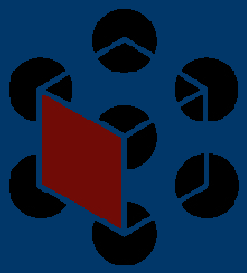
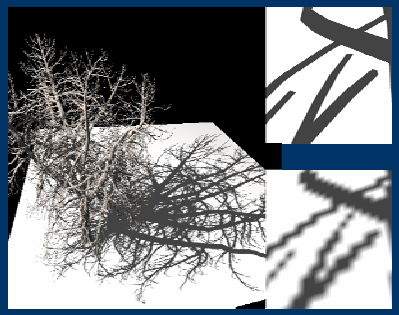
Glift: Generic, Efficient Random-Access GPU Data Structures

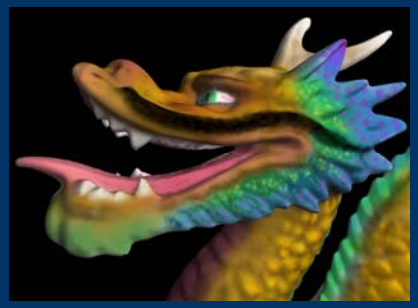
Aaron Lefohn
University of California, Davis



Problem Statement

- "A data structure abstraction for graphics processing units (GPUs) can simplify the description of new and existing data structures, stimulate development of complex GPU algorithms, and perform equivalently to hand-coded implementations."

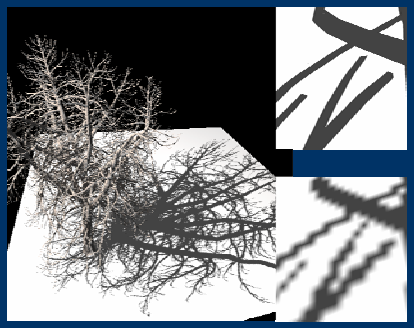




Aaron Lefohn
University of California, Davis

Problem Statement

- **Goal**
  - Simplify creation and use of random-access GPU data structures for graphics and GPGPU programming



- **Contributions**
  - Abstraction for GPU data structures
  - Glift template library
  - Iterator computation model for GPUs

Aaron Lefohn
University of California, Davis

Collaborators

- **Joe Kniss**
  University of Utah

- **Robert Strzodka**
  Stanford University

- **Shubhabrata Sengupta**
  University of California, Davis

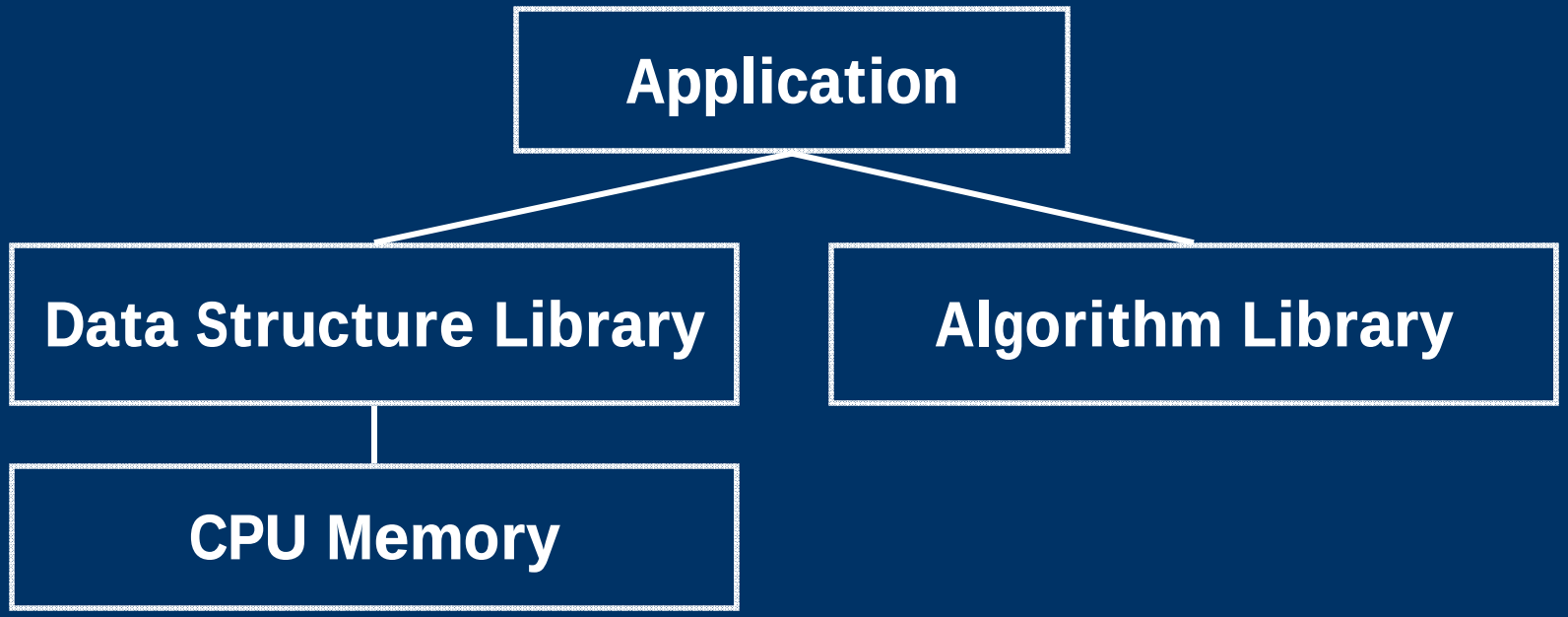
- **John Owens**
  University of California, Davis

Many Interesting GPU Data Structures

• Photon map	Purcell
• Sparse matrix	Boltz, Krueger
• Sparse simulation grid	Lefohn
• Polycube (3D grid, cubeMap, …)	Tarini
• N-tree	Lefebvre

- **But…**
  - No way to distribute/reuse implementations
  - Complexity stifles innovation

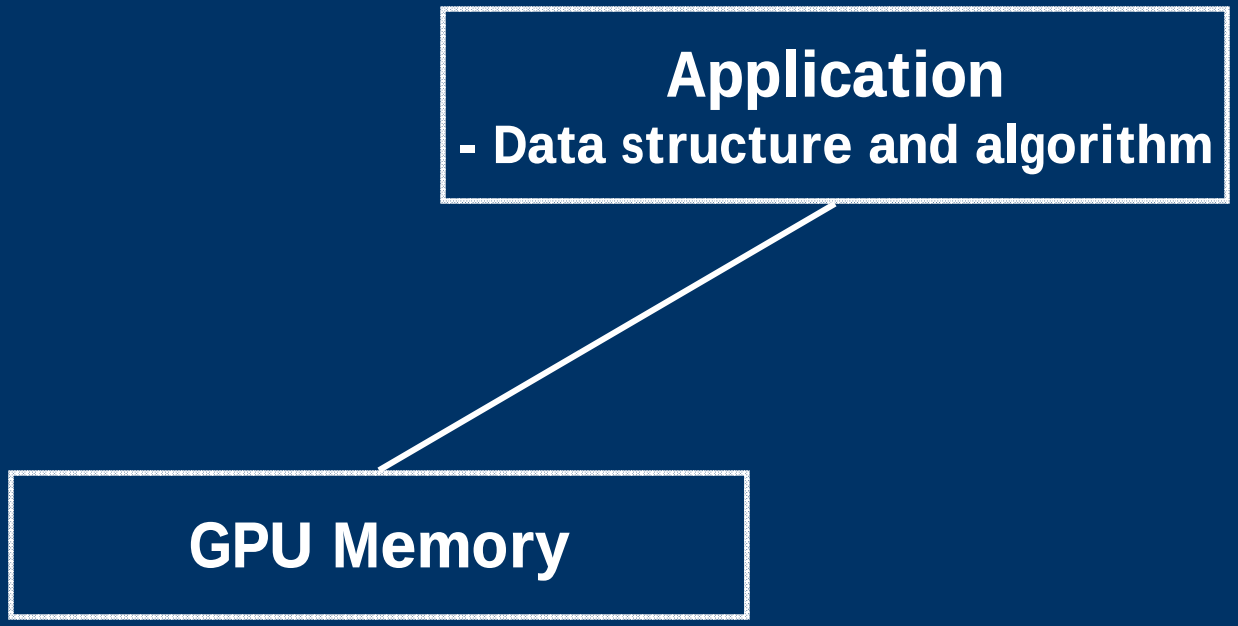
CPU Software Development



- **Benefits**
  - Algorithms and data structures expressed in problem domain
  - Decouple algorithms and data structures
  - Code reuse

