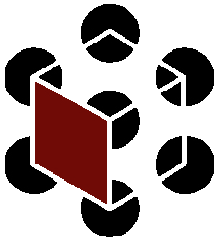


GPU Memory Model Overview

Aaron Lefohn

University of California, Davis

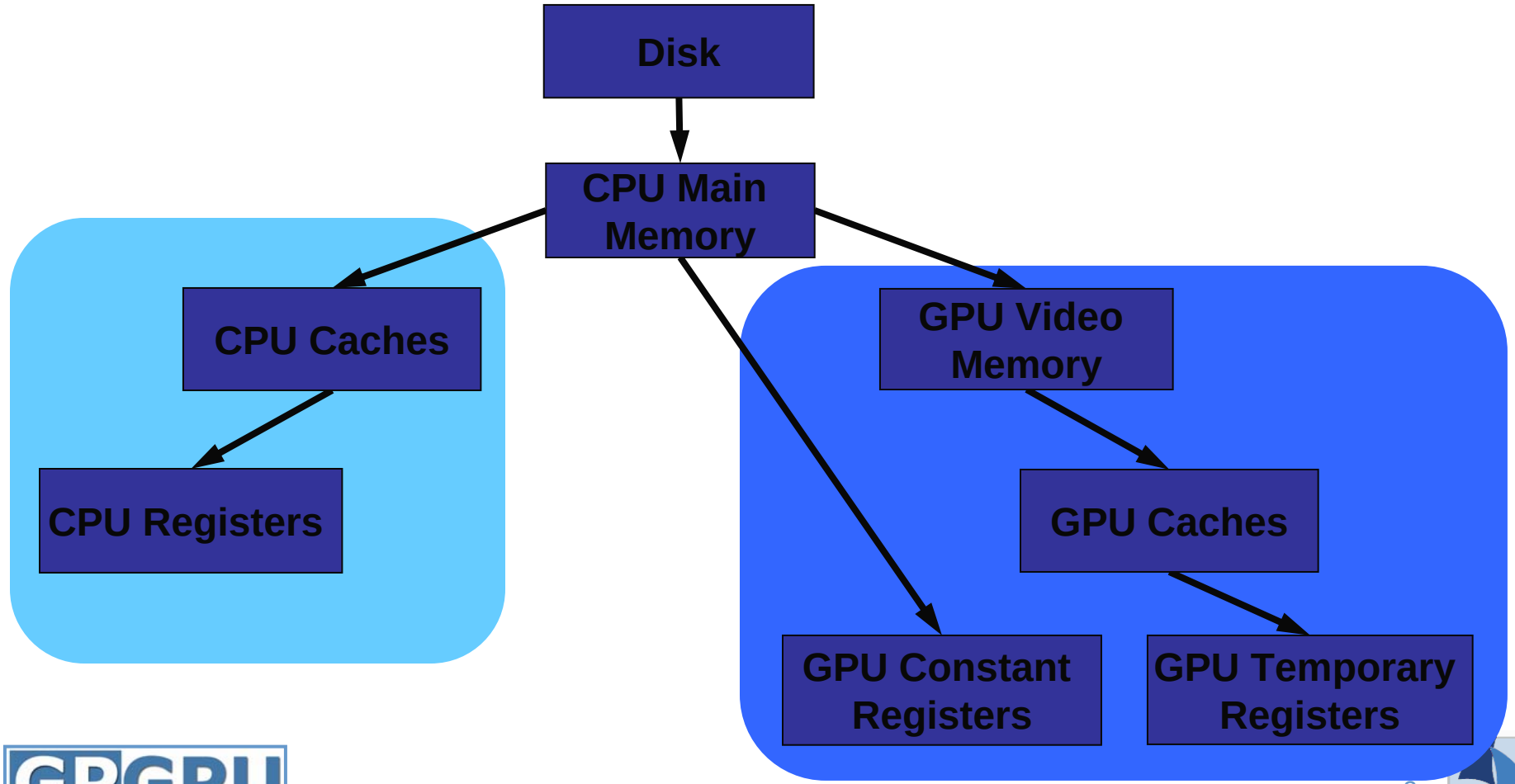
GPGPU

Overview

- **GPU Memory Model**
- GPU Data Structure Basics
- Introduction to Framebuffer Objects

