

---

# GPGPU Memory Model

---



Aaron Lefohn

Institute for Data Analysis and Visualization  
University of California, Davis

# Overview

---

- GPU Memory Model
- GPU-Based Data Structures
- Pbuffer Survival Guide



# CPU Memory Model

---

- Random Memory Access at Any Program Point
  - Read/write to registers
  - Read/write to local (stack) memory
  - Read/write to global (heap) memory
  - Read/write to disk



# GPU Memory Model

---

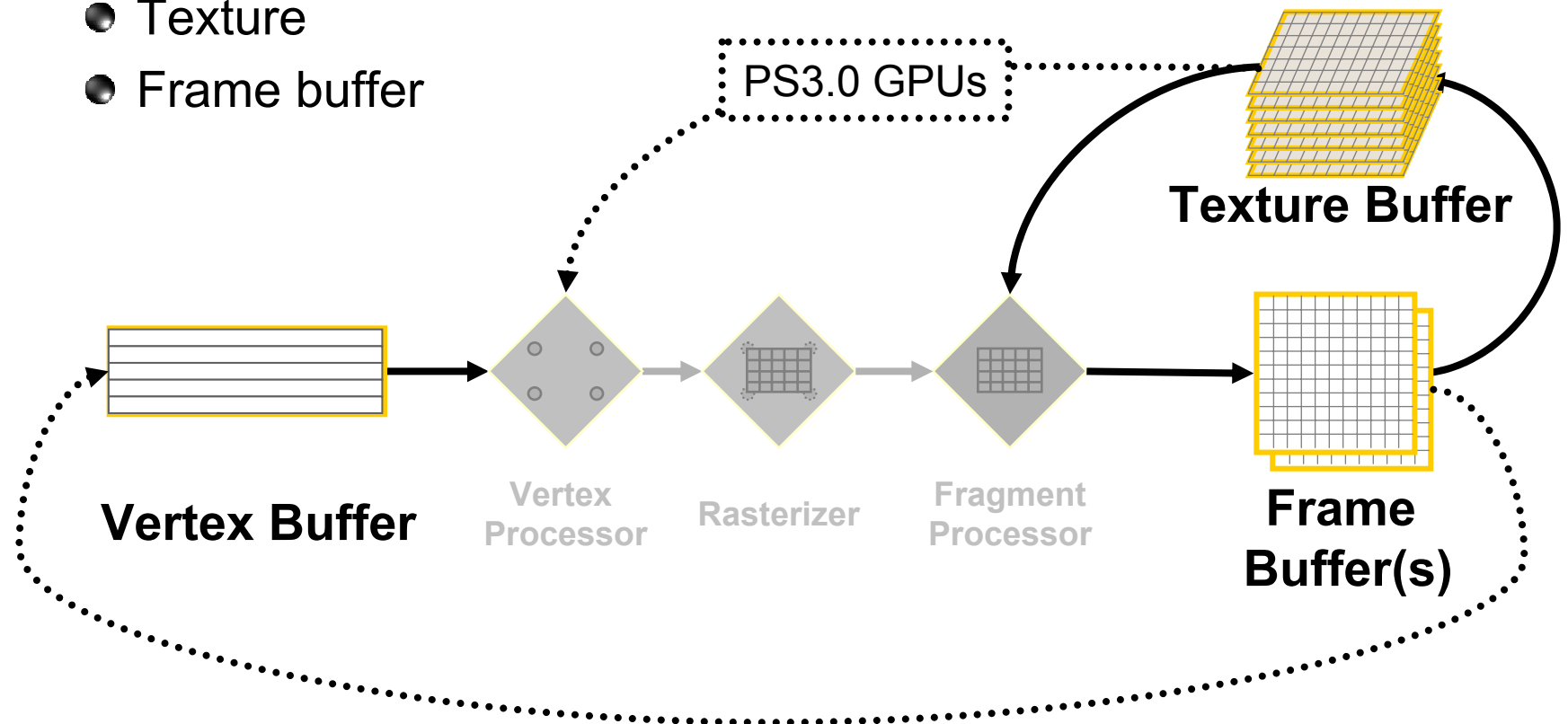
- Much more restricted memory access
  - GPU Kernels
    - Read/write to registers
    - No local stack memory
    - No disk access
    - Read-only global memory access
      - `v = a[i];`
  - Write to global memory at end of pass
    - Pre-computed memory addresses (no scatter)
      - `i = foo(a);`  
`a[i] = bar(b);`
    - Write location set by fragment position
      - `a[fragPos] = bar(b);`



# GPU Memory Model

- Where is GPU Data Stored?