Aaron Lefohn
University of California, Davis

GPU Software Development

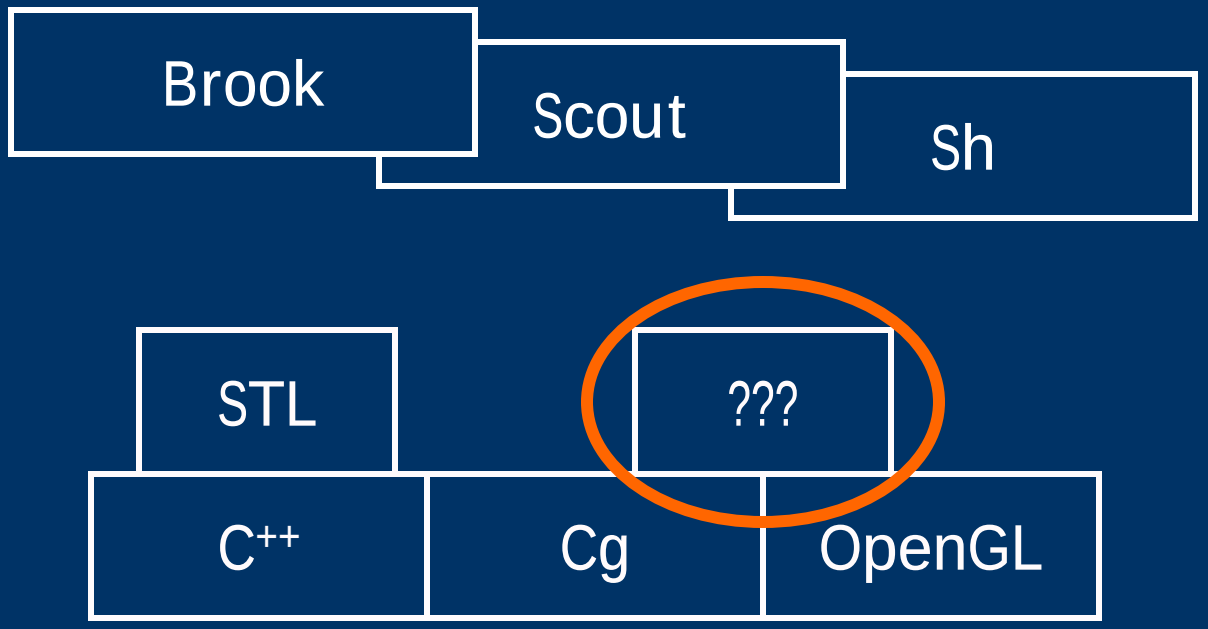


- **Problems**
  - Code is tangled mess of algorithm and data structure access
  - Algorithms expressed in GPU memory domain
  - No code reuse

Aaron Lefohn
University of California, Davis

GPU Data Structures

- **What's Missing?**
  - Standalone abstraction for GPU data structures for graphics or GPGPU programming



Aaron Lefohn
University of California, Davis

Simple Example

- CPU (C++)

```
float srcData[10][10][10];
float dstData[10][10][10];

… initialize data …

for (size_t z = 1; z < 10; ++z) {
  for (size_t y = 1; z < 10; ++y) {
    for (size_t x = 1; z < 10; ++x) {
      dst[z][y][x] = log( 1 + src[z][y][x] );
    }
  }
}
```

Aaron Lefohn
University of California, Davis

We Want To Transform This...

- GPU (Cg)

```
float3 getAddr3D( float2 winPos, float2 winSize, float3 sizeConst3D ) {
  float3 curAddr3D;
  float2 winPosInt = floor(winPos);
  float  addr1D = winPosInt.y * winSize.x + winPosInt.x;
         addr3D.z  = floor( addr1D / sizeConst3D.z );
         addr1D   -= addr3D.z * sizeConst3D.z;
         addr3D.y  = floor( addr1D / sizeConst3D.y );
         addr3D.x  = addr1D - addr3D.y * sizeConst3D.y;
  return addr3D;
}


float3 logAlg(uniform samplerRECT data,
              uniform float2      winSize,
              uniform float3      sizeConst3D,
                      float2      winPos : WPOS ) : COLOR
{
     float3 addr3D = getAddr3D( winPos, winSize, sizeConst3D );
     float  data   = texRECT(data, addr3D );
     return log( 1 + data );
}
```

Aaron Lefohn
University of California, Davis

We Want To Transform This...

- GPU (Cg and C++)

```
float3 getAddr3D( float2 winPos, float2 winSize, float3 sizeConst3D ) {

  float3 curAddr3D;
  float2 winPosInt = floor(winPos);
  float  addr1D = winPosInt.y * winSize.x + winPosInt.x;
         addr3D.z  = floor( addr1D / sizeConst3D.z );
         addr1D   -= addr3D.z * sizeConst3D.z;
         addr3D.y  = floor( addr1D / sizeConst3D.y );
         addr3D.x  = addr1D - addr3D.y * sizeConst3D.y;
  return addr3D;
}


float3 logAlg(uniform samplerRECT data,
                        uniform float2    winSize,
                        uniform float3      sizeConst3D,
                                float2    winPos : WPOS ) : COLOR
{
         float3 addr3D = getAddr3D( winPos, winSize, sizeConst3D );
         float  data   = texRECT(data, addr3D );
         return log( 1 + data );
}
```

```
GLuint srcDataId = 1;
glBindTexture(GL_TEXTURE_RECTANGLE_ARB,   srcDataId);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_WRAP_T, GL_CLAMP);
glTexImage2D(GL_TEXTURE_RECTANGLE_ARB, 0, GL_LUMINANCE32F_ARB,
             0, 0, 40, 40, GL_LUMINANCE, NULL);

GLuint dstDataId = 2;
glBindTexture(GL_TEXTURE_RECTANGLE_ARB,   dstDataId);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_RECTANGLE_ARB, GL_WRAP_T, GL_CLAMP);
glTexImage2D(GL_TEXTURE_RECTANGLE_ARB, 0, GL_LUMINANCE32F_ARB,
             0, 0, 40, 40, GL_LUMINANCE, NULL);

…  Initialize data …
```

Into This.

- GPU (C++ and Cg with Glift)

```
typedef glift::ArrayGpu<vec3i,vec1f> ArrayType;
ArrayType src( vec3i(10,10,10) );
ArrayType dst( vec3i(10,10,10) );

        … initialize data …
```

```
float logAlg( ElementIter srcData ) : COLOR
{
   return log( 1 + srcData.value() );
}
```

Aaron Lefohn
University of California, Davis

Overview

- Motivation and Previous Work
- **Abstraction**
- Implementation
- Examples
- Conclusions

Aaron Lefohn
University of California, Davis

Abstraction Design Goals

- **GPU data structure abstraction that**
  - Enables easy creation of new structures
  - Is minimal abstraction of GPU memory model
  - Separates data structures and algorithms
  - Encourages efficiency

Building the Abstraction

- **Approach**
  - Bottom-up, working towards STL-like syntax
  - Identify common patterns in GPU papers and code
  - Inspired by
    - STL, Boost, Brook, STAPL, Stepanov

What is the GPU Memory Model?

- **CPU interface**

• **`glTexImage`**	malloc	
• **`glDeleteTextures`**	free	
• **`glTexSubImage`**	memcpy	GPU -> CPU
• ***`glGetTexSubImage*`***	memcpy	CPU -> GPU
• **`glCopyTexSubImage`**	memcpy	GPU -> GPU
• **`glBindTexture`**	read-only	parameter bind
• **`glFramebufferTexture`**	write-only	parameter bind

***** Does not exist. Emulate with **`glReadPixels`**

What is the GPU Memory Model?

- **GPU Interface (shown in Cg)**

• `uniform samplerND`	data structure param declaration
• `texND(tex, addr)`	random-access read
• `varying floatN stream`	stream parameter declaration
• `stream`	stream read

Aaron Lefohn
University of California, Davis

GPU Data Structure Abstraction

- **Factor GPU data structures into**
  - Physical memory
  - Virtual memory
  - Address translator
  - Iterators

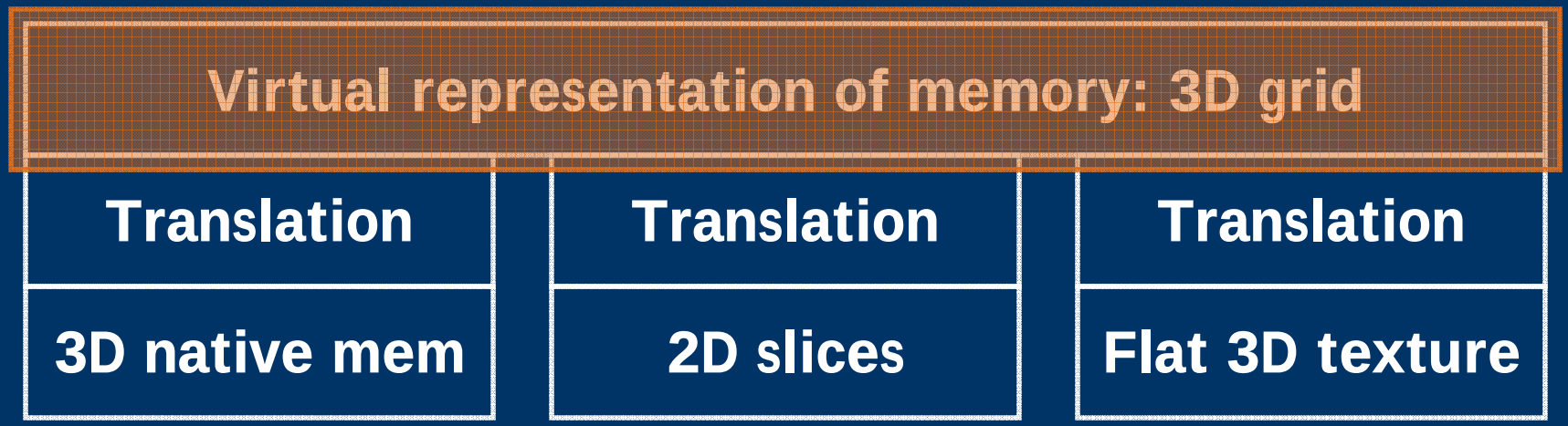
Aaron Lefohn
University of California, Davis

