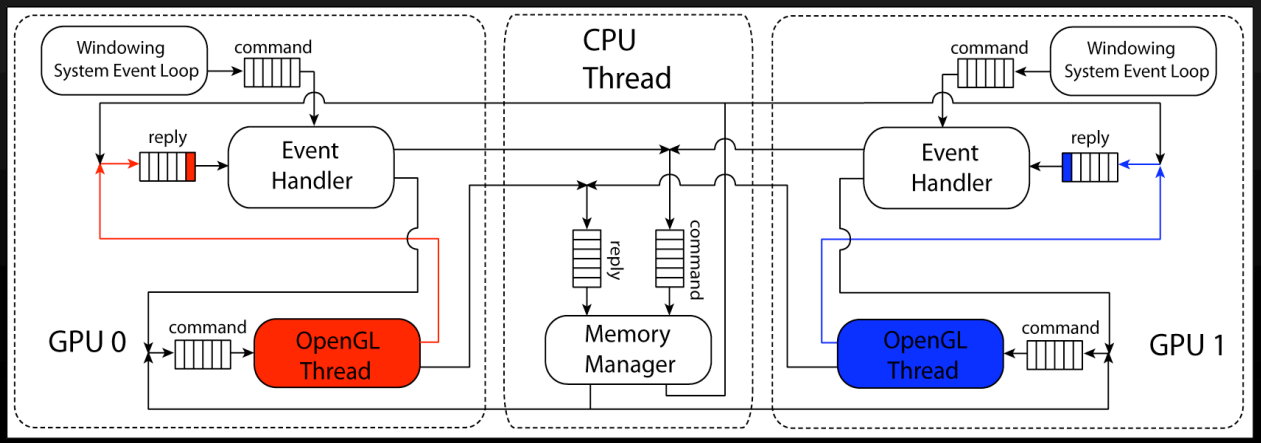


GPU 0

OpenGL Threads

- Receive Geometry and Tex Coords
- Look up required pages in page table and physical memory
- Output final pixel value to framebuffer
- Reply operation complete to Event Handler

GPU 1



ADDR	D	V[0]	V[1]
0x00	F	T	F
0x01	F	T	F
0x02	F	T	F
0x03	F	T	F
0x04	F	T	F
0x05	F	T	F
0x06	F	T	F
0x07	F	T	F
0x08	F	F	T
0x09	F	F	T
0x0A	F	F	T
0x0B	F	F	T
0x0C	F	F	T
0x0D	F	F	T
0x0E	F	F	T
0x0F	F	F	T

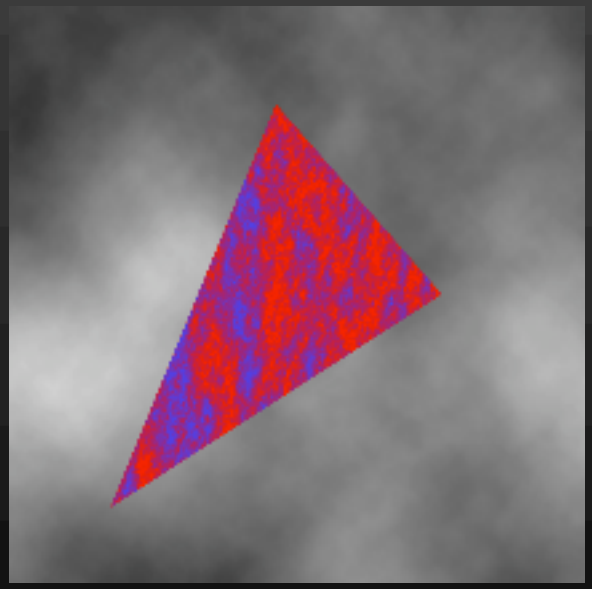
Programmer's View

- Memory is global
  - Only worry about Texture ID and S,T coords
- Independent command stream to each GPU
  - Partition image space
- Currently shaders are rewritten by hand
  - Could be easily automated using Mio-like technology [Riffel et al. 2004]