Memory Hierarchy

- CPU and GPU Memory Hierarchy



CPU Memory Model

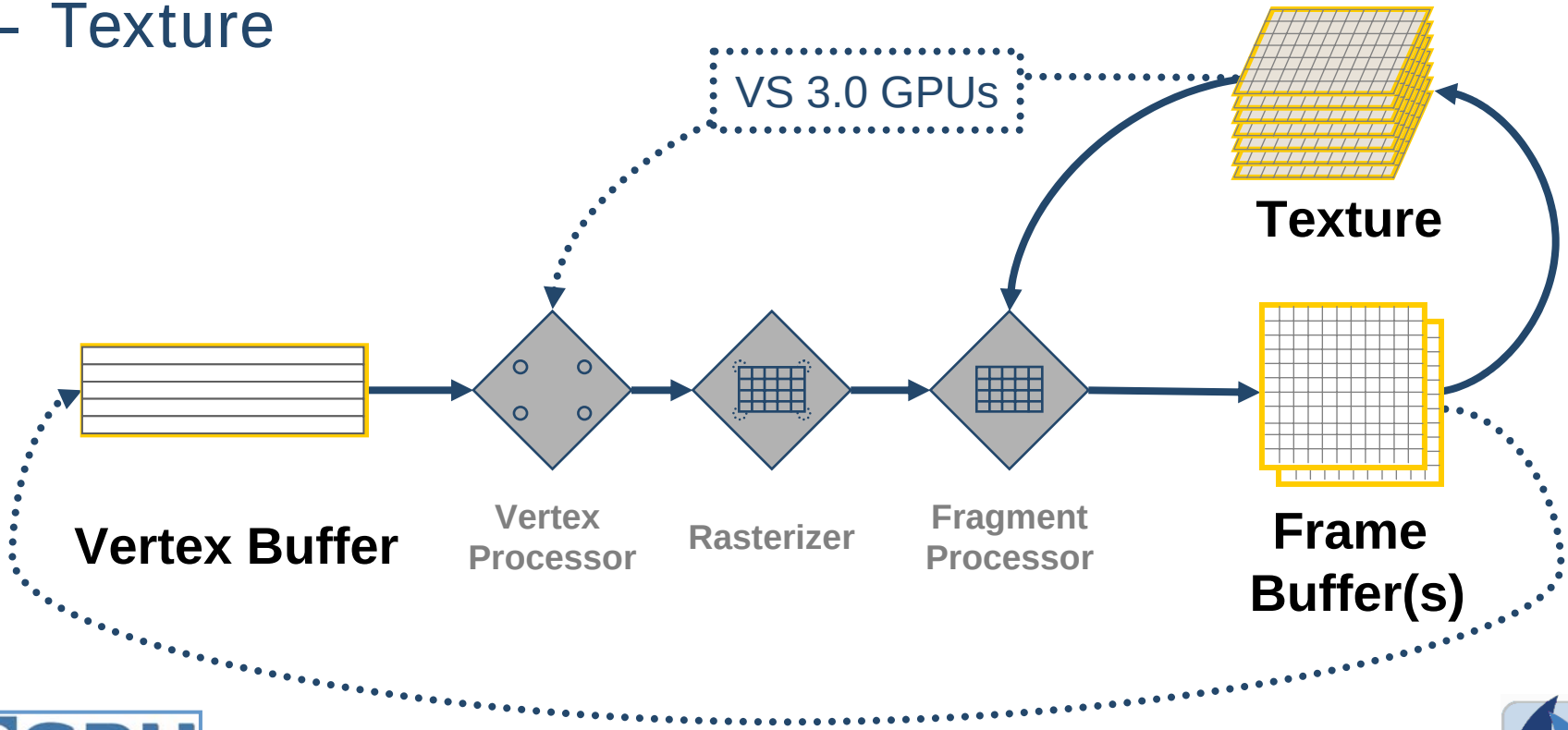
- **At any program point**
  - Allocate/free local or global memory
  - Random memory access
    - Registers
      - Read/write
    - Local memory
      - Read/write to stack
    - Global memory
      - Read/write to heap
    - Disk
      - Read/write to disk

GPU Memory Model

- **Much more restricted memory access**
  - Allocate/free memory only before computation
  - Limited memory access during computation (kernel)
    - Registers
      - Read/write
    - Local memory
      - Does not exist
    - Global memory
      - Read-only during computation
      - Write-only at end of computation (precomputed address)
    - Disk access
      - Does not exist

GPU Memory Model

- **Where is GPU Data Stored?**
  - Vertex buffer
  - Frame buffer
  - Texture



GPU Memory API

- **Each GPU memory type supports subset of the following operations**
  - CPU interface
  - GPU interface

GPU Memory API

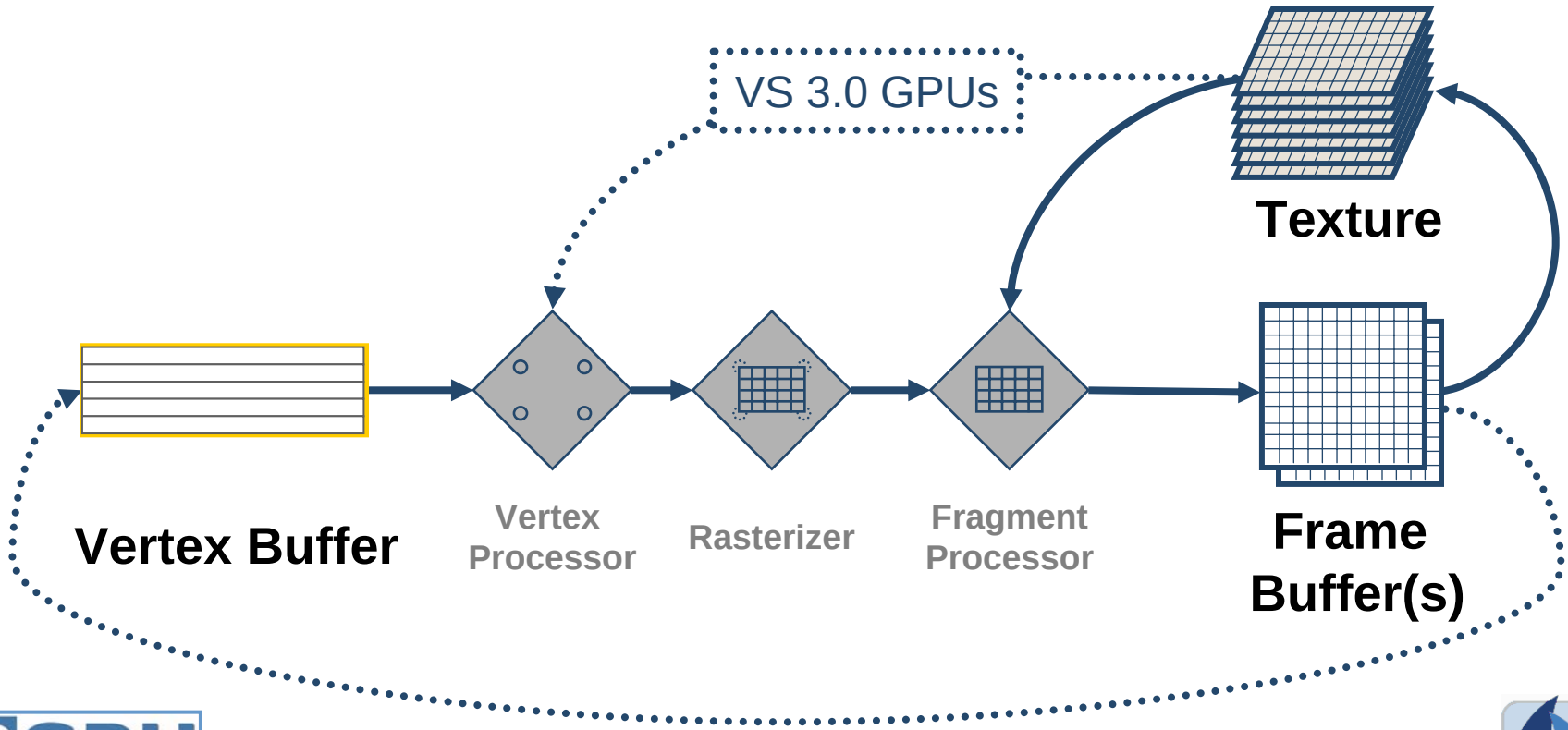
- **CPU interface**
  - Allocate
  - Free
  - Copy CPU → GPU
  - Copy GPU → CPU
  - Copy GPU → GPU
  - Bind for read-only vertex stream access
  - Bind for read-only random access
  - Bind for write-only framebuffer access