Physical Memory

- **Native GPU textures**
  - Choose based on algorithm efficiency requirements
  - 1D, 2D, 3D, Cube, Mip
    - Dimensionality
    - Read-only vs. read-write
    - Point-sample vs. filtering
    - Maximum size

Virtual Memory

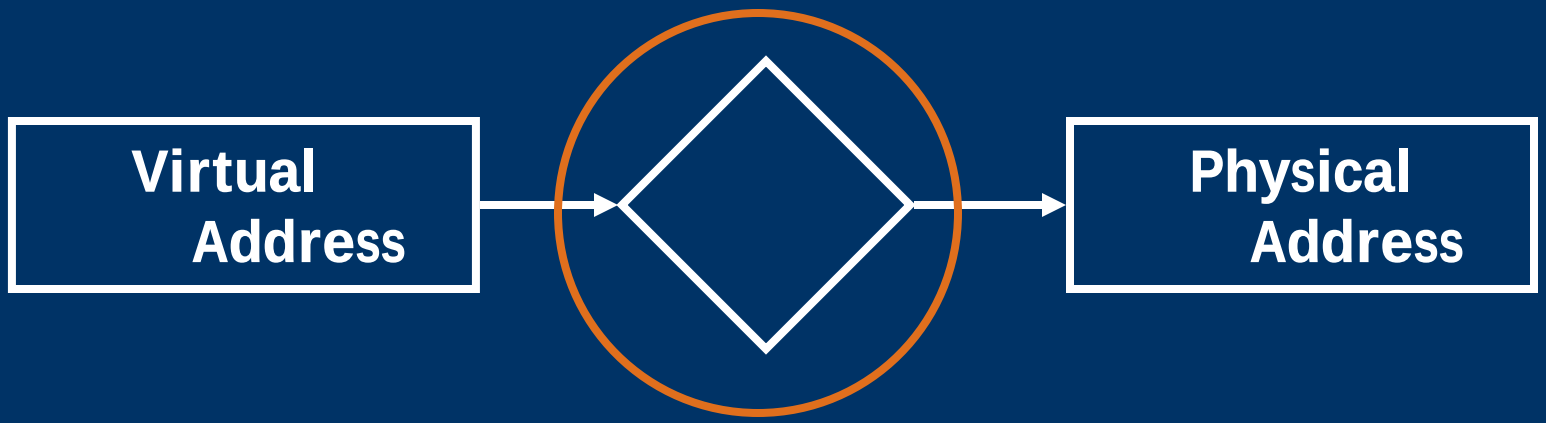
- **Virtual N-D address space**
  - Choose based on problem space of algorithm
  - Defined by physical memory and address translator



Aaron Lefohn
University of California, Davis

Address Translator

- **Mapping between physical and virtual addrs**



  - Core of data structure
  - Small amount of code defines *all* required CPU and GPU memory interfaces

Address Translator

- **Core of data structure**
  - Extension point for creating new structures
  - Must define

```
translate(…)
translate_range(…)
```

Aaron Lefohn
University of California, Davis

Address Translator Classifications

- **Representation**
  - Analytic / Discrete
- **Memory Complexity**
  - O(1), O(log N), O(N), …
- **Compute Complexity**
  - O(1), O(log N), O(N), …
- **Compute Consistency**
  - Uniform vs. non-uniform
- **Total / Partial**
  - Complete vs. sparse
- **One-to-one / Many-to-one**
  - Uniform vs. adaptive

Aaron Lefohn
University of California, Davis

Data Structure Examples

- **Brook streams** **(Buck et al. 2004)**

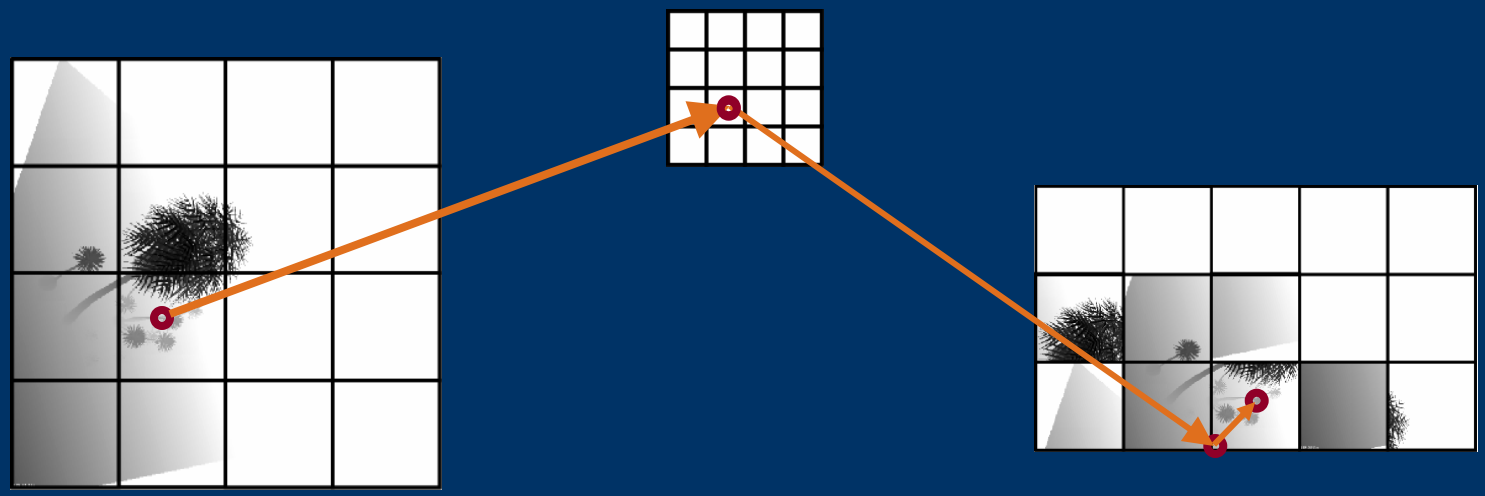
Aaron Lefohn
University of California, Davis

Data Structure Examples

- **Brook streams** **(Buck et al. 2004)**
  - Physical address — 2D
  - Virtual address — N-D
  - Address translator — ND-to-2D
    - Analytic
    - O(1) memory
    - O(1) compute
    - Uniform consistency
    - Total, uniform mapping

Data Structure Examples

- **Dynamic sparse 3D grid** **(Lefohn et al. 2003)**



Virtual Domain **Page Table** **Physical Memory**

Aaron Lefohn
University of California, Davis

Data Structure Examples

- **Dynamic sparse 3D grid (Lefohn et al. 2003)**
  - Physical address — 2D
  - Virtual address — 3D
  - Address translator — 3D page table
    - Discrete
    - O(N) memory
    - O(1) compute
    - Uniform consistency
    - Partial, uniform mapping

Data Structure Examples

- **Photon Map (kNN-grid)** **(Purcell et al. 2003)**

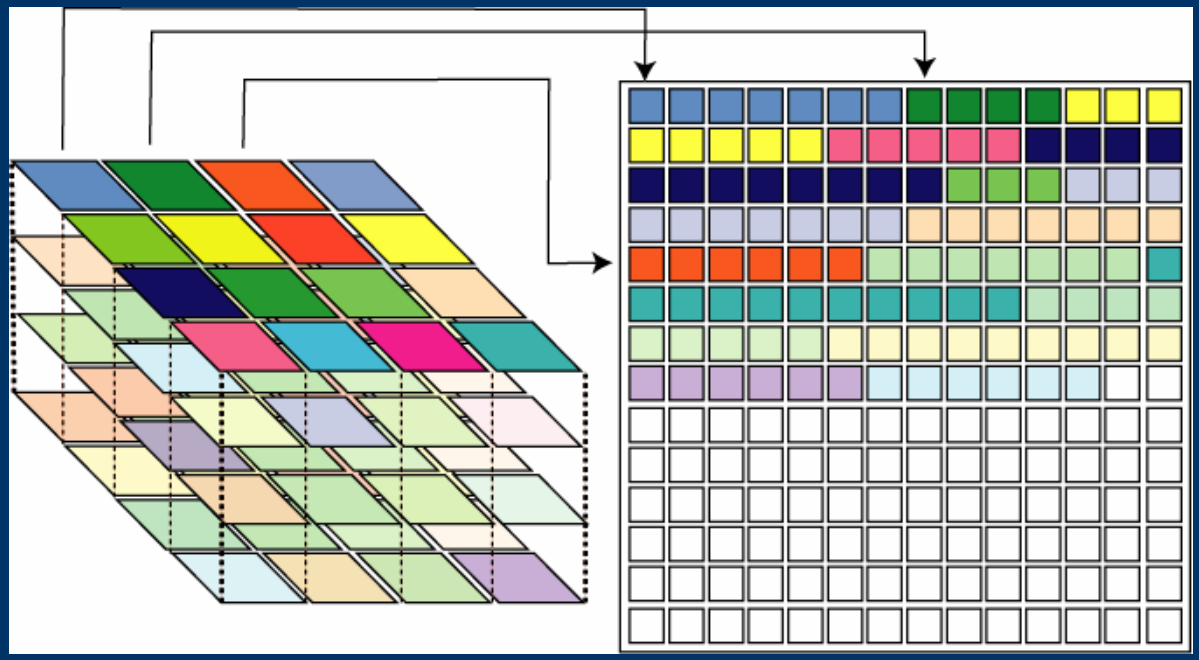


Image from "Implementing Efficient Parallel Data Structures on GPUs," Lefohn et al., GPU Gems II, ch. 33, 2005