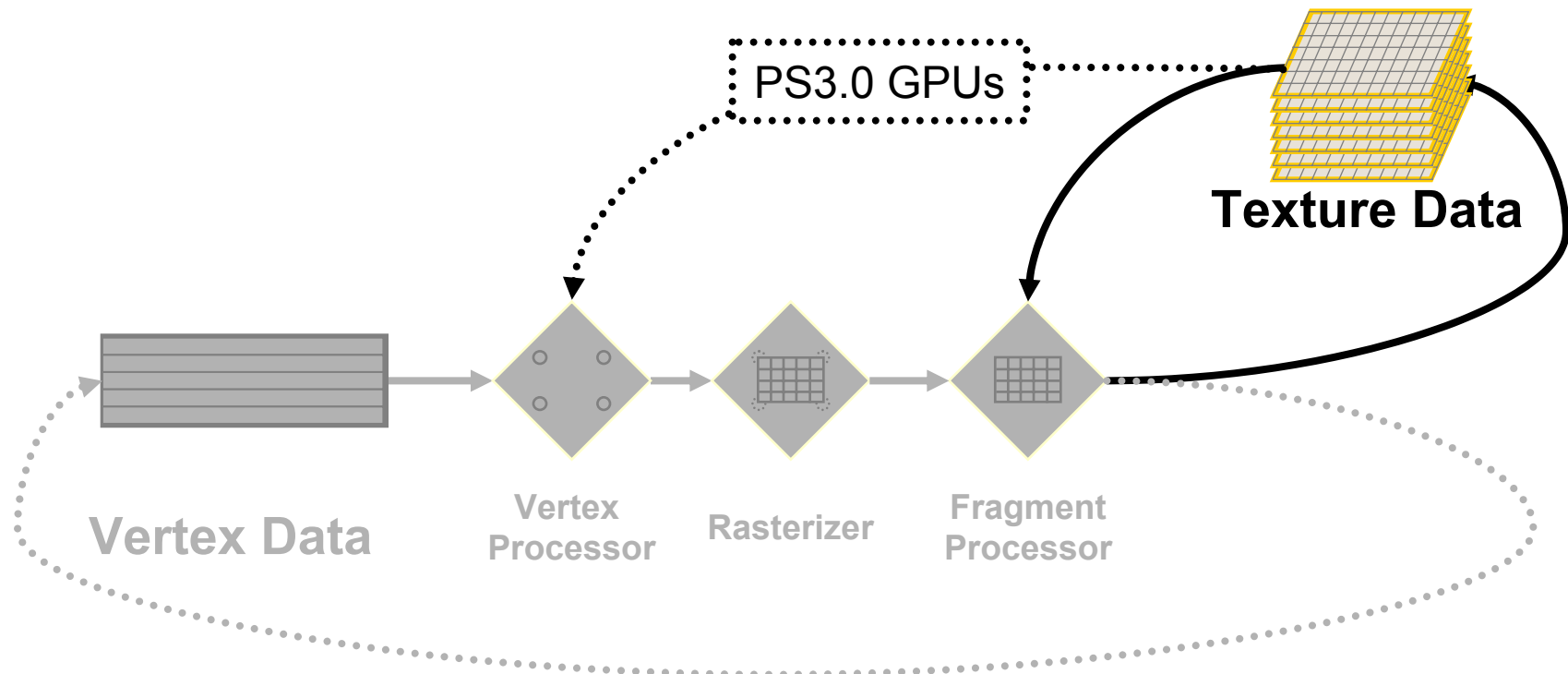
- Vertex buffer
- Texture
- Frame buffer



# Render-to-Texture

- Idea

- Write rendering result to texture memory
- Enables GPU-based computational iterations



# Render-to-Texture

---

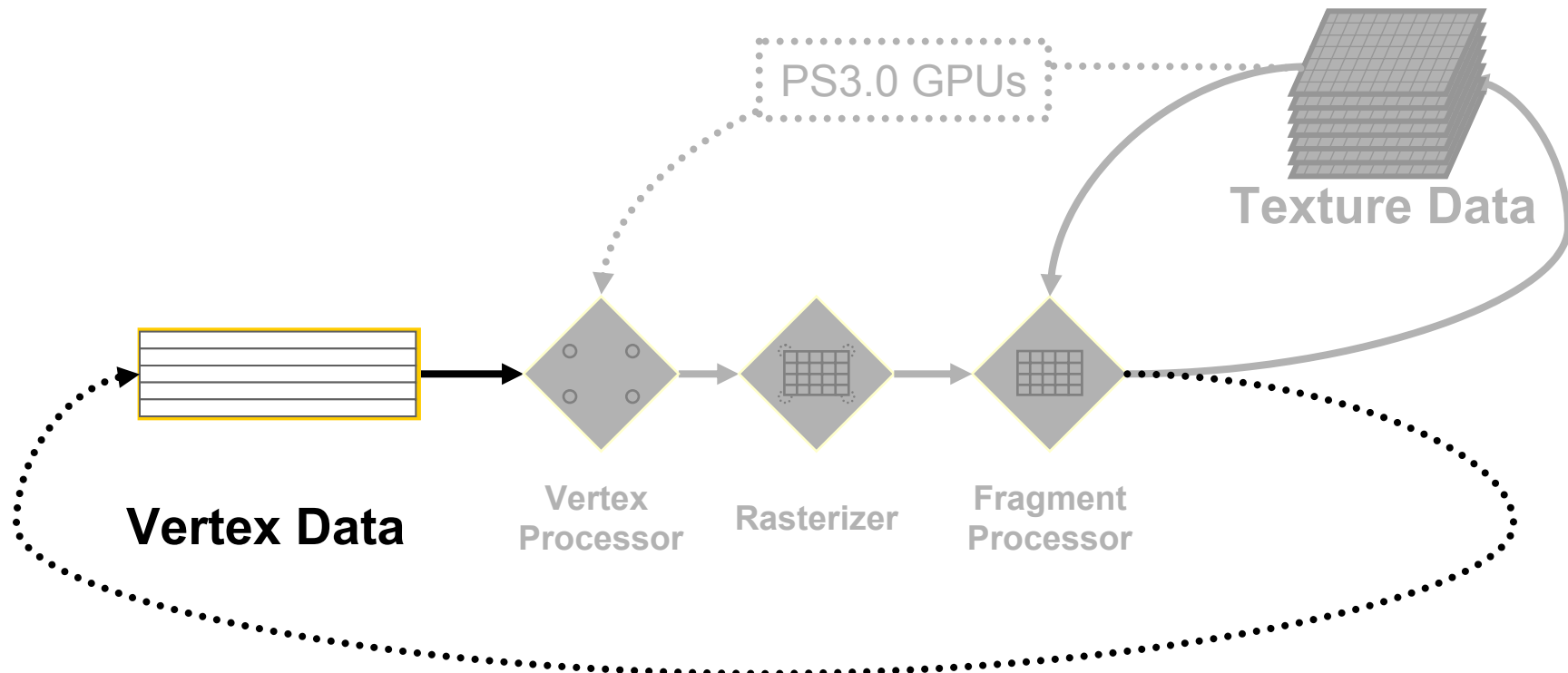
- OpenGL Support
  - Save up to 16, 32-bit floating values per pixel
    - Multiple Render Targets (MRTs) on ATI and NVIDIA
- 1. Copy-to-texture
  - glCopyTexSubImage
- 2. Render-to-texture
  - WGL\_ARB\_render\_texture
    - Pbuffers: Current state of the art
  - GL\_EXT\_render\_target
    - Proposed extension
  - Superbuffers
    - Proposed extension



