Generic Data Structures for Graphics Hardware

Ph.D. Defense
March 31, 2006

Aaron Lefohn
University of California, Davis
Why GPU Data Structure Abstraction?

Octree Textures

*Image from Benson et al., SIGGRAPH 2002*
Why GPU Data Structure Abstraction?

High-Quality Shadows

Image from Johnson et al.
ACM Transactions on Graphics 2005
Why GPU Data Structure Abstraction?

High-Quality Shadows

*Image from Aila et al. EGSR 2004*
Why GPU Data Structure Abstraction?

Depth-of-Field
Thesis Statement

“A data structure abstraction for graphics processing units (GPUs) can:

• simplify description of new and existing data structures,

• stimulate development of complex GPU algorithms,

• perform equivalently to hand-coded implementations.”
Talk Outline

- Background / Motivation
- Glift
- Applications
  - Quadtree Shadow Maps
  - Octree 3D Paint
  - Heat Diffusion Depth-of-Field
- Conclusions
Desktop Parallel Revolution

“The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software”
Herb Sutter, Dr. Dob’s Journal, 30(3), March 2005
Desktop Parallel Revolution

- Old software model
  - 1) Write code
  - 2) Faster clock speeds increase performance
Desktop Parallel Revolution

- New software model
  - 1) Re-write code to take advantage of parallelism
  - 2) More parallelism increases performance
Desktop Parallel Architectures

- **CPUs**
  - 2 processors (dual core)

- **IBM Cell processor**
  - 9 processors

- **Graphics processing units (GPUs)**
  - Up to 48 parallel processors
But, GPU Programming is Hard

- Cannot write C/C++ code for GPUs
  - Graphics API (OpenGL / DirectX)
  - Store data in “textures”

- GPU computation languages
  - Brook, Scout, Sh

- No data structure abstraction
  - “How do you build a quadtree using glTexImage2D?”
  - “How do you run algorithm on nodes of quadtree?”
General Purpose Computation on GPUs

- Draw screen-sized quad
- Run fragment program over all fragments
- Read data from textures
- Write to texture

*Figure courtesy of John Owens*
Selected Publications

- **Glift: Generic, Efficient, Random-Access GPU Data Structures**
  - Aaron Lefohn, Joe Kniss, Robert Strzodka, Shubhabrata Sengupta, John Owens

- **Resolution Matched Shadow Maps**
  - Aaron Lefohn, Shubhabrata Sengupta, John Owens
  - To be published...

- **Interactive Depth of Field Using Simulated Diffusion on a GPU**
  - Michael Kass, Aaron Lefohn, John Owens
  - To be published...
Selected Publications

• Implementing Efficient Parallel Data Structures on GPUs
  • Aaron Lefohn, Joe Kniss, John Owens
  • GPU Gems II: Programming Techniques for High-Performance Graphics and General-Purpose Computation, Addison Wesley, Ch. 33, pp 521-545, 2005

• Dynamic Adaptive Shadow Maps on Graphics Hardware
  • Aaron Lefohn, Shubhabrata Sengupta, Joe Kniss, Robert Strzodka, John Owens
  • Technical Sketch at ACM SIGGRAPH, 2005

• Octree Textures on Graphics Hardware
  • Joe Kniss, Aaron Lefohn, Robert Strzodka, Shubhabrata Sengupta, John Owens
  • Technical Sketch at ACM SIGGRAPH, 2005
Talk Outline

• **Background / Motivation**
• **Glift**
• **Applications**
  • Quadtree Shadow Maps
  • Octree 3D Paint
  • Heat Diffusion Depth-of-Field
• **Conclusions**
# 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
• Problems
  • Code is tangled mess of algorithm and data structure access
  • Algorithms expressed in GPU memory domain
  • No code reuse
GPU Data Structures

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

![Diagram](https://via.placeholder.com/150)

- Brook
- Scout
- Sh
- STL
- C++
- Cg
- OpenGL
- ???
Simple Example

• CPU (C++)

```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] );
        }
    }
}
```
We Want To Transform This...

- **GPU (Cg)**

```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 );
}
```
We Want To Transform This...