Data Structure Examples

- **Photon Map (kNN-grid)** **(Purcell et al. 2003)**
  - Physical address — 2D
  - Virtual address — 3D
  - Address translator — 3D page table
    - Variable sized phys pages
    - "Grid of lists"
    - Discrete
    - O(N) memory
    - O(L) compute
    - Non-uniform consistency
    - Partial, adaptive mapping

Glift Iterators

- **We've so far only discussed data *access***
- **What about data structure *traversal*?**

Aaron Lefohn
University of California, Davis

Iterators

- **Separate algorithms and data structures**
  - Minimal interface between data and algorithm
  - Required for GPGPU use of data structure
  - Encapsulate GPGPU optimizations

Iterators

- **Abstract data access and traversal**

```
DataStructureType::iterator it;
for (it = data.begin(); it != data.end(); ++it)
{
   *it = -(*it);
}
```

Aaron Lefohn
University of California, Davis

Glift Iterators

- **Address iterators**
  - Iterator value is N-D address
  - GPU interpolants
- **Element iterators**
  - Iterator value is data structure element
  - C/C++ pointer, STL iterator, streams

Aaron Lefohn
University of California, Davis

Element Iterator Concepts

- **Permission**
  - Read-only, write-only, read-write
- **Access region**
  - Single, neighborhood, random
- **Traversal**
  - Forward, backward, parallel range

Aaron Lefohn
University of California, Davis

Which Element Iterators?

- **Read-only, single access, range iterator**
  - GPU stream input
- **Read-only, random-access, range iterator**
  - GPU texture input
- **Write-only, single access, range iterator**
  - GPU render target

Aaron Lefohn
University of California, Davis

Example 1 : “Before” and “After” Glift

- Transform GPU code with Glift

Aaron Lefohn
University of California, Davis

Simple Example

- **3D Array with 2D physical memory**

CPU (C++)

```
float srcData[10][10][10];
float dstData[10][10][10];

 … initialize data …

for (size_t z = 1; z < 10; ++z) {
  for (size_t y = 1; z < 10; ++y) {
    for (size_t x = 1; z < 10; ++x) {
      dstData[z][y][x] = srcData[z-1][y-1][x-1];
    }
  }
}
```

Aaron Lefohn
University of California, Davis

Example 1: Shader w/out Glift

```
float3 physToVirt( float2 pa, float2 physSize, float3 virtSizes ) {
  float3 va;
  float  addr1D = pa.y * physSize.x + pa.x;
         va.z  = floor( addr1D / virtSizes.z );
         addr1D   -= va.z * sizeConst3D.z;
         va.y  = floor( addr1D / virtSizes.y );
         va.x  = addr1D - va.y * virtSizes.y;
  return va;
}
```

Physical-to-Virtual Address Translation

```
float2 virtToPhys( float3 va, float2 physSize, float3 virtSizes ) {
  float  addr1D = dot( va, virtSizes );
  float  normAddr1D = addr1D / physSize.x;
  float2 pa = float2(frac(normAddr1D) * physSize.x, normAddr1D);
}
```

Virtual-to-Physical Address Translation

```
float3 main(  uniform samplerRECT physMem,
              uniform float2      physSize,
              uniform float3      virtSizes,
                      float2      pa : WPOS ) : COLOR
{
  float3 va = physToVirt( floor(pa), physSize, virtSizes );
  float3 neighborAddr = va - float3(1, 1, 1);
  return texRECT(data, virtToPhys(neighborAddr3D, physSize, virtSizes) );
}
```

Physical Memory Read

Aaron Lefohn
University of California, Davis

Example 1: Glift Components

```
float3 physToVirt( float2 pa, float2 physSize, float3 virtSizes ) {
  float3 va;
  float  addr1D = pa.y * physSize.x + pa.x;
         va.z  = floor( addr1D / virtSizes.z );
         addr1D   -= va.z * sizeConst3D.z;
         va.y  = floor( addr1D / virtSizes.y );
         va.x  = addr1D - va.y * virtSizes.y;
  return va;
}
```

Address Iterator

```
float2 virtToPhys( float3 va, float2 physSize, float3 virtSizes ) {
  float  addr1D = dot( va, virtSizes );
  float  normAddr1D = addr1D / physSize.x;
  float2 pa = float2(frac(normAddr1D) * physSize.x, normAddr1D);
}
```

VirtMem

```
float3 main(  uniform samplerRECT physMem,
              uniform float2      physSize,
              uniform float3      virtSizes,
                      float2      pa : WPOS ) : COLOR
{
  float3 va = physToVirt( floor(pa), physSize, virtSizes );
  float3 neighborAddr = va - float3(1, 1, 1);
  return texRECT(data, virtToPhys(neighborAddr3D, physSize, virtSizes) );
}
```

VirtMem

Aaron Lefohn
University of California, Davis

Example 1: GPU Shader with Glift

Cg Usage

```
float3 main( uniform VMem3D srcData,
             AddrIter3D     iter ) : COLOR
{
   float3 va = iter.value();
   return srcData.vTex3D( va – float3(1,1,1) );
}
```

Aaron Lefohn
University of California, Davis

Example 1: Glift Data Structures

C++ Usage

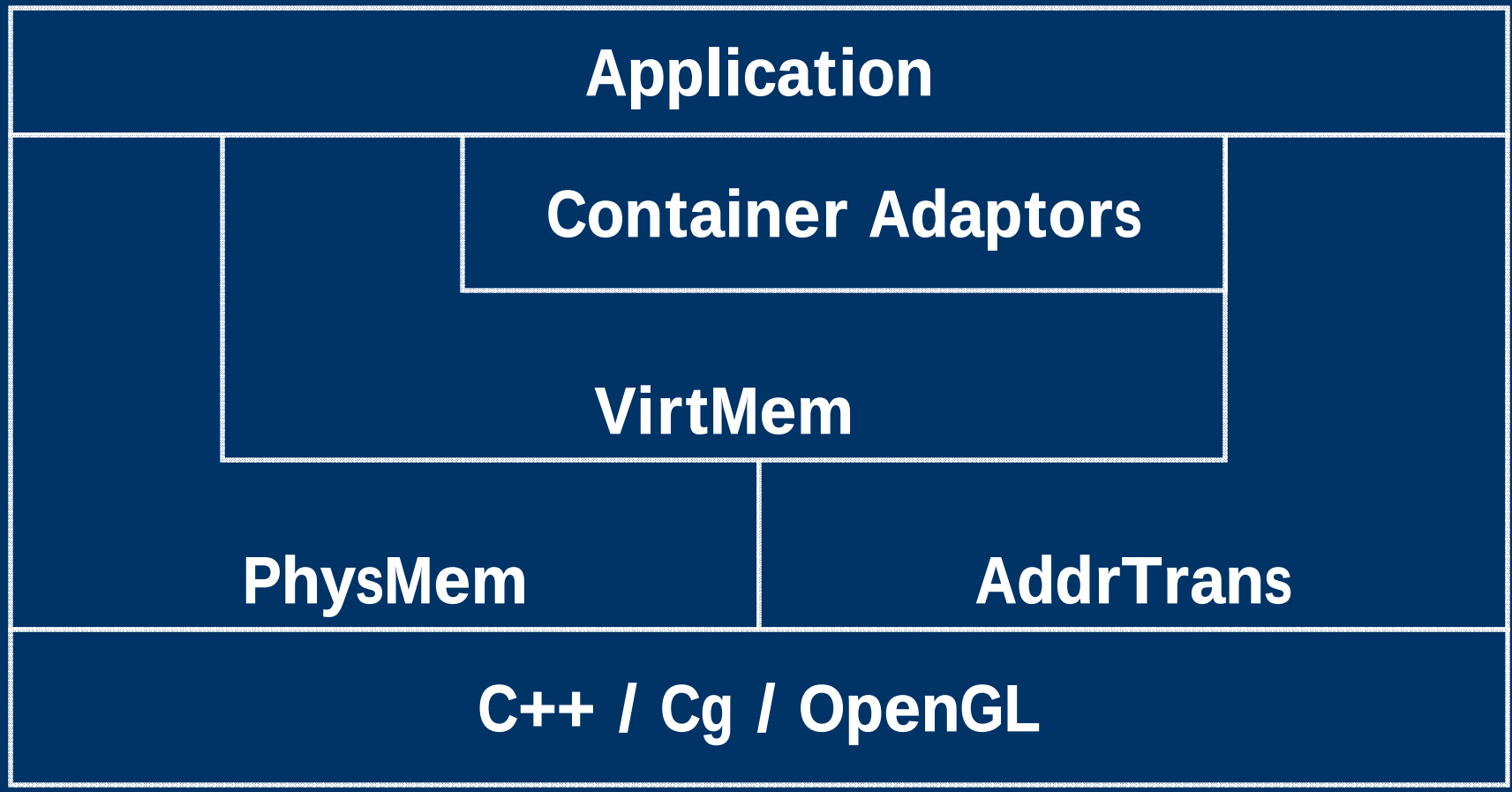
```
vec3i origin(0,0,0);
vec3i size(10,10,10);

typedef ArrayGpu<vec3i,vec1f> ArrayType;
ArrayType srcData( size );
ArrayType dstData( size );

… initialize dataPtr …
srcData.write( origin, size, dataPtr );

typedef ArrayType::addr_trans AddrTransType;
AddrTransType::gpu_range it =
        dstData.addr_trans().gpu_range(origin, size);

it.bind_for_read( iterCgParam );
srcData.bind_for_read( srcCgParam );
dstData.bind_for_write( COLOR0, myFrameBufferObject );

exec_gpu_iterators( it );
```