Texture Write

Goal: Render textured triangle into original texture

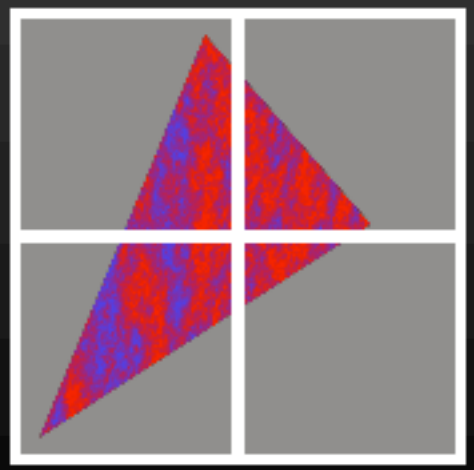
Original Texture



Modified Texture

Texture Write

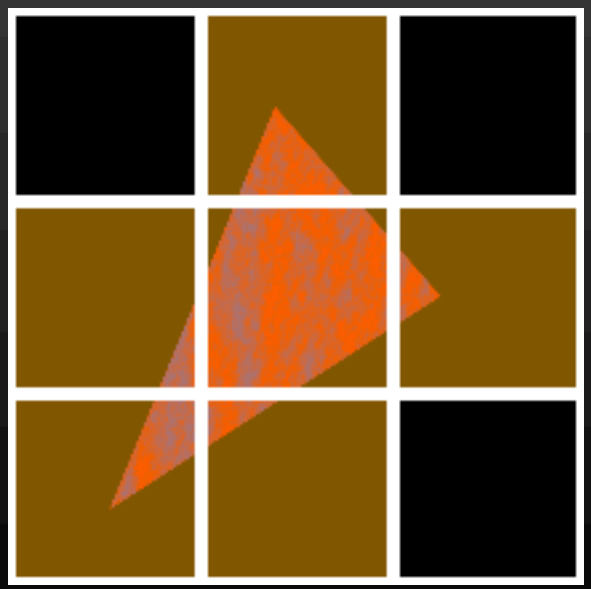
Pass 1: Check read dependencies in the same manner as a texture read. Load required texture pages from directory to GPU texture memory.



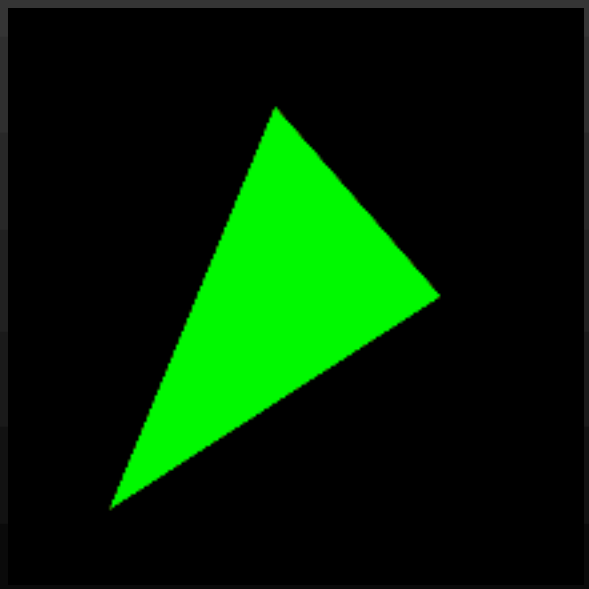
Texture Write

Pass 2:

- Render textured triangle to temporary buffer
- Request exclusive copy of modified pages
- Create write mask