- GPU (Cg and C++)

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

    return addr3D;
}

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

```cpp
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 Glipt)

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() );
}
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
Glift Design Challenges

- Cross-processor memory model
- Cross-language memory model
- Parallel operations
- Primitive GPU languages
- Efficiency
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
  - texND(tex, addr)
  - varying floatN stream
  - stream

- data structure param declaration
- random-access read
- stream parameter declaration
- stream read
GPU Data Structure Abstraction

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

Virtual representation of memory: 3D grid

<table>
<thead>
<tr>
<th>Translation</th>
<th>Translation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D native mem</td>
<td>2D slices</td>
<td>Flat 3D texture</td>
</tr>
</tbody>
</table>
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
    
    \[
    \text{translate}(\ldots) \\
    \text{translate\_range}(\ldots)
    \]
Data Structure Examples

- Brook streams (Buck et al. 2004)

1D Virtual 2D Physical
## Data Structure Examples

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brook Streams</td>
<td>(Buck et al. 2004)</td>
</tr>
<tr>
<td>Physical address</td>
<td>2D</td>
</tr>
<tr>
<td>Virtual address</td>
<td>N-D</td>
</tr>
<tr>
<td>Address translator</td>
<td>ND-to-2D</td>
</tr>
<tr>
<td></td>
<td>Analytic</td>
</tr>
<tr>
<td></td>
<td>O(1) memory</td>
</tr>
<tr>
<td></td>
<td>O(1) compute</td>
</tr>
<tr>
<td></td>
<td>Uniform consistency</td>
</tr>
<tr>
<td></td>
<td>Total, uniform mapping</td>
</tr>
</tbody>
</table>
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?
Iterators

- Abstract data access and traversal

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

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

- **Permission**
  - Read-only, write-only, read-write

- **Access pattern**
  - Single, neighborhood, random, ...

- **Traversal**
  - Forward, backward, parallel range
Iterator Contribution

• Parallel iteration model for GPU computation
  • Stream model is subset
  • Generalizes over complex data structures
  • Memory access patterns are first-class primitives
  • Connection to CPU libraries such as STL
Simple Example

- 3D Array with 2D physical memory

CPU (C++)

```cpp
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: GPU Shader with Glift

Cg Usage

```c
float3 main( uniform VMem3D srcData,
             AddrIter3D iter ) : COLOR
{
    float3 va = iter.value();
    return srcData.vTex3D( va - float3(1,1,1) );
}
```
Example: Glift Data Structures

C++ Usage

```cpp
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);
```
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

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

- All other compilation/loading/binding identical to standard shader
Cg Compilation Example

- **Cg code**

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

- **C++ code**

```c++
typedef OctreeGPU<vec4ub> octree_type;
GliftType type = cgGetTemplateType<octree_type>();
CGprogram prog = cgCreateProgram(...);
prog = cgInstantiateParameter(prog, "octree", type);
cgCompileProgram(prog);
```
**Glift Code Efficiency**

- Number of GPU cycles required for data structure access

<table>
<thead>
<tr>
<th></th>
<th>Glift</th>
<th>By-Hand</th>
<th>Brook</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D → 2D</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3D page table</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>ASM</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Octree</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>ASM + offset</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

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

Measured with NVShaderPerf, NVIDIA driver 75.22, Cg 1.4a
Talk Outline

• **Background / Motivation**
• **Glift**
• **Applications**
  • Quadtree Shadow Maps
  • Octree 3D Paint
  • Heat Diffusion Depth-of-Field
• **Conclusions**
Motivation
Motivation

32,7682 quadtree shadow map
Shadow Map Overview

- Williams, 1978
  - Depth image rendered from the light position
Shadow Map Overview

- Shadow lookup
Problems: Aliasing

- Incorrect shadow map sampling
  - Objects close to eye are far from light
  - Occluder parallel to light but perpendicular to eye
- Depth precision aliasing

Image from NVIDIA GDC 2004 presentation, Gary King
Quadtree Shadow Maps

“Adaptive Shadow Maps,” Fernando et al., SIGGRAPH 2001
“Dynamic Adaptive Shadow Maps on GPUs,” Lefohn et al., SIGGRAPH Sketch 2005
“Resolution-Matched Shadow Maps,” Lefohn et al., submitted to SIGGRAPH 2006
Interactive Quadtree Shadow Maps?

“The adaptive shadow map approach addresses the resolution problem by using a hierarchical grid structure instead of the standard `flat' shadow map. A great improvement to the shadow quality is gained, but it is currently not possible to map this approach to graphics hardware. Therefore, this approach is slow and not suitable for real-time applications.”