Overview

- Motivation
- Abstraction
- **Implementation**
- Examples
- Conclusions

Aaron Lefohn
University of California, Davis

Glift Components



Aaron Lefohn
University of California, Davis

Glift Design Goals

- **Efficiency**
- **Easy, incremental adoption**
- **Easily extensible**
- **CPU/GPU interoperability**

Glift Design Goals

- **Efficiency**
  - Static polymorphism (C++ and Cg)
  - Cg program specialization
  - Cg compiler optimizations
- **Easy, incremental adoption**
- **Easily extensible**
- **CPU/GPU interoperability**

Glift Design Goals

- **Efficiency**
- **Easy, incremental adoption**
  - Integrate with Cg/OpenGL/C++
  - STL-like and texture-like interfaces
  - Use components alone or composited
- **Easily extensible**
- **CPU/GPU interoperability**

Aaron Lefohn
University of California, Davis

Glift Design Goals

- **Efficiency**
- **Easy, incremental adoption**
- **Easily extensible**
  - Create new structure by:
    - Change behavior of existing address translator
    - New address translator
    - New container adaptor
- **CPU/GPU interoperability**

Glift Design Goals

- Efficiency
- Easy, incremental adoption
- Easily extensible
- **CPU/GPU interoperability**
  - Unified C++/Cg code base
  - Map memory to CPU or GPU
  - CPU and GPU iterators

C++/Cg Integration

- **Each component defines C++ and Cg code**
  - C++ objects have Cg struct representation
  - Stringified Cg parameterized by C++ templates
- **Cg "template" instantiation**
  - Insert generated Glift source code into shader

```
glift::cgGetTemplateType<MyDataStructType>();
glift::cgInstantiateParameter(…);
```

- **All other compilation/loading/binding identical to standard shader**

Cg Compilation Example

- **Cg code**

```
float4 main( uniform VMem3D octree,
             float3 coord ) : COLOR
{
    return octree.vMem3D(coord);
}
```

- **C++ code**

```
typedef OctreeGPU<vec4ub> octree_type;
GliftType type = cgGetTemplateType<octree_type>();
CGprogram prog = cgCreateProgram(…);
prog = cgInstantiateParameter(prog, “octree”, type);
cgCompileProgram(prog);
```

Aaron Lefohn
University of California, Davis

Overview

- Motivation and previous work
- Abstraction
- **Case Study**
  - Adaptive shadow maps and octree 3D paint
- Conclusions

Aaron Lefohn
University of California, Davis

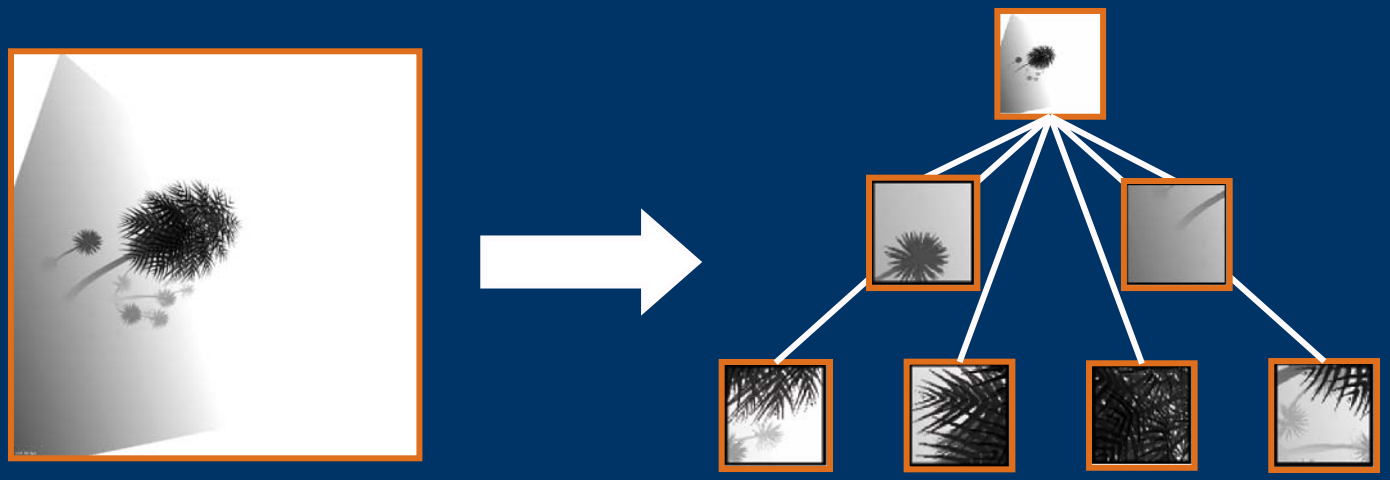
Example 2: Adaptive Shadow Maps

- **Show Glift usage with**
  - Complex application
  - Complex data structure

Aaron Lefohn
University of California, Davis

Example 2: Adaptive Shadow Maps

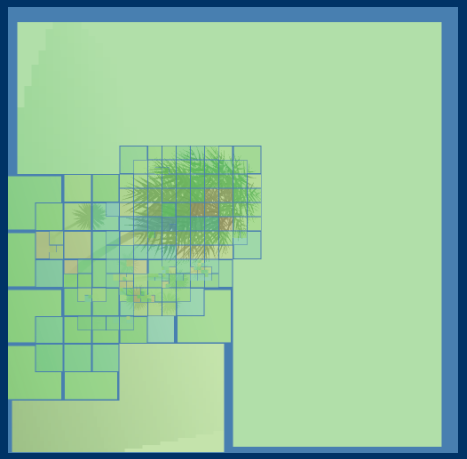
- **Fernando et al., ACM SIGGRAPH 2001**
- **Elegant solution to shadow map aliasing**
  - Quadtree of small shadow maps
  - Shadow maps need resolution only on shadow boundary
  - Required resolution determined by projected area of screen space pixel into light space

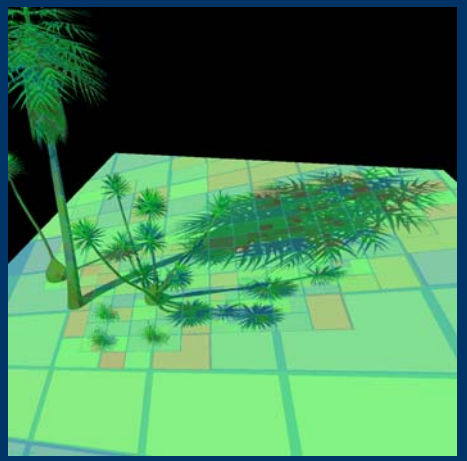


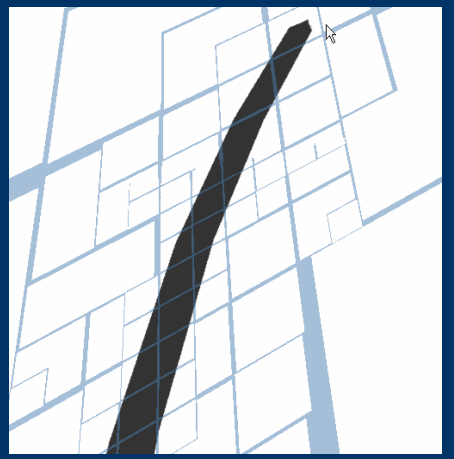
Aaron Lefohn
University of California, Davis

Adaptive Shadow Maps

- **Why Adaptive Shadow Maps with Glift?**
  - Many recent (2004) shadow papers cite ASMs as high quality solution but not possible on graphics hardware
  - Algorithm is simple. Data structure is hard.