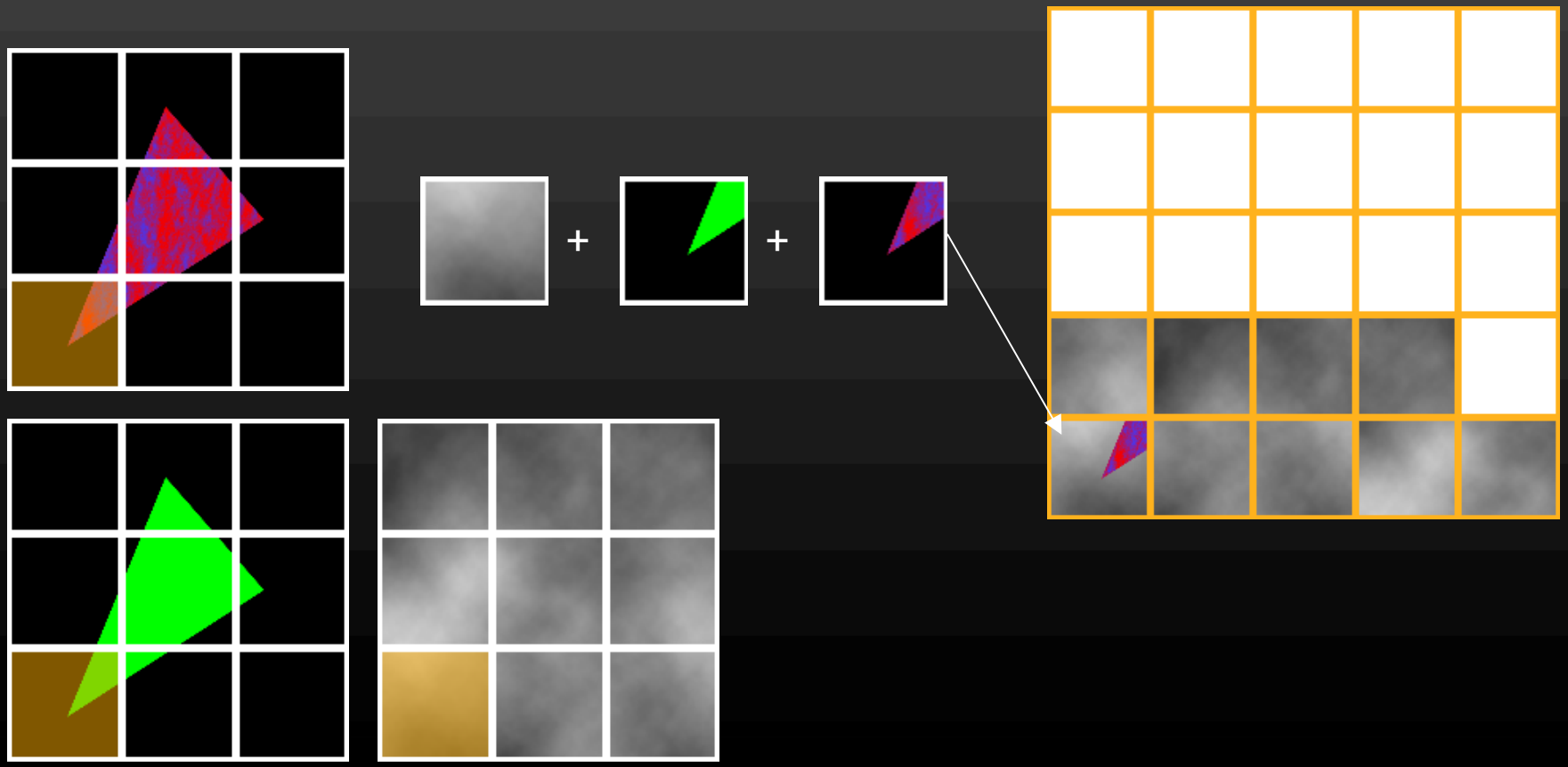
Pages to be written



Write Mask

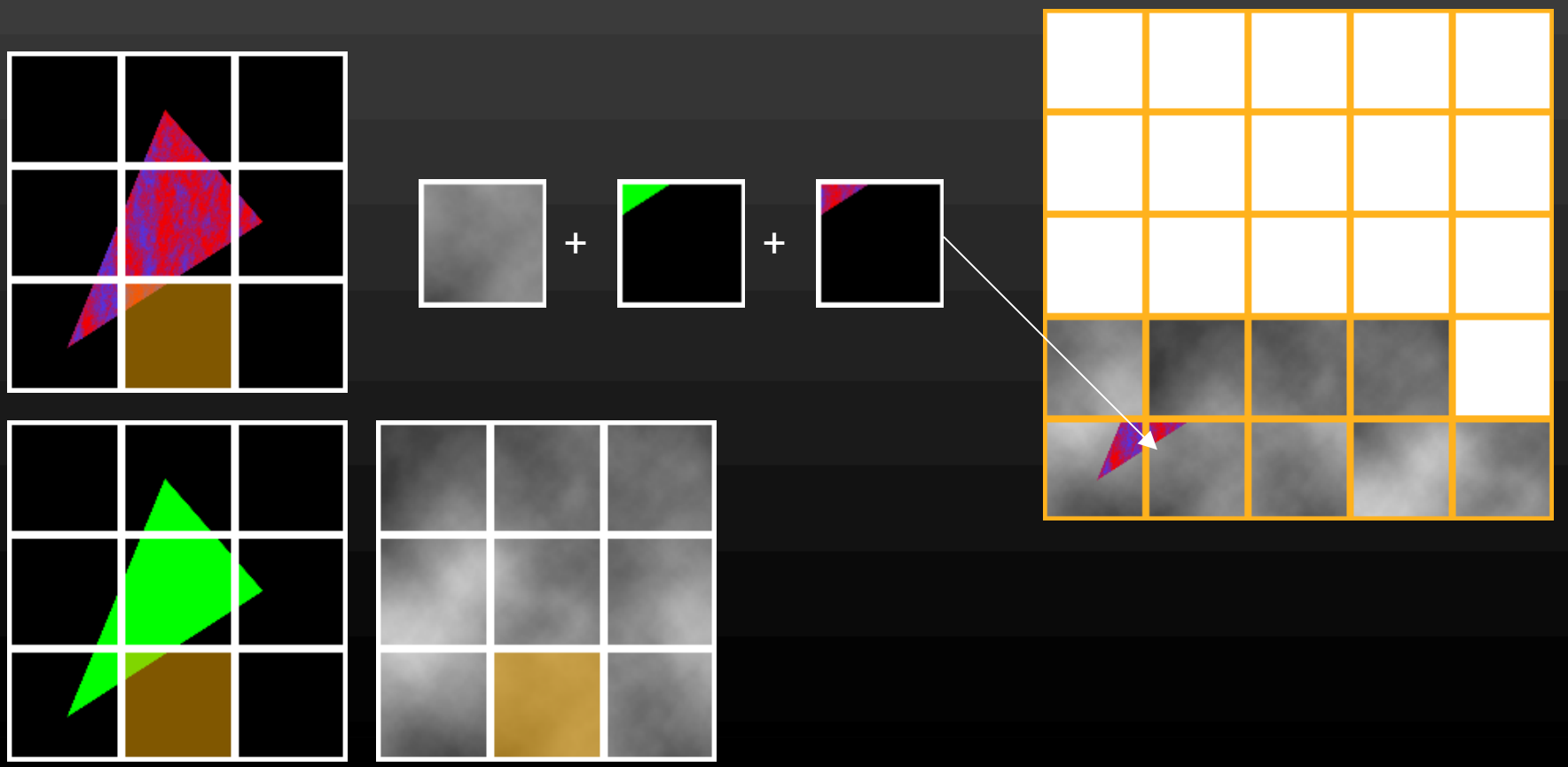
Texture Write

Pass 3: Copy modified pages into physical memory using write mask



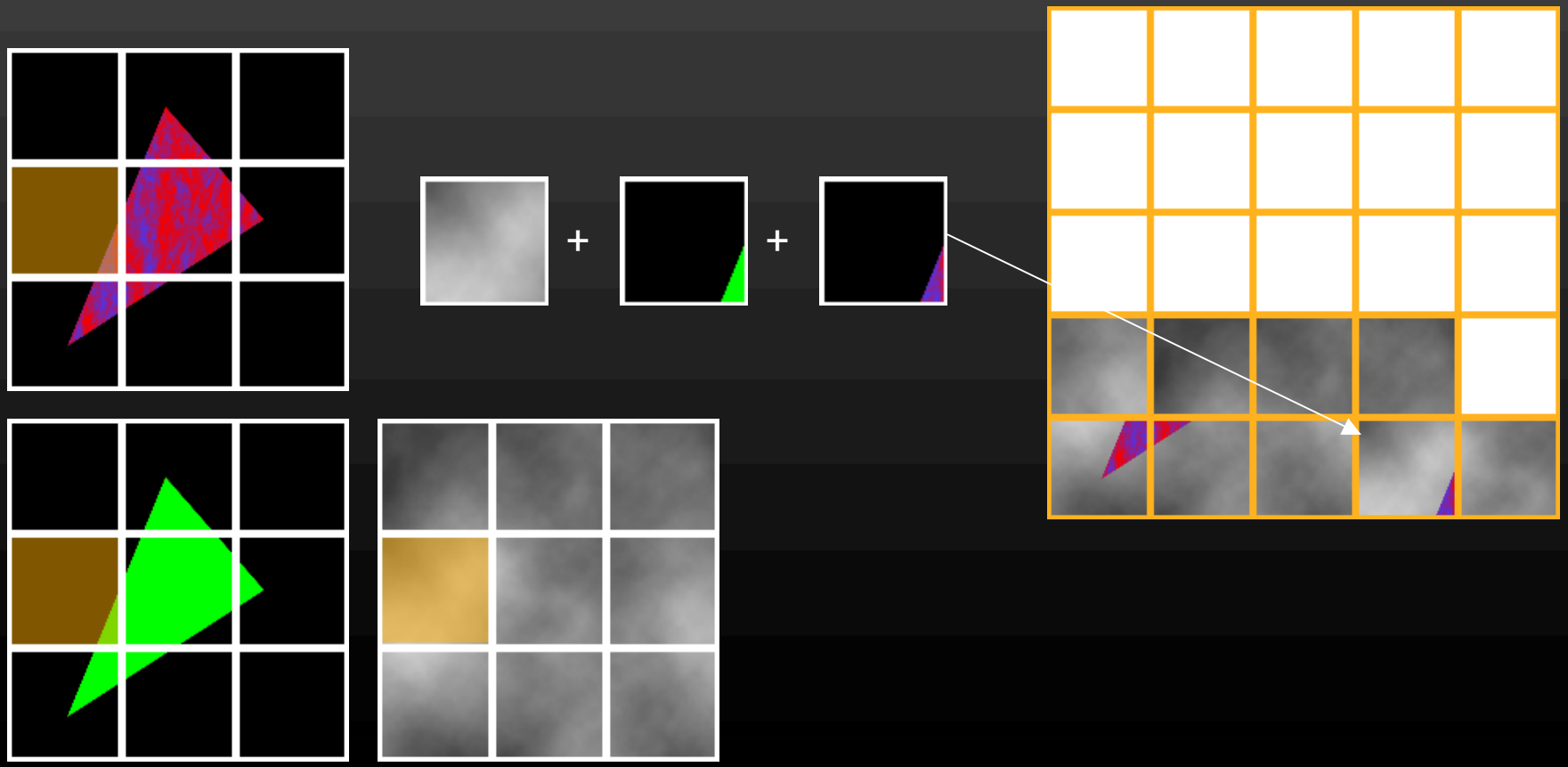
Texture Write

Stage 3: Copy modified pages into physical memory using write mask