# Render-to-Vertex-Array

- Idea

- Write rendering results to vertex array
- Allows GPU to loop back to beginning of pipeline



# Render-To-Vertex-Array

---

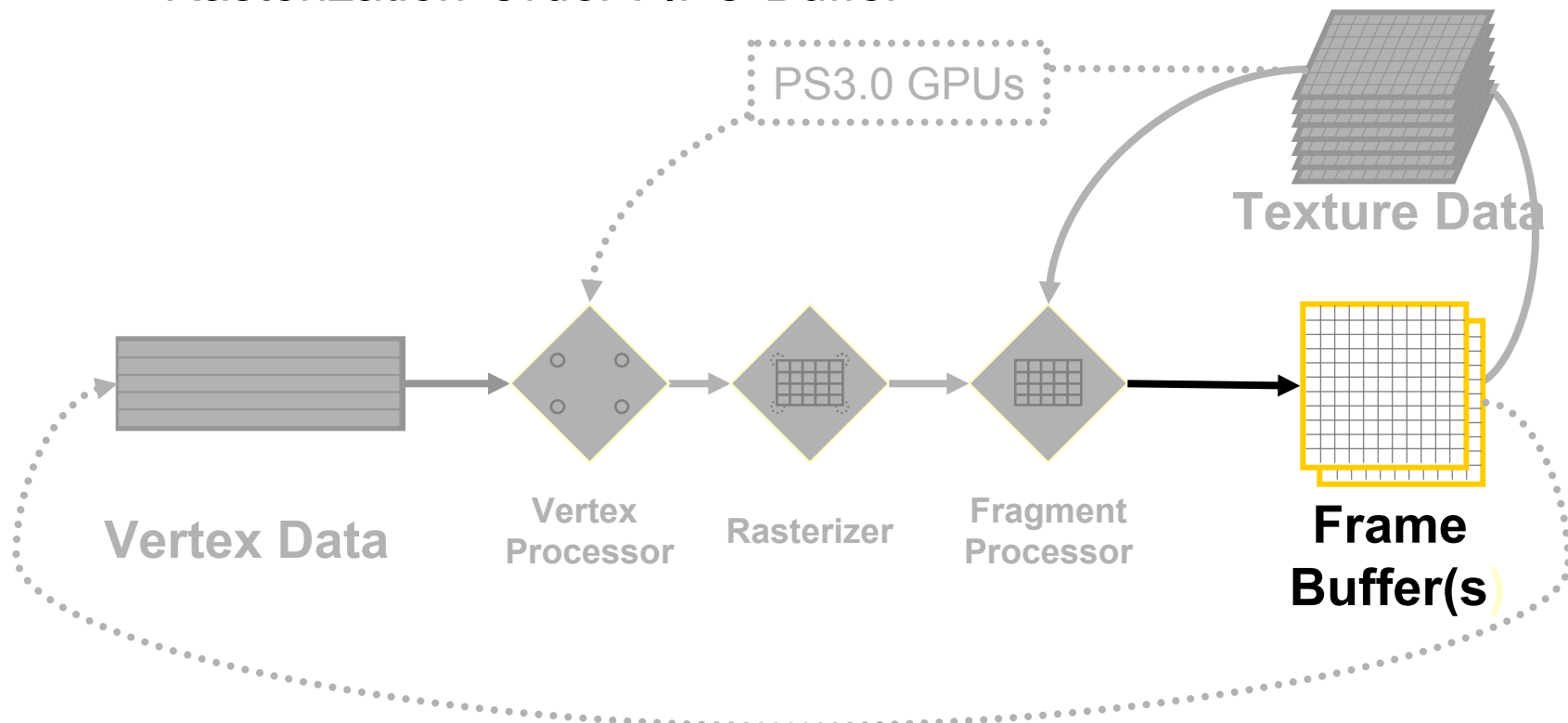
- OpenGL Support
  - Copy-to-vertex-array
    - GL\_EXT\_pixel\_buffer\_object
    - NVIDIA and ATI
  - Render-to-vertex-array
    - Superbuffers
- Semantics Still Under Development...