GPU Memory API

- **GPU (shader/kernel) interface**
  - Random-access read
  - Stream read

Vertex Buffers

- GPU memory for vertex data
- Vertex data required to initiate render pass



Vertex Buffers

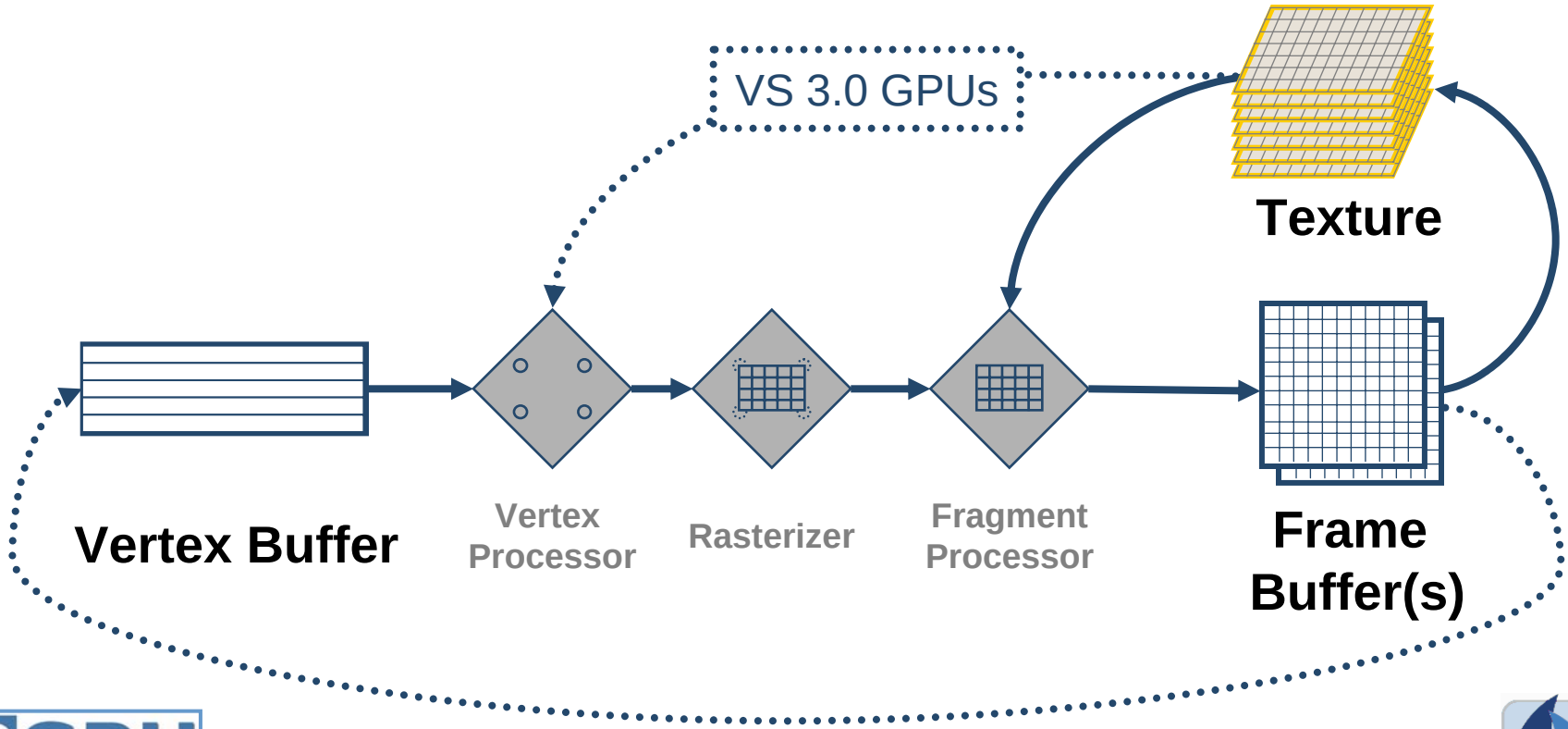
- **Supported Operations**
  - CPU interface
    - Allocate
    - Free
    - Copy CPU → GPU
    - Copy GPU → GPU (Render-to-vertex-array)
    - Bind for read-only vertex stream access
  - GPU interface
    - Stream read (vertex program only)