Eurographics Symposium on Rendering, 2004
Data Structure Challenges

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

- Shadow map coordinates

(0,0)  (1,0)

(0,1)  (1,1)
Quadtree Physical Domain

- Paged 2D texture memory
Quadtree Address Translator

- Mipmapped page table
Quadtree Data Structure Usage

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

quadtree.bind_for_read( ... );
quadtree.bind_for_write( ... );

quadtree.alloc_pages( ... );
quadtree.free_page( ... );

...
```

Aaron Lefohn
University of California, Davis
Algorithm

- Generate shadow page request for all pixels
- Allocate pages
- Write shadow data
Algorithm

- **Generate shadow page request for all pixels**
  - Render \((s, t, lod)\) from camera
  - Request one page per contiguous region in image
  - Remove NULL page requests (compact)
  - Remove redundant page requests (uniquify)
  - Transfer requests to CPU

- **Allocate pages**
- **Write shadow data**
Algorithm Contribution #1

- Fast Parallel Prefix (scan) on GPUs
  - Up to 4x faster than previous implementation
  - $O(n)$ vs. $O(n \log n)$
  - Hybrid step- and work-efficient
Algorithm Contribution #2

- Resolution Matched Shadow Maps
  - No iterative refinement needed
  - Strong coherence between image and shadow space

32,768² ASM
1.8 fps, 2200 pages

32,768² RMSM
20.6 fps, 3300 pages
Algorithm

- **Generate shadow page request for all pixels**
- **Allocate pages**
  - Render to address translator (page tables)
- **Write shadow data**
Algorithm

- Generate shadow page request for all pixels
- Allocate pages
- Write shadow data
  - Render depth into physical memory
Demo
# Quadtree Shadow Map Performance

<table>
<thead>
<tr>
<th></th>
<th>SM</th>
<th>ASM</th>
<th>RMSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dynamic scenes (fps)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coherent receiver:</td>
<td>40</td>
<td>2-3</td>
<td>20-25 fps</td>
</tr>
<tr>
<td>Incoherent receiver:</td>
<td>16</td>
<td>5-7</td>
<td>6-7  fps</td>
</tr>
<tr>
<td><strong>Static scenes (fps)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coherent receiver:</td>
<td>70</td>
<td>20</td>
<td>30-35 fps</td>
</tr>
<tr>
<td>Incoherent receiver:</td>
<td>20</td>
<td>7-10</td>
<td>15   fps</td>
</tr>
</tbody>
</table>

1024^2 image  
*Furball scene, 48,000 line segments (4,000 hairs)*  
32,768^2 maximum effective resolution quadtree  
*NVIDIA GeForce 7800 GTX*  
2.4 GHz AMD Athlon  
*PCI Express (16x)*
Talk Outline

- Background / Motivation
- Glift
- Applications
  - Quadtree Shadow Maps
  - Octree 3D Paint
  - Heat Diffusion Depth-of-Field
- Conclusions
Octree Textures

- Benson & Davis
- DeBry et al.
- Christensen & Batali
What is an Octree?

*Image from Lefebvre et al., 2005*
Glift “Octree” 3D Paint

- Interactive painting on unparameterized 3D surfaces

- 3D version of quadtree data structure
  - 3D virtual domain
  - 3D physical domain
  - 3D page table address translator

- Differs from previous work:
  - Quadrilinear filtering
  - $O(1)$, uniform access
Octree Data Structure Usage

```c
float4 main( uniform VMem3D octree,
    float3 texCoord ) : COLOR
{
    return octree.vTex3D(texCoord);
}
```