# Fbuffer: Capturing Fragments

- Idea

- Save all fragment values instead of one per pixel
- “Rasterization-Order FIFO Buffer”



# Fbuffer: Capturing Fragments

---

- Details

- Designed for multi-pass rendering with transparent geometry
- Mark and Proudfoot, Graphics Hardware 2001
  - <http://graphics.stanford.edu/projects/shading/pubs/hwwws2001-fbuffer/>*
- New possibilities for GPGPU
  - Varying number of results per pixel
  - RTT and RTVA with an fbuffer

- OpenGL Support

- ATI Radeon 9800 and newer ATI GPUs
- Not yet exposed to user (ask for it!)



# Overview

---

- GPU Memory Model
- GPU-Based Data Structures
- Pbuffer Survival Guide



# GPU-Based Data Structures

---

- Building Blocks
  - GPU memory addresses
    - Address Generation
    - Address Use
    - Pointers
  - Multi-dimensional arrays
  - Sparse representations

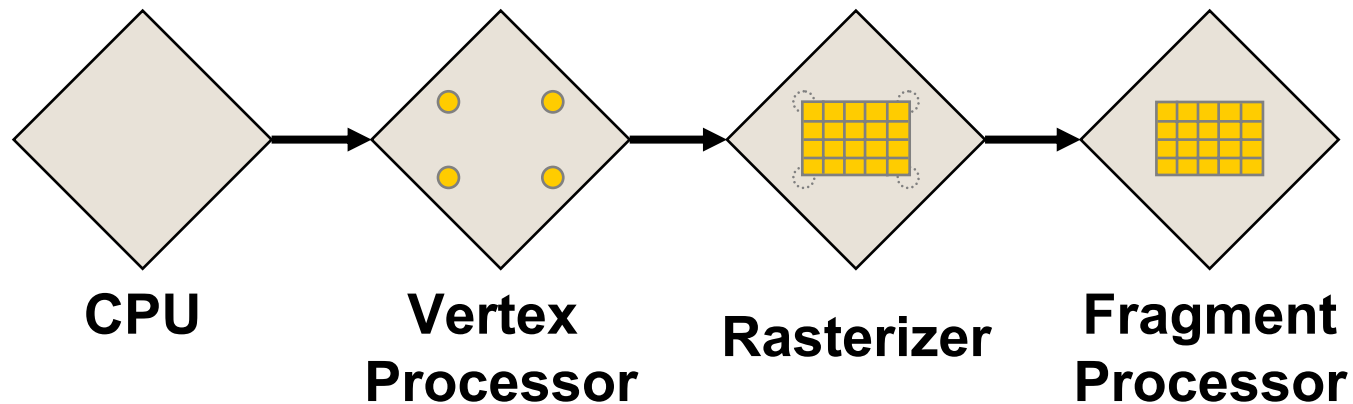


# GPU Memory Addresses

---

- Where Are GPU Addresses Generated?

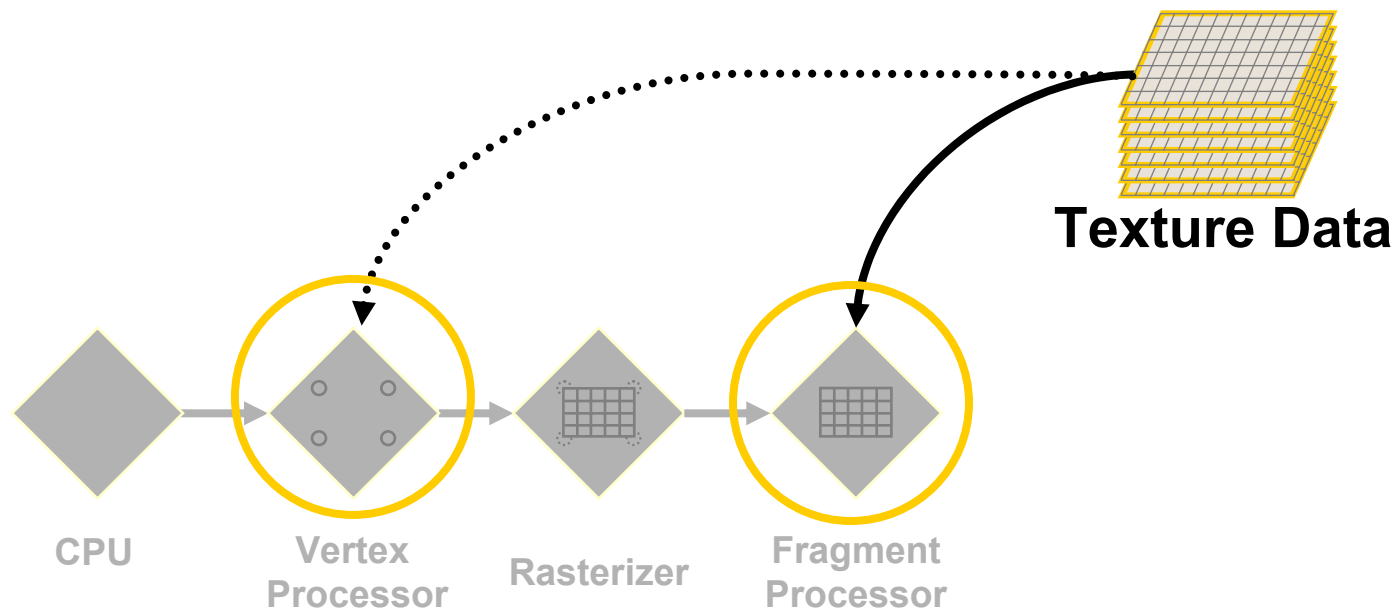
- CPU Vertex stream or textures
- Vertex processor Input stream, ALU ops or textures
- Rasterizer Interpolation
- Fragment processor Input stream, ALU ops or textures