Vertex Buffers

- **Limitations**
  - CPU
    - No copy GPU → CPU
    - No bind for read-only random access
    - No bind for write-only framebuffer access
      - ATI supported this in uberbuffers. Perhaps we'll see this as an OpenGL extension?
  - GPU
    - No random-access reads
    - No access from fragment programs

Textures

- Random-access GPU memory



Textures

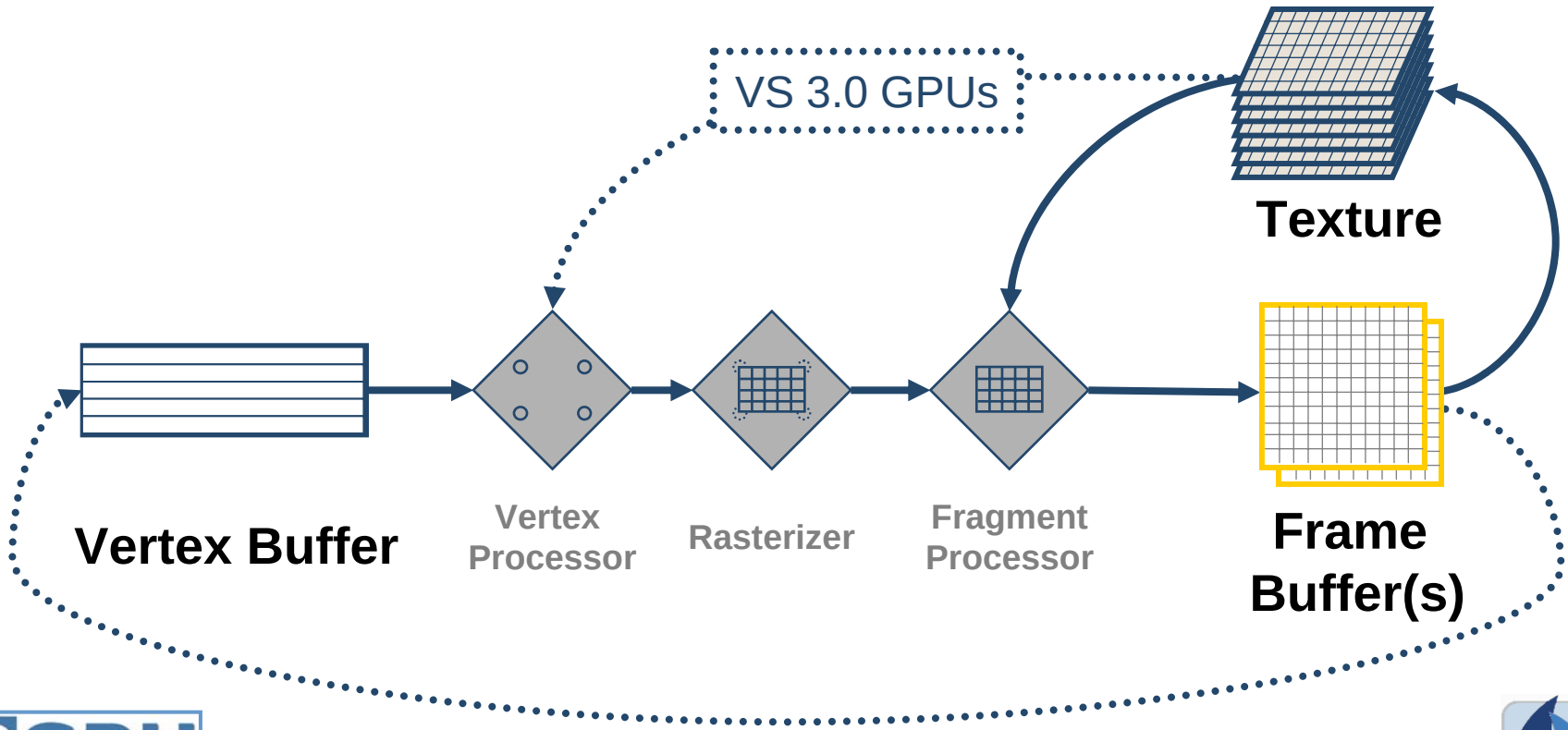
- **Supported Operations**
  - CPU interface
    - Allocate
    - Free
    - Copy CPU → GPU
    - Copy GPU → CPU
    - Copy GPU → GPU (Render-to-texture)
    - Bind for read-only random access (vertex or fragment)
    - Bind for write-only framebuffer access
  - GPU interface
    - Random read