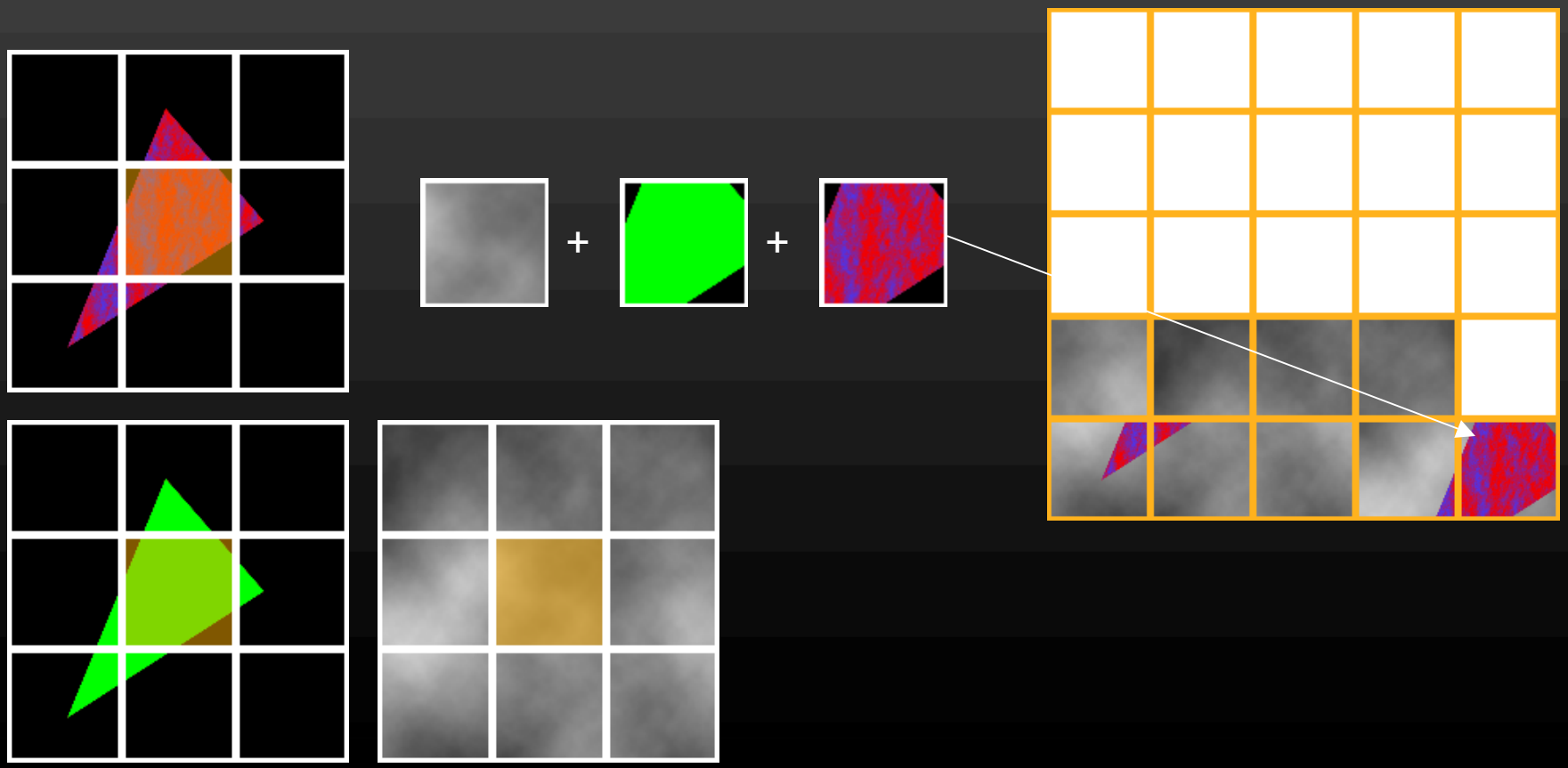
Texture Write

Stage 3: Copy modified pages into physical memory using write mask



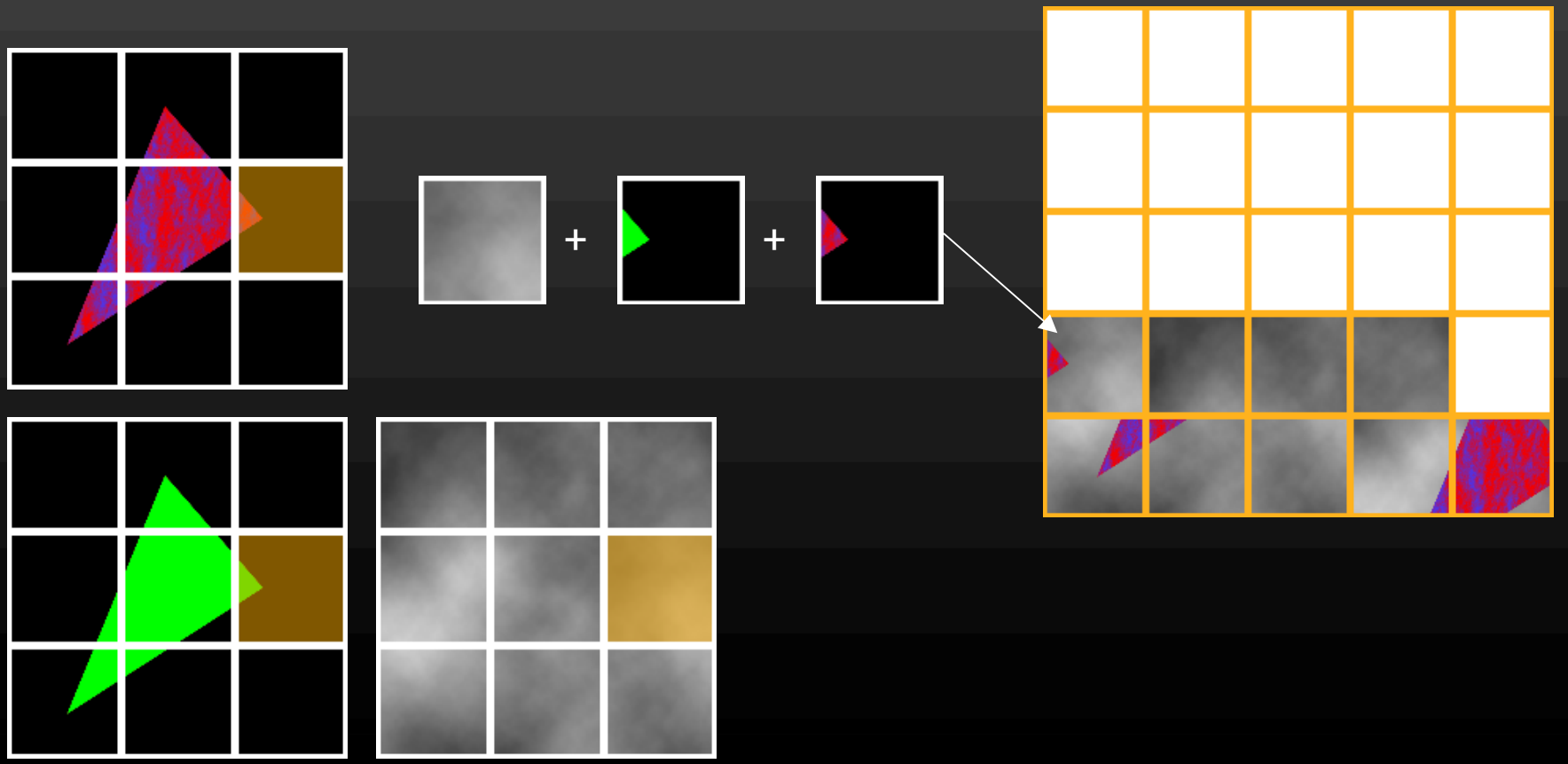
Texture Write

Stage 3: Copy modified pages into physical memory using write mask



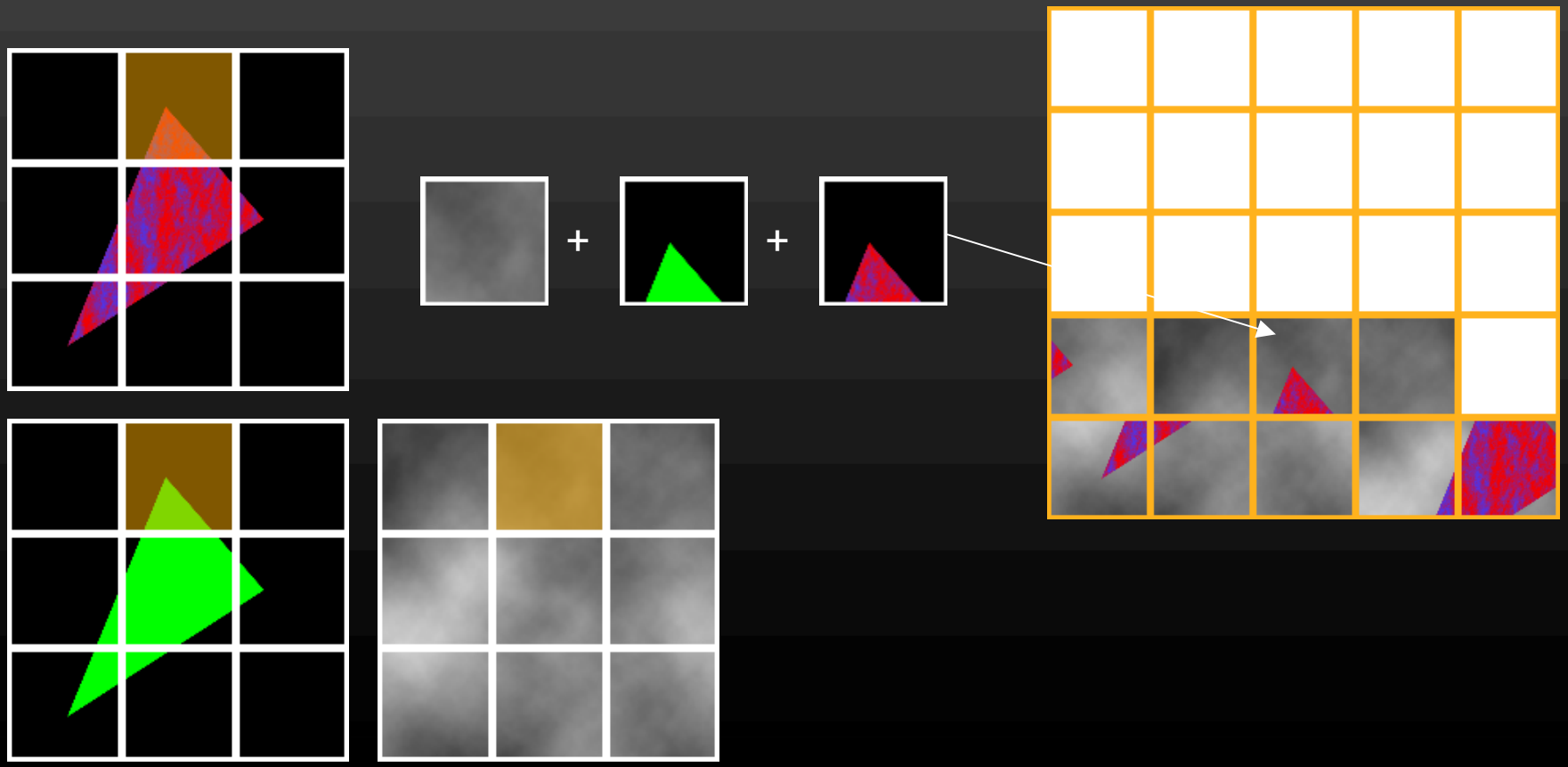