# GPU Memory Addresses

---

- Where Are Addresses Used?
  - Vertex textures (PS3.0 GPUs)
  - Fragment textures



# GPU Memory Addresses

---

- Floating-Point Addressing

- Normalized addresses [0,1]

- GL\_TEXTURE\_1D, \_2D, \_3D, \_CUBE

- Non-Normalized addresses [0,N]

- GL\_TEXTURE\_RECTANGLE

- Warning: Floating-point can leave unaddressable texels

NVIDIA FP32:	16,777,217	Counting numbers
ATI 24-bit float:	131,073	Counting numbers
NVIDIA FP16:	2,049	Counting numbers

*Courtesy of Ian Buck*



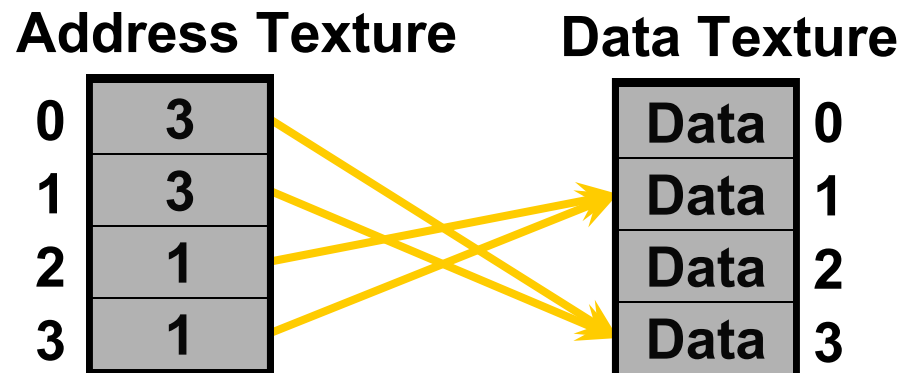
# GPU Memory Addresses

---

- Pointers

- Store addresses in texture
- Dependent texture read

```
float2 addr = tex2D( addrTex, texCoord );  
float2 data = tex2D( dataTex, addr );
```