Textures

- **Limitations**
  - No bind for vertex stream access

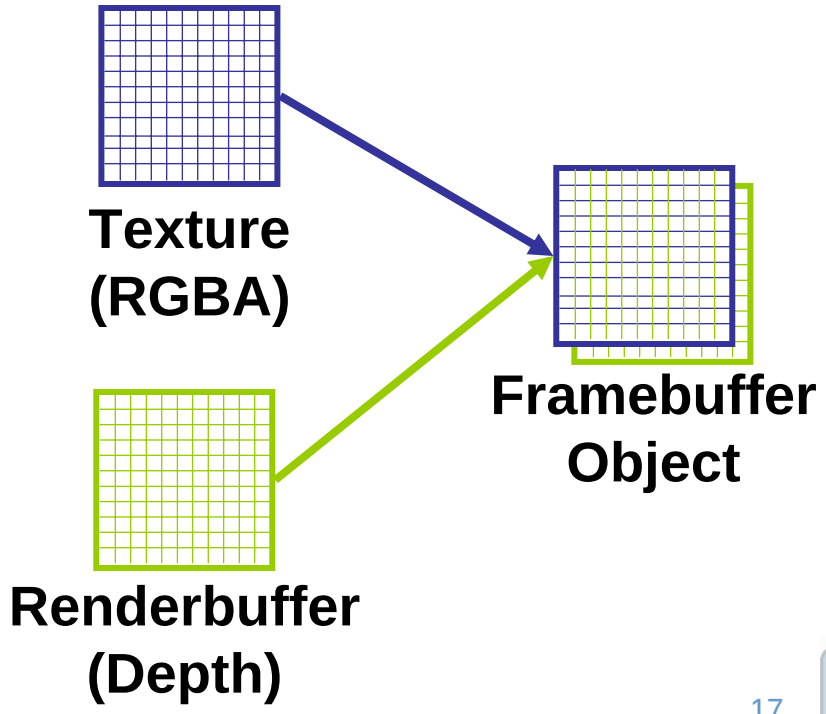
Framebuffer

- Memory written by fragment processor
- Write-only GPU memory



OpenGL Framebuffer Objects

- **General idea**
  - Framebuffer object is lightweight struct of pointers
  - Bind GPU memory to framebuffer as write-only
  - Memory cannot be read while bound to framebuffer
- **Which memory?**
  - Texture
  - Renderbuffer
  - Vertex buffer??



Framebuffer Object

- **New OpenGL extension**
  - Enables true render-to-texture in OpenGL
  - Mix-and-match depth/stencil buffers
  - Replaces pbuffers!
  - More details coming later in talk…

http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt

What is a Renderbuffer?

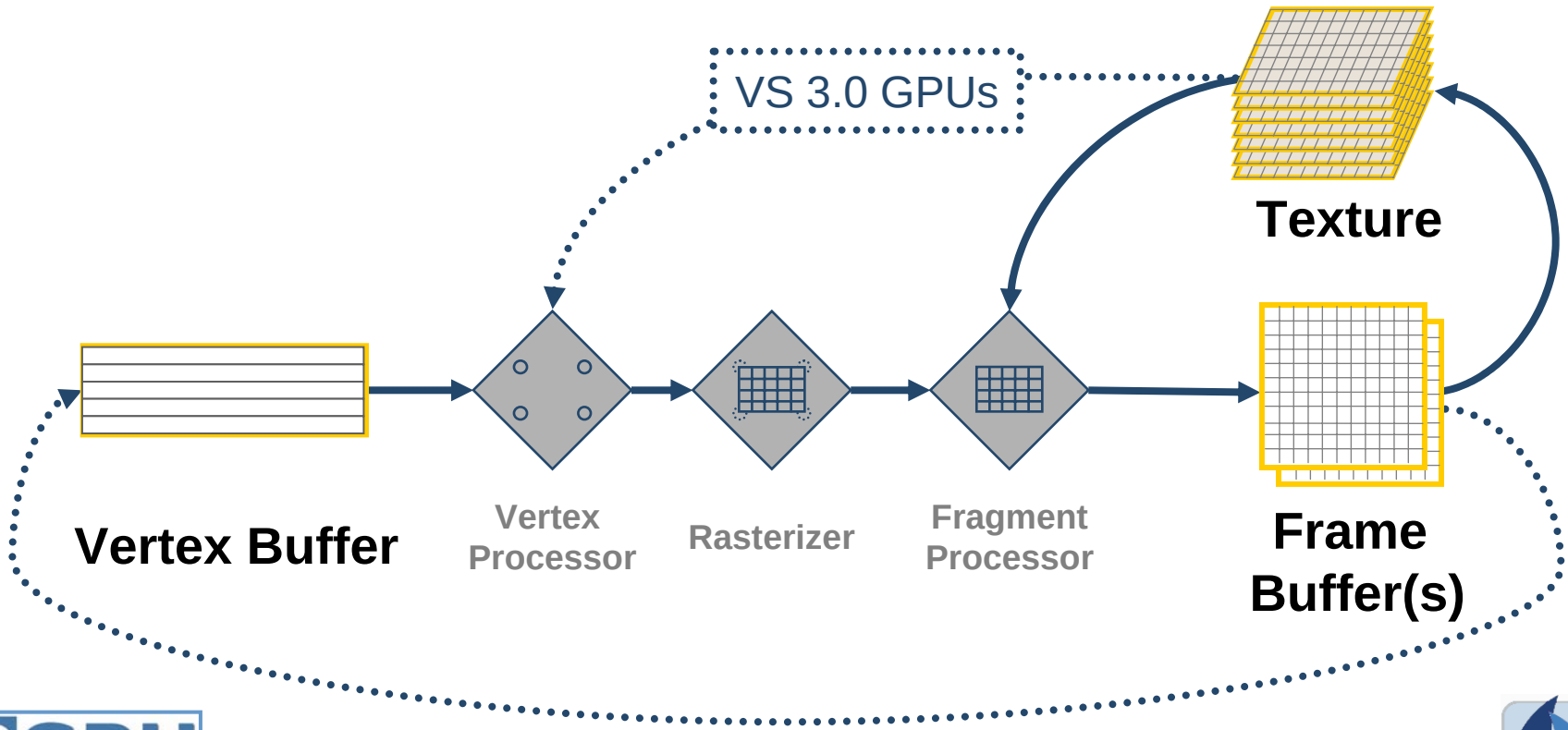
- **“Traditional” framebuffer memory**
  - Write-only GPU memory
    - Color buffer
    - Depth buffer
    - Stencil buffer
- **New OpenGL memory object**
  - Part of Framebuffer Object extension