Aaron Lefohn
University of California, Davis

Adaptive Shadow Map Algorithm

- **Iterative refinement algorithm**
  - Identify shadow pixels w/ resolution mismatch
  - Create small shadow map "pages" at requested resolution
- **Shadow lookup**
  - Compute shadow map coordinate and resolution
  - Lookup in ASM (tree of small shadow map pages)
- **ASM depends on both camera and light position!**

Aaron Lefohn
University of California, Davis

ASM Data Structure Requirements

- **Adaptive**
- **Multiresolution**
- **Fast, parallel random-access read**
  - 2x2 native Percentage Closer Filtering (PCF)
  - Trilinear interpolated mipmapped PCF
- **Fast, parallel write**
- **Fast, parallel insert and erase**

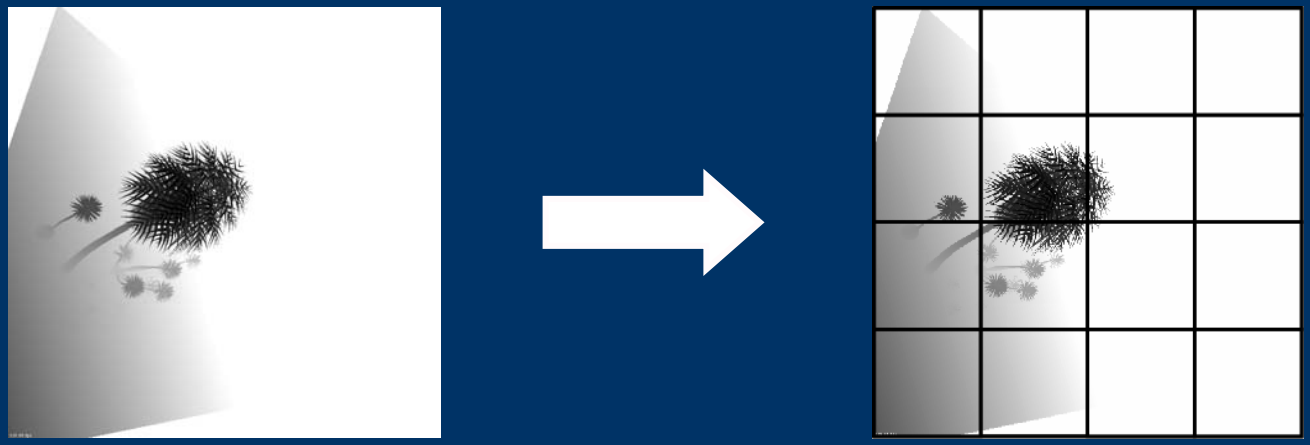
Aaron Lefohn
University of California, Davis

ASM Data Structure

- **Start with page table address translator**
  - Coarse, uniform discretization of virtual domain
    - O(N) memory
    - O(1) computation
    - Uniform consistency
    - Partial mapping (sparse)

    O(1) insert

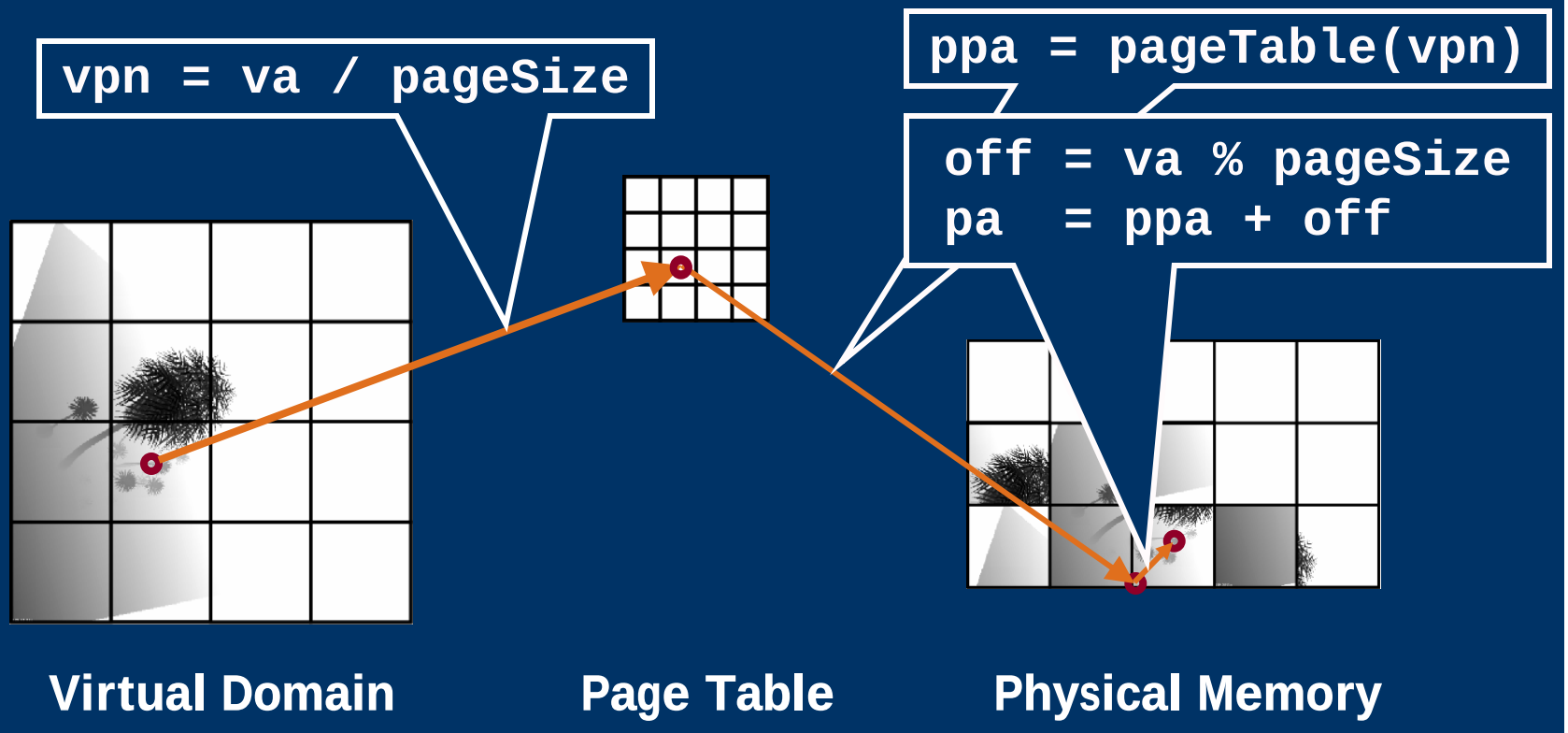
    O(1) erase



Aaron Lefohn
University of California, Davis

ASM Data Structure

- **Page table example**



Aaron Lefohn
University of California, Davis

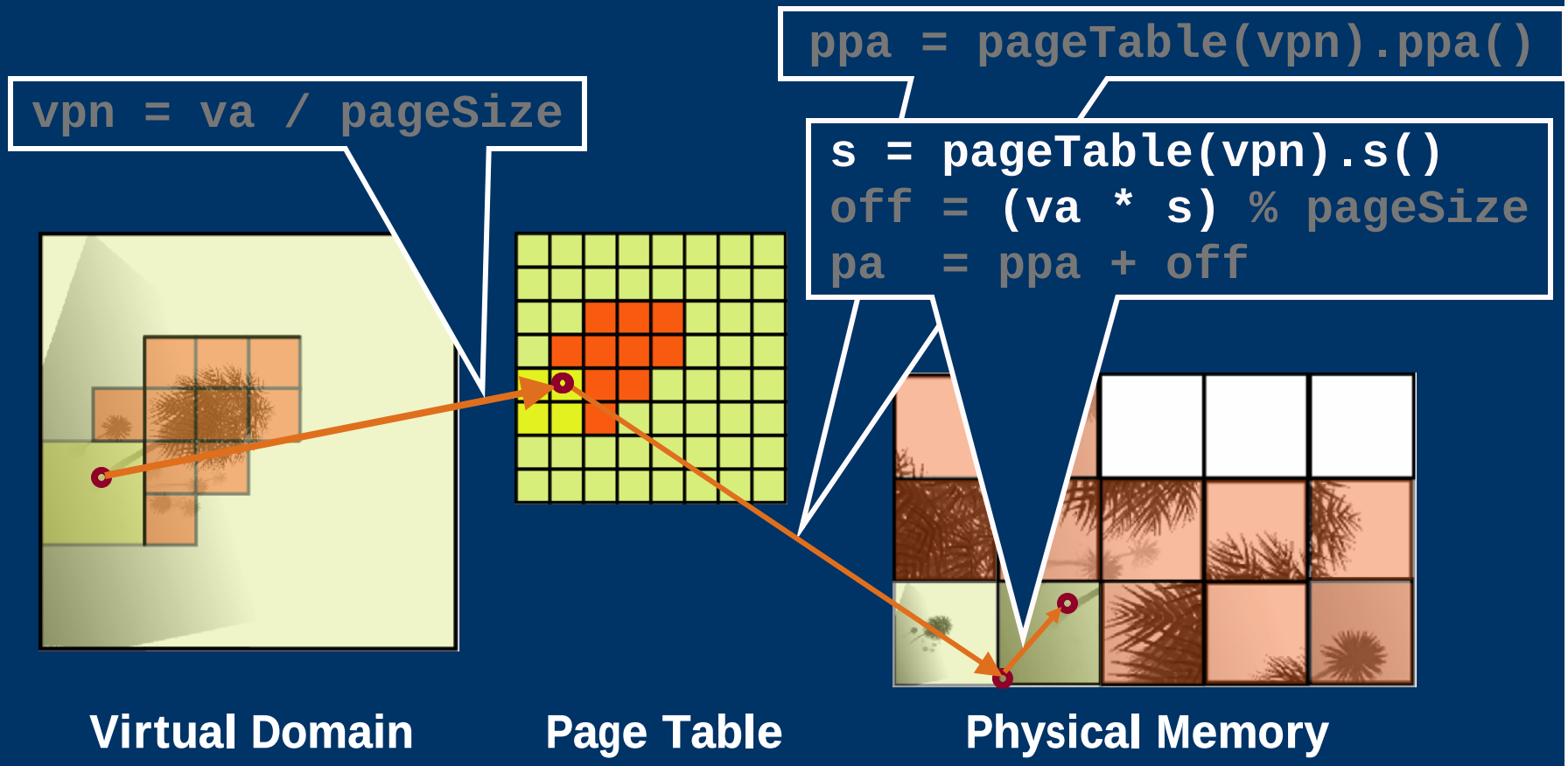
ASM Data Structure Requirements

- **Adaptive**
- **Multiresolution**
- **Fast, parallel random-access read**
  - 2x2 native Percentage Closer Filtering (PCF)
  - Trilinear interpolated mipmapped PCF
- **Fast, parallel write**
- **Fast, parallel insert and erase**