# GPU-Based Data Structures

---

- Building Blocks

- GPU memory addresses

- Address Generation

- Address Use

- Pointers

- Multi-dimensional arrays and structs

- Sparse representations

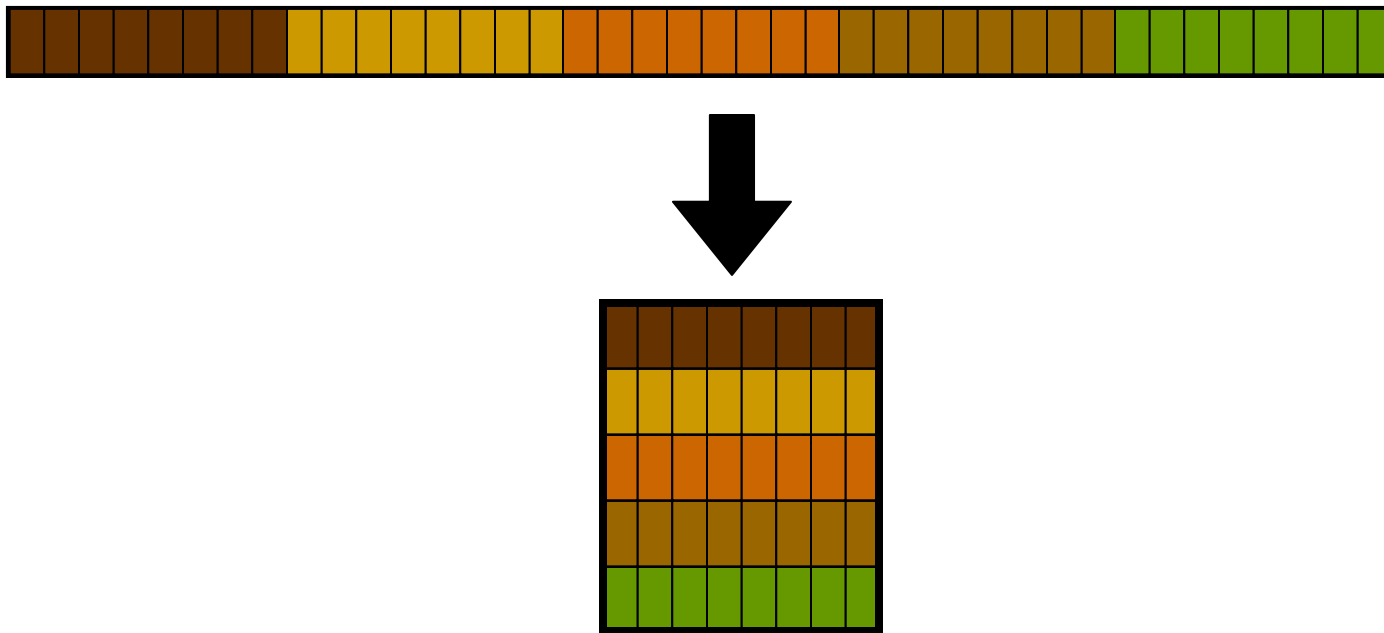


# GPU Arrays

---

- Large 1D Arrays

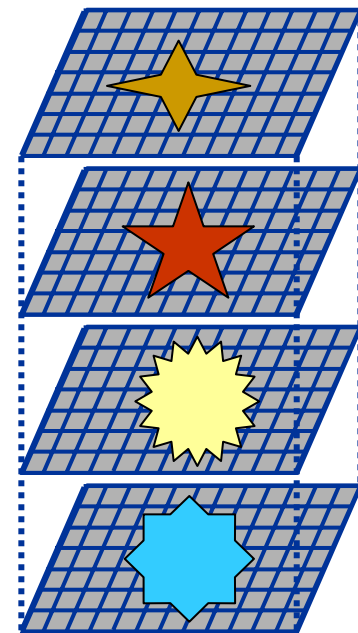
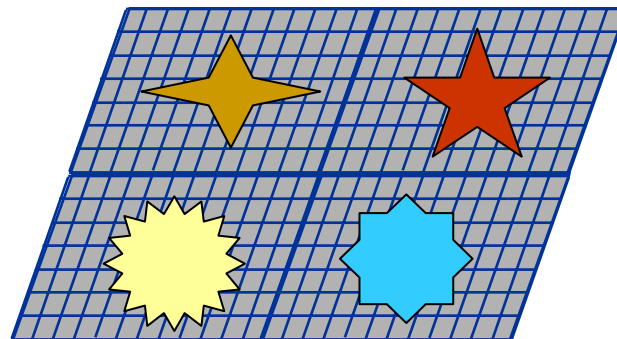
- Current GPUs limit 1D array sizes to 2048 or 4096
- Pack into 2D memory
- 1D-to-2D address translation



# GPU Arrays

---

- 3D Arrays
  - Problem
    - GPUs do not have 3D frame buffers
    - No RTT to slice of 3D texture with pbuffers
  - 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
    - GL\_EXT\_render\_target and Superbuffers
  - Volume rendering of slice-based 3D data
    - Course 28, “Real-Time Volume Graphics”, Siggraph 2004
    - “Deferred filtering”



# 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

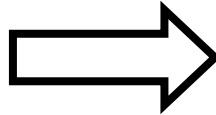


# GPU Structures

---

- Store each member in a different “array” (texture)
  - Update structs with Multiple-Render Targets (MRTs)

```
struct Foo {  
    float4 a;  
    float4 b;  
};  
Foo foo[N];
```



```
float4 Foo_a[N];  
float4 Foo_b[N];
```



# GPU-Based Data Structures

---

- Building Blocks
  - GPU memory addresses
    - Address Generation
    - Address Use
    - Pointers
  - Multi-dimensional arrays
  - Sparse representations



# Sparse Data Structures

---

- Why Sparse Data Structures?

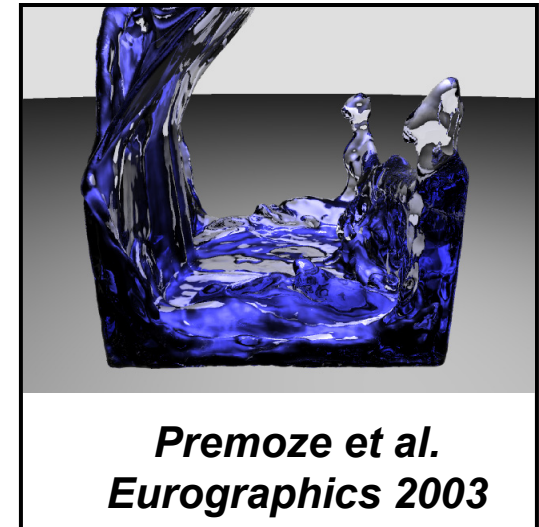
- Reduce memory pressure
- Reduce computational workload

- Examples

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

- Deformable implicit surfaces (sparse volumes/PDEs)

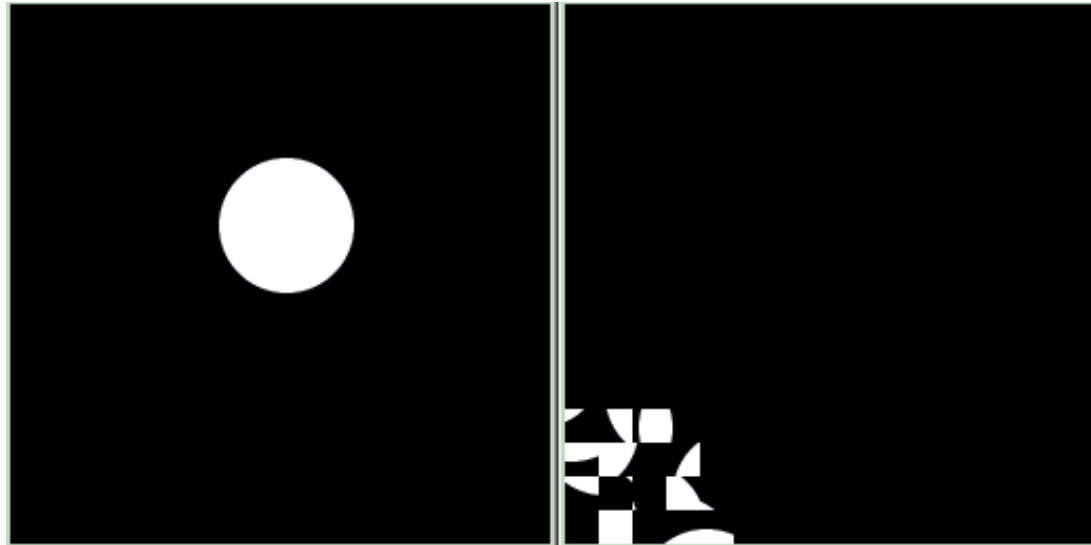
- Lefohn et al., IEEE Visualization 2003