Renderbuffer

- **Supported Operations**
  - CPU interface
    - Allocate
    - Free
    - Copy GPU → CPU
    - Bind for write-only framebuffer access

Pixel Buffer Objects

- **Mechanism to efficiently transfer pixel data**
  - API nearly identical to vertex buffer objects



Pixel Buffer Objects

- **Uses**
  - Render-to-vertex-array
    - glReadPixels into GPU-based pixel buffer
    - Use pixel buffer as vertex buffer
  - Fast streaming textures
    - Map PBO into CPU memory space
    - Write directly to PBO
    - Reduces one or more copies

Pixel Buffer Objects

- **Uses (continued)**
  - Asynchronous readback
    - Non-blocking GPU → CPU data copy
    - glReadPixels into PBO does *not* block
    - Blocks when PBO is mapped into CPU memory

Summary : Render-to-Texture

- **Basic operation in GPGPU apps**

- **OpenGL Support**
  - Save up to 16, 32-bit floating values per pixel
    - Multiple Render Targets (MRTs) on ATI and NVIDIA

  1. Copy-to-texture
     - glCopyTexSubImage

     Render-to-texture
     - GL_EXT_framebuffer_object

Summary : Render-To-Vertex-Array

- **Enable top-of-pipe feedback loop**
- **OpenGL Support**
  - Copy-to-vertex-array
    - GL_ARB_pixel_buffer_object
    - NVIDIA and ATI
  - Render-to-vertex-array
    - Maybe future extension to framebuffer objects

Overview

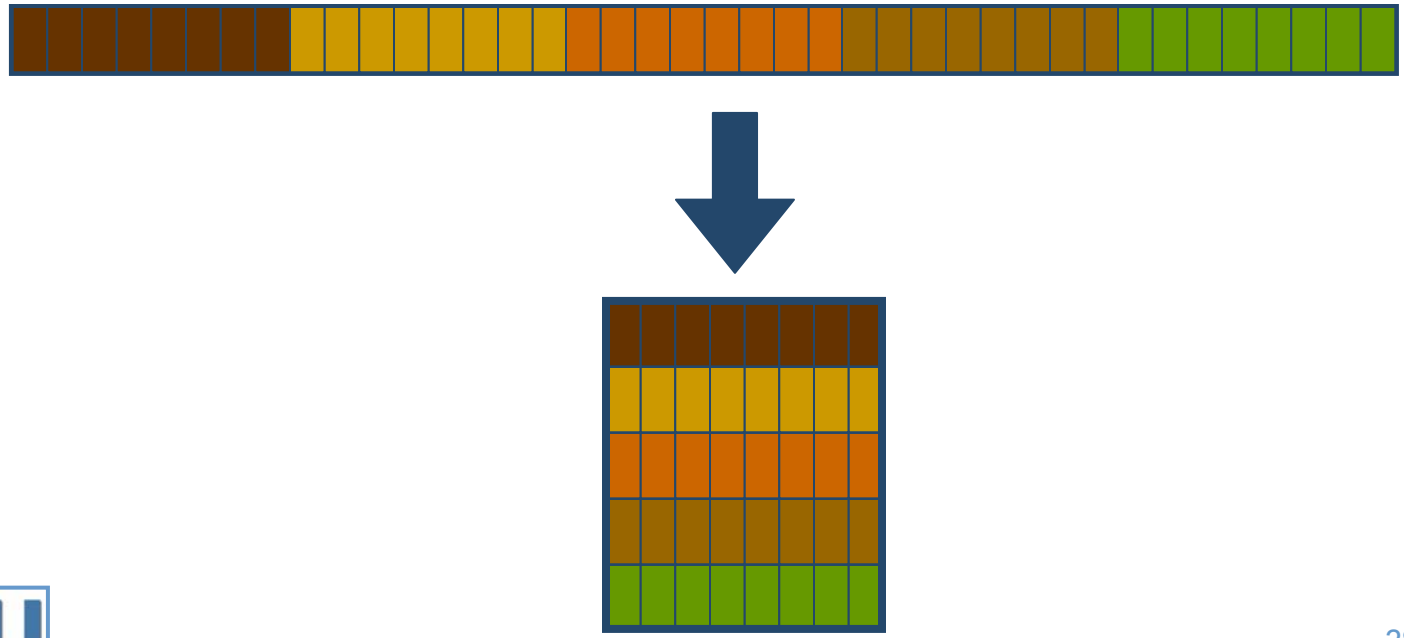
- GPU Memory Model
- **GPU Data Structure Basics**
- Introduction to Framebuffer Objects

GPU Data Structure Basics

- **Summary of "Implementing Efficient Parallel Data Structures on GPUs"**
  - Chapter 33, GPU Gems II
- **Low-level details**
  - The "Glift" talk described high-level view of GPU data structures
- **Now for the gory details…**