Texture Write

Stage 3: Copy modified pages into physical memory using write mask



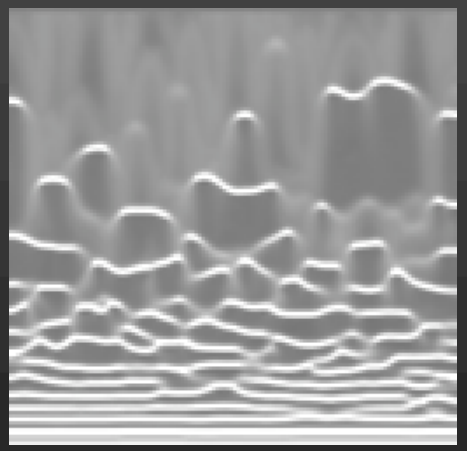
Texture Write

Stage 3: Copy modified pages into physical memory using write mask



Applications

- GPGPU - Boiling Application



- Game Trace - GL Quake

Limitations

- One Fragment per Pixel
  - Can only receive one fragment's texel requests
  - Example: Blending two fragment's requests makes no sense
- Solution: F-Buffer
  - Would allow each pixel to generate requests from every contributing fragment

Limitations

- Mipmapping
  - Cannot use hardware mipmapping
    - Mipmapping across pages makes no sense
    - Do 8 lookups by hand - inefficient
- Solutions
  1. Add border and mip-pyramid to each page
  2. Expose mipmapping hardware to programmer

Future Work

- Eviction Strategies
- Threading to minimize GPU idle time
- Optimizing for case when all textures local
- Advanced directory designs

Conclusion

- Goals
  - Scalable
  - Globally Addressable
  - Programmer Transparent
- Mechanisms
- Limitations

graphics hardware 06

Acknowledgements

Aaron Lefohn, Shubho Sengupta - UC Davis

Pat McCormick, Jeff Inman - LANL

Eric Demers, Bob Drebin - ATI

Henry Moreton - NVIDIA

Mike Houston - Stanford University