# Sparse Data Structures

---

- Basic Idea
  - Pack “active” data elements into GPU memory
  - For more information
    - Linear algebra section in this course : Static structures
    - Level-set case study in this course : Dynamic structures



# 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
- Pbuffer Survival Guide



# Pbuffer Survival Guide

---

- Pbuffers Give us Render-To-Texture
  - Designed to create an environment map or two
  - Never intended to be used for GPGPU (100s of pbuffers)
- Problem
  - Each pbuffer has its own OpenGL render context
  - Each pbuffer may have depth and/or stencil buffer
  - Changing OpenGL contexts is slow
- Solution
  - Many optimizations to avoid this bottleneck...

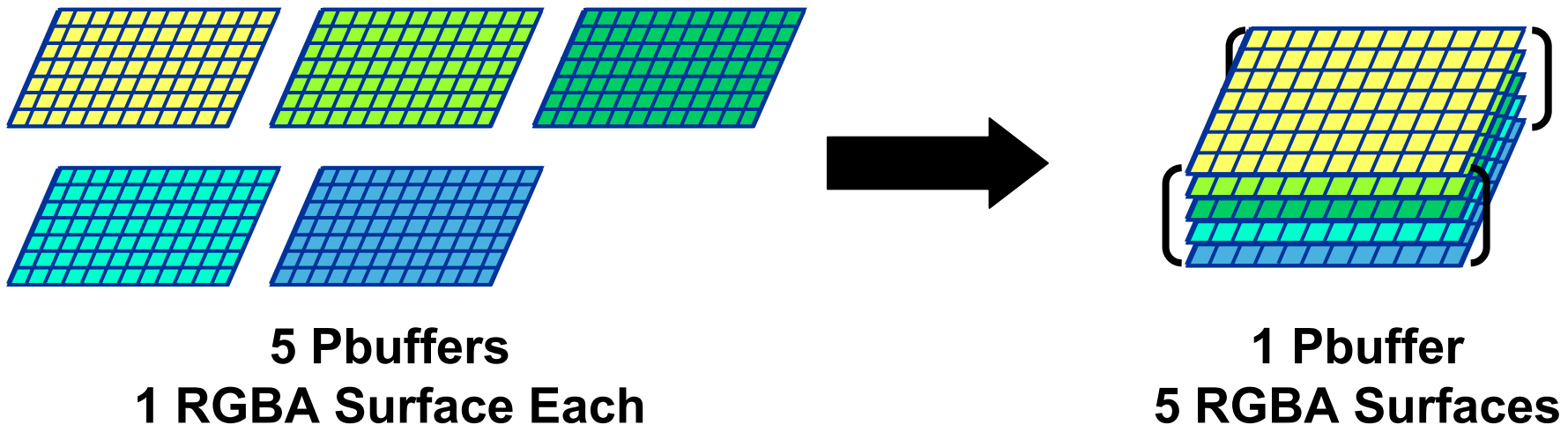


# Pbuffer Survival Guide

---

## 1. Use Multi-Surface Pbuffers

- Each RGBA surface is its own render-texture
  - Front, Back, AuxN (N = 0,1,2,...)
- Greatly reduces context switches
- Technically illegal, but “blessed” by ATI and NVIDIA



# Pbuffer Survival Guide

---

## 1. Using Multi-Surface Pbuffers

a) Allocate multi-surface pbuffer (Front/Back/AUX buffers)

b) Set render target to back buffer

```
glDrawBuffer(GL_BACK)
```

c) Bind front buffer as texture

```
wglBindTexImageARB(pbuffer, WGL_FRONT_ARB)
```

d) Render

e) Switch buffers

```
wglReleaseTexImageARB(pbuffer, WGL_FRONT_ARB)
```

```
glDrawBuffer(GL_FRONT)
```

```
wglBindTexImageARB(pbuffer, WGL_BACK_ARB)
```