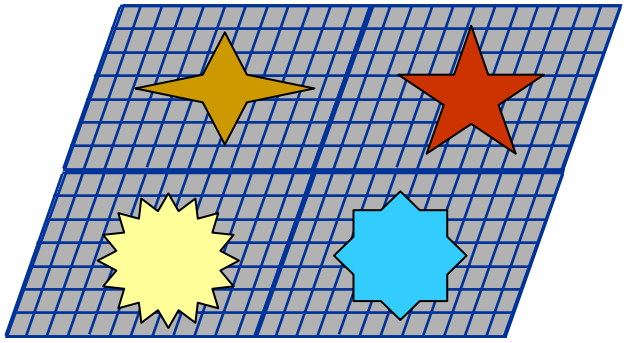
GPU Arrays

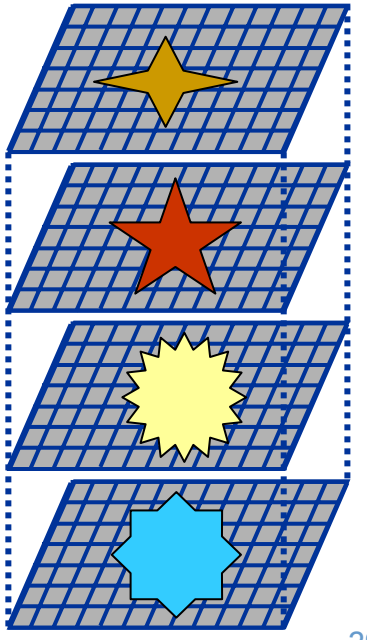
- **Large 1D Arrays**
  - Current GPUs limit 1D array sizes to 2048 or 4096
  - Pack into 2D memory
  - 1D-to-2D address translation



GPU Arrays

- **3D Arrays**
  - Problem
    - GPUs do not have 3D frame buffers
    - No render-to-slice-of-3D-texture yet (coming soon?)
  - Solutions
    1. Stack of 2D slices
    2. Multiple slices per 2D buffer





GPU Arrays

- **Problems With 3D Arrays for GPGPU**
  - Cannot read stack of 2D slices as 3D texture
  - Must know which slices are needed in advance
  - Visualization of 3D data difficult

- **Solutions**
  - Flat 3D textures
  - Need render-to-slice-of-3D-texture
    - Maybe with GL_EXT_framebuffer_object
  - Volume rendering of flattened 3D data
    - "Deferred Filtering: Rendering from Difficult Data Formats," GPU Gems 2, Ch. 41, p. 667

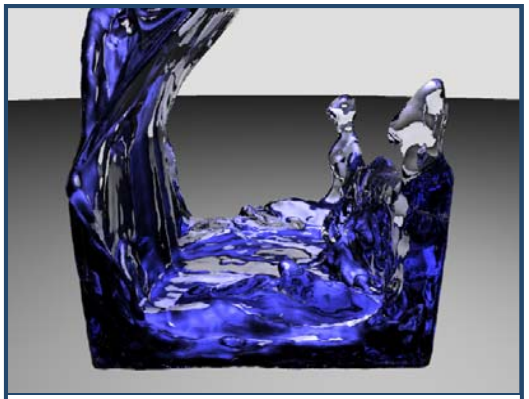
GPU Arrays

- **Higher Dimensional Arrays**
  - Pack into 2D buffers
  - N-D to 2D address translation
  - Same problems as 3D arrays if data does not fit in a single 2D texture

Sparse/Adaptive Data Structures

- **Why?**
  - Reduce memory pressure
  - Reduce computational workload

- **Examples**
  - Sparse matrices
    - Krueger et al., Siggraph 2003
    - Bolz et al., Siggraph 2003



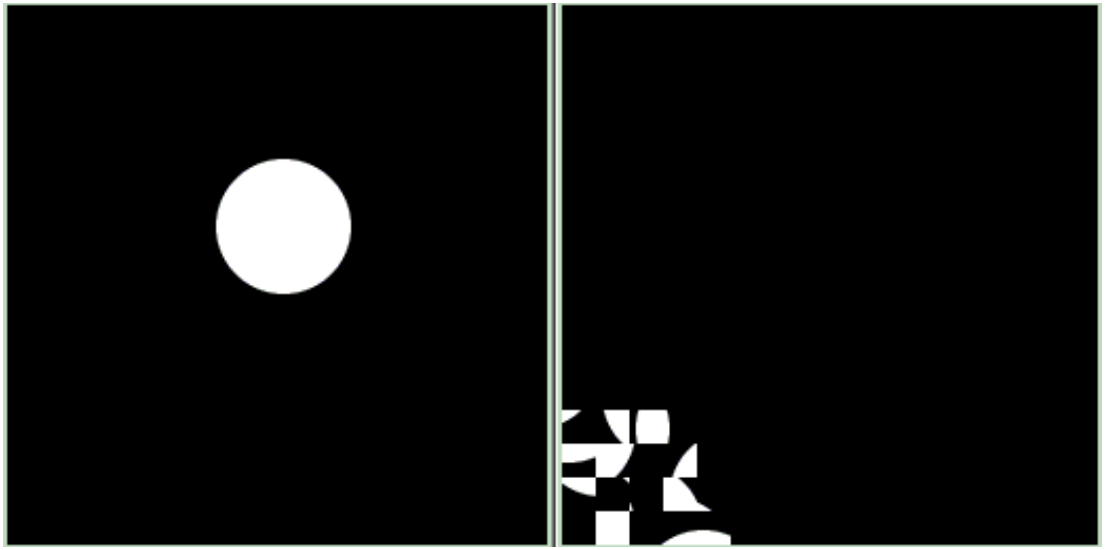
Premoze et al. Eurographics 2003

  - Deformable implicit surfaces (sparse volumes/PDEs)
    - Lefohn et al., IEEE Visualization 2003 / TVCG 2004

  - Adaptive radiosity solution (Coombe et al.)