Aaron Lefohn
University of California, Davis

ASM Data Structure

- **Adaptive Page Table**
  - Map multiple virtual pages to single physical page



Aaron Lefohn
University of California, Davis

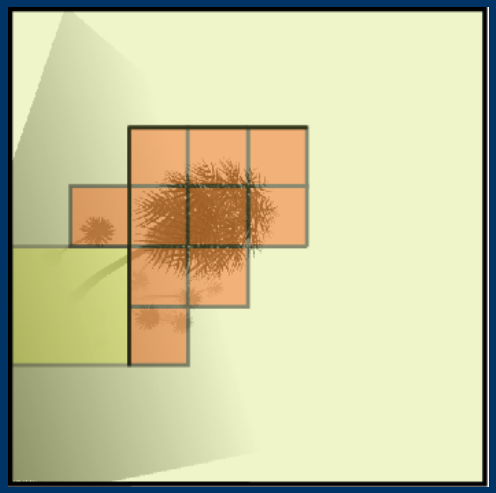
ASM Data Structure Requirements

- **Adaptive**
- **Multiresolution**
- **Fast, parallel random-access read**
  - 2x2 native Percentage Closer Filtering (PCF)
  - Trilinear interpolated mipmapped PCF
- **Fast, parallel write**
- **Fast, parallel insert and erase**

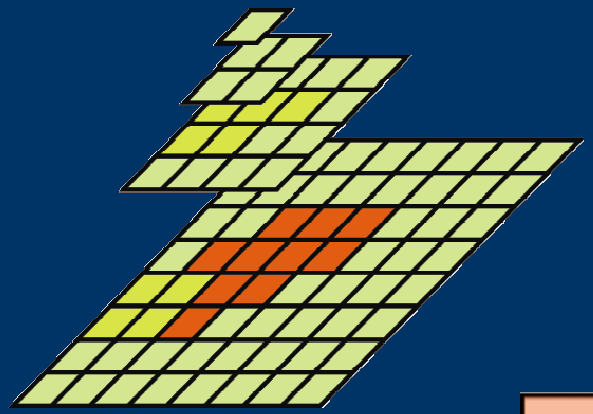
Aaron Lefohn
University of California, Davis

ASM Data Structure

- **Multiresolution Page Table**



Virtual Domain



Mipmap Page Table

Physical Memory

Aaron Lefohn
University of California, Davis

ASM Data Structure Requirements

- **Adaptive**
- **Multiresolution**
- **Fast, parallel random-access read**
  - 2x2 native Percentage Closer Filtering (PCF)
  - Trilinear interpolated mipmapped PCF
- **Fast, parallel write**
- **Fast, parallel insert and erase**

Aaron Lefohn
University of California, Davis

ASM Data Structure Requirements

- **How support bilinear filtering?**
  - Duplicate 1 column and 1 row of texels in each page
- **Mipmapped trilinear?**
  - "By-hand" interpolation between mipmap levels

Aaron Lefohn
University of California, Davis

ASM Data Structure Requirements

- **Adaptive**
- **Multiresolution**
- **Fast, parallel random-access read**
  - 2x2 native Percentage Closer Filtering (PCF)
  - Trilinear interpolated mipmapped PCF
- **Fast, parallel write**
- **Fast, parallel insert and erase**

Aaron Lefohn
University of California, Davis

How Define ASM Structure in Glift?

- **Start with generic page table AddrTrans**
  - Use mipmapped PhysMem for page table
  - Change template parameter to add adaptivity
- **Write page allocator**
  - `alloc_pages, free_pages`
- **Finally…**

```
typedef PageTableAddrTrans<…>              PageTable;
typedef PhysMemGPU<vec2f, vec1s>           PMem2D;
typedef VirtMemGPU<PageTable, PMem2D>      VPageTable;
typedef AdaptiveMem<VPageTable, PageAllocator> ASM;
```

Aaron Lefohn
University of California, Davis

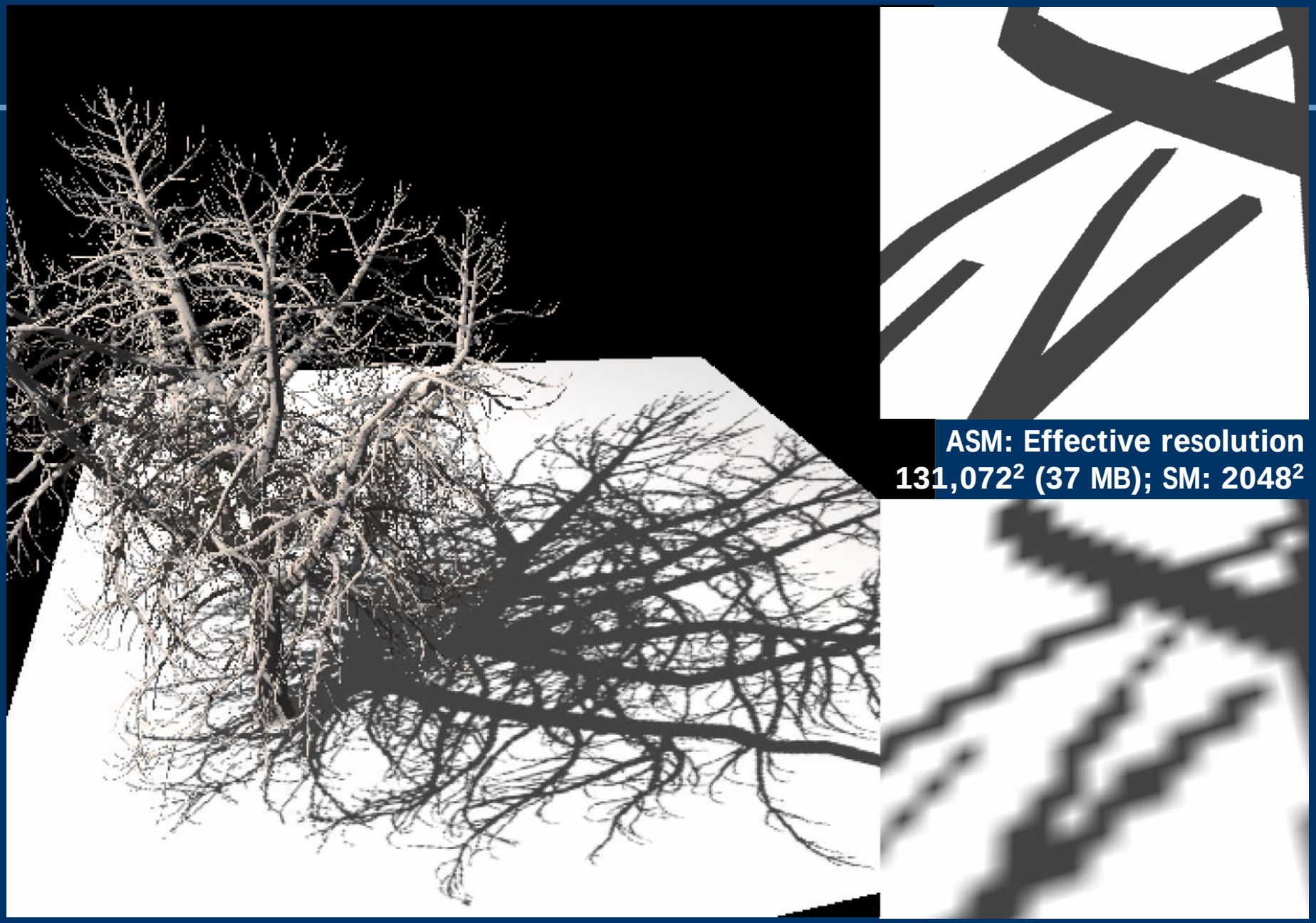
ASM Data Structure Usage

```
float4 main(  uniform VMem2D asm,
              float3 shadowCoord,
              float4 litColor     ) : COLOR
{
      float isInLight = asm.vTex2Ds( shadowCoord );
      return lerp( black, litColor, isInLight );
}

asm.bind_for_read( … );
asm.bind_for_write( … );

asm.alloc_pages( … );
asm.free_page( … );

…
```