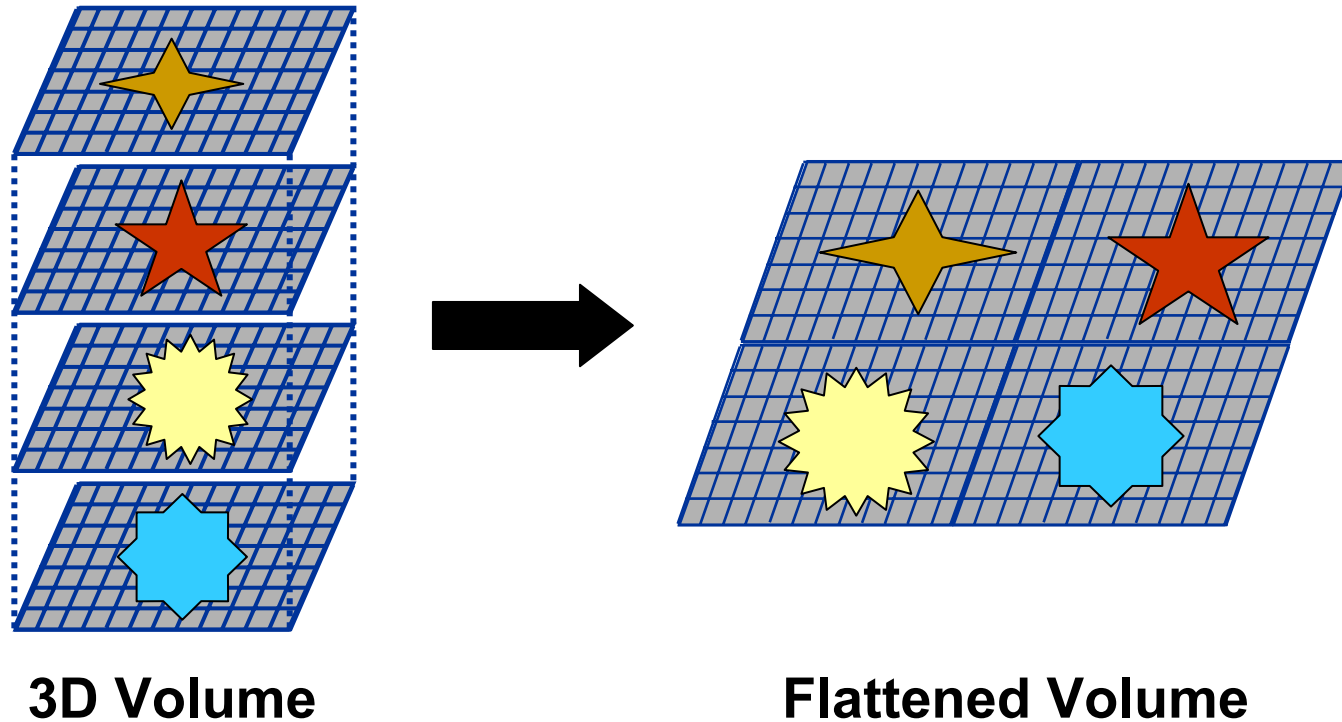


# Pbuffer Survival Guide

---

## 2. Pack 2D domains into large buffer

- “Flat 3D textures”
- Be careful of read-modify-write hazard



# 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 stream processors
- GPGPU Data Structures
  - Basic memory primitive is limited-size, 2D texture
  - Use address translation to fit all array dimensions into 2D
- Render-To-Texture
  - Use pbuffers with care and eagerly adopt their successor



# Acknowledgements

---

- Nick Triantos, Craig Kolb, Cass Everitt, Chris Seitz at NVIDIA
- Mark Segal, Rob Mace, Arcot Preetham, Evan Hart at ATI
- Brian Budge, Ph.D. student at UC Davis and NVIDIA intern
- The other GPGPU IEEE Visualization 2004 course presenters
  
- John Owens, Ph.D. advisor, Univ. of California Davis
- Ross Whitaker, M.S. advisor, SCI Institute, Univ. of Utah
  
- National Science Foundation Graduate Fellowship
- Pixar Animation Studios, summer internships
-  Interchangeable mobile GPUs



# Selected References

---

- J. Boltz, I. Farmer, E. Grinspun, P. Schoder, “Spare Matrix Solvers on the GPU: Conjugate Gradients and Multigrid,” SIGGRAPH 2003
- N. Goodnight, C. Woolley, G. Lewin, D. Luebke, G. Humphreys, “A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware,” Graphics Hardware 2003
- M. Harris, W. Baxter, T. Scheuermann, A. Lastra, “Simulation of Cloud Dynamics on Graphics Hardware,” Graphics Hardware 2003
- H. Igehy, M. Eldridge, K. Proudfoot, “Prefetching in a Texture Cache Architecture,” Graphics Hardware 1998
- J. Krueger, R. Westermann, “Linear Algebra Operators for GPU Implementation of Numerical Algorithms,” SIGGRAPH 2003
- A. Lefohn, J. Kniss, C. Hansen, R. Whitaker, “A Streaming Narrow-Band Algorithm: Interactive Deformation and Visualization of Level Sets,” IEEE Transactions on Visualization and Computer Graphics 2004



# Selected References

---

- A. Lefohn, J. Kniss, C. Hansen, R. Whitaker, “Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware,” IEEE Visualization 2003
- W. Mark, K. Proudfoot, “The F-Buffer: A Rasterization-Order FIFO Buffer for Multi-Pass Rendering,” Graphics Hardware 2001
- T. Purcell, C. Donner, M. Cammarano, H. W. Jensen, P. Hanrahan, “Photon Mapping on Programmable Graphics Hardware,” Graphics Hardware 2003
- A. Sherbondy, M. Houston, S. Napel, “Fast Volume Segmentation With Simultaneous Visualization Using Programmable Graphics Hardware,” IEEE Visualization 2003



# OpenGL References

---

- GL\_EXT\_pixel\_buffer\_object  
[http://www.nvidia.com/dev\\_content/nvopenglspecs/GL\\_EXT\\_pixel\\_buffer\\_object.txt](http://www.nvidia.com/dev_content/nvopenglspecs/GL_EXT_pixel_buffer_object.txt)
- GL\_EXT\_render\_target,  
[http://www.opengl.org/resources/features/GL\\_EXT\\_render\\_target.txt](http://www.opengl.org/resources/features/GL_EXT_render_target.txt)
- OpenGL Extension Registry  
<http://oss.sgi.com/projects/ogl-sample/registry/>
- Superbuffers  
<http://www.ati.com/developer/gdc/SuperBuffers.pdf>
- WGL\_ARB\_render\_texture  
[http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl\\_render\\_texture.txt](http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl_render_texture.txt)  
[http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl\\_pbuffer.txt](http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl_pbuffer.txt)