Sparse/Adaptive Data Structures

- **Basic Idea**
  - Pack “active” data elements into GPU memory
  - For more information
    - Owens et al., GPGPU Eurographics 2005 STAR Report
    - Lefohn et al., “Glift : Generic, Efficient, Random-Access GPU Data Structures,” Transactions on Graphics, To Appear, 2005



GPU Data Structures

- **Conclusions**
  - Fundamental GPU memory primitive is a fixed-size 2D array
  - GPGPU needs more general memory model
  - Building and modifying complex GPU-based data structures is an open research topic…

Overview

- GPU Memory Model
- GPU-Based Data Structures
- **Introduction to Framebuffer Objects**

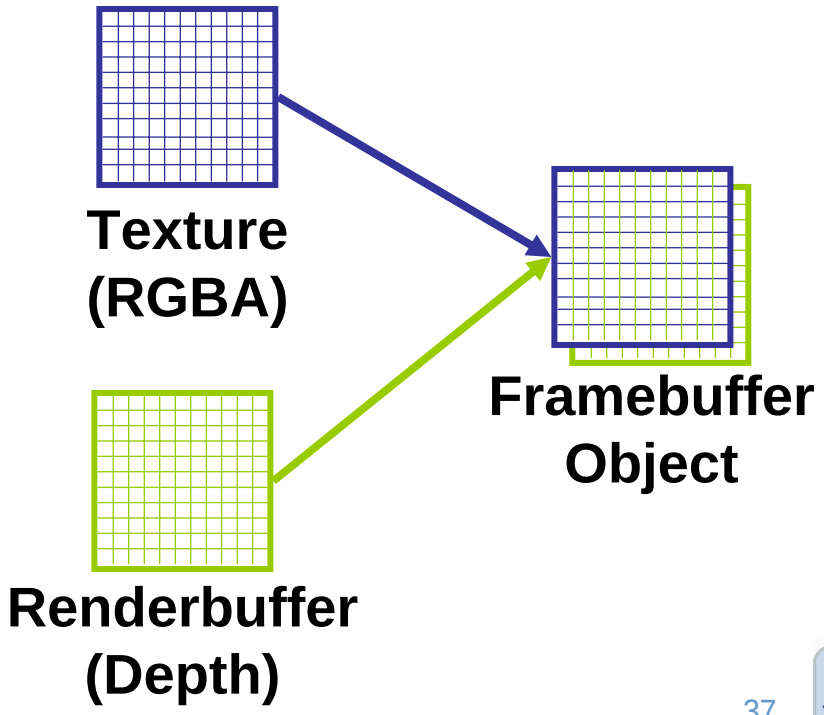
Introduction to Framebuffer Objects

- **Where is the “Pbuffer Survival Guide”?**
  - Gone!!!
  - Framebuffer objects replace pbuffers
  - Simple, intuitive, fast render-to-texture in OpenGL

http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt

Framebuffer Objects

- **What is an FBO?**
  - A struct that holds pointers to memory objects
  - Each bound memory object can be a framebuffer rendering surface
  - Platform-independent



Framebuffer Objects

- **Which memory can be bound to an FBO?**
  - Textures
  - Renderbuffers
    - Depth, stencil, color
    - Traditional write-only framebuffer surfaces

Framebuffer Objects

- **Usage models**
  - Keep N textures bound to one FBO (up to 16)
    - Change render targets with glDrawBuffers
  - Keep one FBO for each size/format
    - Change render targets with attach/unattach textures
  - Keep several FBOs with textures attached
    - Change render targets by binding FBO

Framebuffer Objects

- **Performance**
  - Render-to-texture
    - glDrawBuffers is fastest on NVIDIA/ATI
      - As-fast or faster than pbuffers
    - Attach/unattach textures same as changing FBOs
      - Slightly slower than glDrawBuffers but faster than wglMakeCurrent
    - Keep format/size identical for all attached memory
      - Current driver limitation, not part of spec
  - Readback
    - Same as pbuffers for NVIDIA and ATI

Framebuffer Objects

- **Driver support still evolving**
  - GPUBench FBO tests coming soon…
  - fbocheck.exe evalulates completeness

Framebuffer Object

- **Code examples**
  - Simple C++ FBO and Renderbuffer classes
    - HelloWorld example
    - http://gpgpu.sourceforge.net/
  - OpenGL Spec

    http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt

Conclusions

- **GPU Memory Model Evolving**
  - Writable GPU memory forms loop-back in an otherwise feed-forward streaming pipeline
  - Memory model will continue to evolve as GPUs become more general data-parallel processors

- **GPGPU Data Structures**
  - Basic memory primitive is limited-size, 2D texture
  - Use address translation to fit all array dimensions into 2D
  - See “Glift” talk for higher-level GPU data structures

Acknowledgements

- Adam Moerschell, Shubho Sengupta, UCDavis
- Mike Houston, Stanford University

- Nick Triantos, Craig Kolb, Cass Everitt, Chris Seitz, NVIDIA
- Mark Segal, Rob Mace, Arcot Preetham, Evan Hart, ATI

- John Owens, Ph.D. advisor, Univ. of California Davis
- National Science Foundation Graduate Fellowship

Questions?

- **Thank you!**
- **Google “Lefohn GPU”**
  - http://graphics.cs.ucdavis.edu/~lefohn/