Adaptive Shadow Map Algorithm

- **Faithful to Fernando et al. 2001**
- **Refinement algorithm**
  - Identify shadow pixels w/ resolution mismatch (GPU)
  - Compact pixels into small stream (GPU)
  - CPU reads back compacted stream (GPU→CPU)
  - Allocate pages
    - Draw new PTEs into mipmap page tables (CPU→GPU)
  - Draw depth into ASM for each new page (GPU)

Aaron Lefohn
University of California, Davis



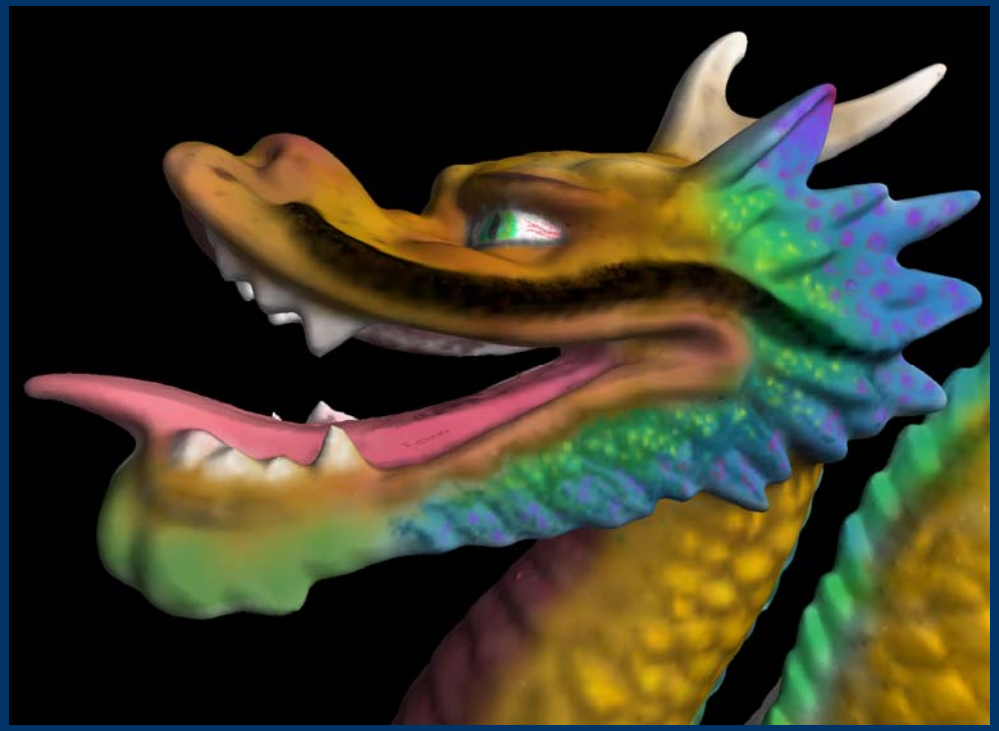
ASM: Effective resolution $131{,}072^2$ (37 MB); SM: 2048^2

[Thanks to Yong Kil for the tree model]

Aaron Lefohn
University of California, Davis

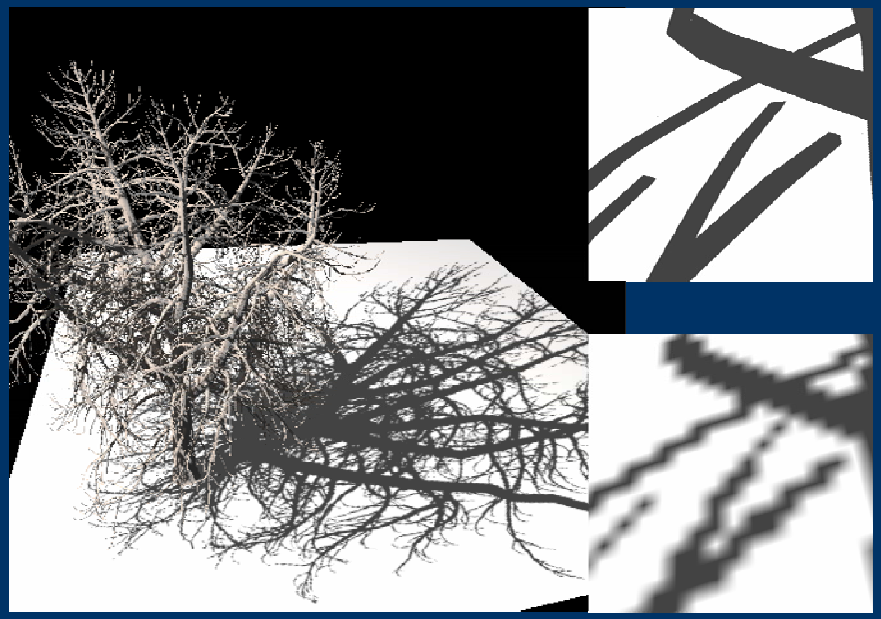
"Octree" 3D Paint

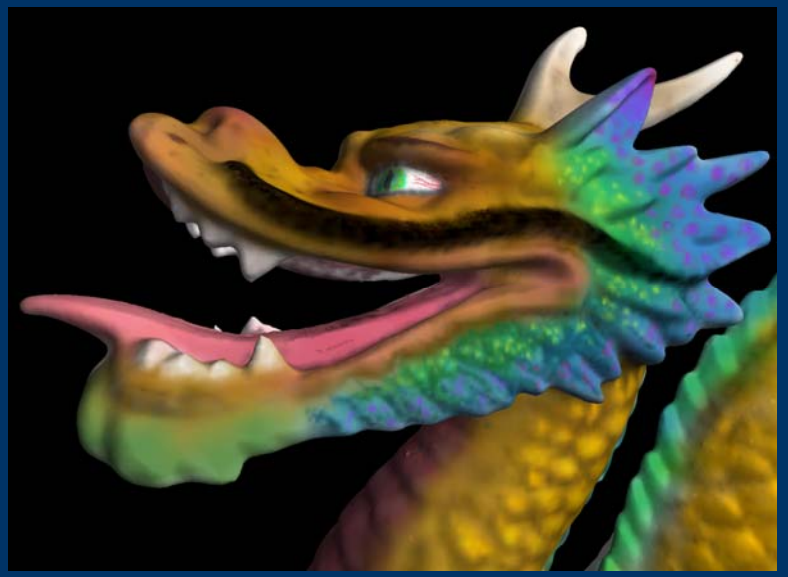
- **Interactive painting on unparameterized 3D surfaces**
- **3D version of ASM data structure**
- **Differs from previous work:**
  - Quadrilinear filtering
  - O(1), uniform access
- **Interactive with effective resolutions between 64^3 and 2048^3**



Aaron Lefohn
University of California, Davis

Demo





Aaron Lefohn
University of California, Davis

ASM Results

- **Effective shadow map resolution up to $131{,}072^2$**
  - 16^2 - 64^2 page size
  - 512^2 - 2048^2 page table
  - 2048^2 - 4096^2 physical memory
  - 20 - 80 MB
- **Performance (45k polygon model)**
  - 15 fps while moving camera (including refinement)
  - 5-10 fps while moving light
- **Lookup time compared to 2048^2 shadow map:**
  - Bilinear filtered: 90% performance of traditional
  - Trilinear filtered mipmapped: 73%

Glift Results

- **Static instruction results**
  - With Cg program specialization

	Glift	By-Hand	Brook
• 1D → 2D	4	3	4
• 3D page table	5	5	
• ASM	9	9	
• Octree	10	9	
• ASM + offset	10	9	

  - Conclusion : Glift structures within 1 instr of hand-coded Cg

Measured with NVShaderPerf, NVIDIA driver 75.22, Cg 1.4a

Overview

- Motivation and previous work
- Abstraction
- Implementation
- Examples
- Conclusions

Aaron Lefohn
University of California, Davis

Summary

- **GPU programming needs data structure abstraction**
  - Separate data structures and algorithms
  - More complex data structures and algorithms
- **Why programmable address translation?**
  - Common pattern in GPU data structures
  - Small amount of code virtualizes GPU memory model

Summary

- **Glift template library**
  - Generic C++/Cg implementation of abstraction
  - Nearly as efficient as hand coding
  - Integrates with OpenGL/Cg
- **Iterator computation model**
  - Generalize GPU computation model
  - Can future rasterizer increment iterators?

Acknowledgements

- Craig Kolb, Nick Triantos, Cass Everitt — NVIDIA
- Fabio Pellacini — Dartmouth

- Adam Moerschell, Yong Kil — UCDavis
  Serban Porumbescu, Chris Co, ….
- Ross Whitaker, Chuck Hansen, Milan Ikits — U. of Utah

- National Science Foundation Graduate Fellowship
- Department of Energy

More Information

- **Paper in ACM Transactions on Graphics (Jan. 2006)**
  - "Glift : Generic, Efficient, Random-Access GPU Data Structures"
- **ACM SIGGRAPH 2005 Sketches**
  - "Dynamic Adaptive Shadow Maps on Graphics Hardware"
  - "Octree Texture on Graphics Hardware"
- **Google "Glift"**
  - http://graphics.cs.ucdavis.edu/~lefohn/