```c
octree.bind_for_read( ... );
octree.bind_for_write( ... );
```

```c
octree.alloc_pages( ... );
octree.free_page( ... );
```

...
Results
Results

Octree 3D Paint
(2048^3 Texture)
Performance

- **Read access is geometry bound (70+ fps)**
  - $8^3 - 16^3$ pages results in coherent accesses

- **Brushing**
  - Interactive with small brushes
  - Slows down with large brushes

- **Memory usage example**
  - $2084^3$ maximum effective resolution
  - $128^3$, RGBA8 page table (9.2 MB)
  - $128^3$, RGBA8 physical memory (8 MB)
Talk Outline

• Background / Motivation
• Glift
• Applications
  • Quadtree Shadow Maps
  • Octree 3D Paint
  • Heat Diffusion Depth-of-Field
• Conclusions
Depth of Field
Circle of Confusion (CoC)

\[ \frac{1}{P} + \frac{1}{I} = \frac{1}{F} \]

C = \left| \frac{A F (P - D)}{D (P - F)} \right|

C - Circle of Confusion
A - Aperture
F - Focal Length
P - Plane in Focus
D - Object Distance
I - Image Distance

Figure from Demers et al.
GPU Gems, p. 376
Depth of Field (DOF) in Graphics

- **Accurate simulation**
  - Sample from multiple positions on “lens”
  - Offline rendering technique (slow!)

- **Post-process approximation**
  - Compute DOF from RGBZ image
  - Spread pixel color out over area based on CoC
Post-Process DOF

- **Splat each pixel with size based on CoC**
  - Used in software film preview, not interactive
    - Sort all pixels
    - 32-bit floating-point blending
    - Hard to conserve energy

- **Convolution: Blur each pixel by size of CoC**
  - Lots of GPU hacks that do this
  - Problems
    - Aliasing, limited blur size, edge bleed
Our Goal

• Better convolution method
  • Efficient, variable-width blur
  • No aliasing
  • Unlimited blur size
  • No edge bleeding
Depth-of-Field Contributions

• Heat diffusion model for interactive DOF
• Direct tridiagonal linear solver on GPU
• Fast recursive filters on GPU

(Glifit iterators guided refactoring of solver)
Heat Diffusion Model

- **Anisotropic heat equation**
  - Variable-width, Gaussian blur
  - Implicit solver requires constant work per pixel

- **Idea**
  - Input image is “initial heat distribution”
  - Define “material model” based on CoC
  - Obtain DOF result by allowing heat to diffuse
Solving Heat Equation

- Use separable, implicit solver
  - One tridiagonal system for each row/column
  - 100s - 1000s of tridiagonal matrices

\[
\begin{pmatrix}
  b_1 & c_1 & 0 \\
  a_2 & b_2 & c_2 \\
  a_3 & b_3 & c_3 \\
  \vdots & \vdots & \ddots \\
  0 & a_n & b_n
\end{pmatrix}
\begin{pmatrix}
  h_1 \\
  h_2 \\
  h_3 \\
  \vdots \\
  h_n
\end{pmatrix}
= 
\begin{pmatrix}
  h_1^0 \\
  h_2^0 \\
  h_3^0 \\
  \vdots \\
  h_n^0
\end{pmatrix}
\]
GPU Challenge #1

- **Need direct tridiagonal linear solver**
  - CPU methods use forward- and back-substitution
    ```
    for (int j = n - 2; j >= 0; --j) {
        u[j] -= gam[j+1] * u[j+1];
    }
    ```
  - “Impossible” to parallelize
    - Recurrence relation
    - No GPU iterators support this access pattern
GPU Challenge #2

- **Cyclic reduction**
  - Parallel solver algorithm
  - “Requires” unsupported GPU iterators
    - Write-only, random-access iterator (scatter)
    - Ragged iteration domain
GPU Solution

- Refactor cyclic reduction in terms of GPU iterators
  - No scatter
  - Dense iteration domains
  - GPU iterators: Neighbor input, stream output

- Data structure
  - Array of tridiagonal matrices
  - Each matrix is array of \((a_i, b_i, c_i)\)

- Final computation is simply parallel prefix (Scan)!
Interactive Depth of Field
Interactive Depth of Field
Interactive Depth of Field
Interactive Depth of Field
# Performance Results

- Speed based only on image size, $O(n)$
- Bound by speed of GPU memory

<table>
<thead>
<tr>
<th>Image size</th>
<th>Framerate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$256^2$</td>
<td>85-90 fps</td>
</tr>
<tr>
<td>$512^2$</td>
<td>35-40 fps</td>
</tr>
<tr>
<td>$1024^2$</td>
<td>10-11 fps</td>
</tr>
</tbody>
</table>

*NVIDIA 7800 GTX, 2.4 GHz Athlon*
Main Contribution

• Efficient GPU recursive filter
  • Widely regarded as impossible
    (except by Simon Green at NVIDIA)
  • Fully utilizes GPU parallelism
  • Large blurs with constant work per pixel
  • Many other potential applications
Talk Outline

• Background / Motivation
• Glift
• Applications
  • Quadtree Shadow Maps
  • Octree 3D Paint
  • Heat Diffusion Depth-of-Field
• Conclusions
Future Work

• Programming model for commodity parallelism
  • Language?
  • Library?
  • Convergence of CPU, GPU, Cell?
Future Work

- Richer set of memory access patterns
  - What else beside single, neighbor, random?
  - Critical for architectures like IBM Cell?
Future Work

- Additional GPU data structures
  - Set
  - Hash
  - Linked list
  - Graphs
  - ...
Future Work

- **Generic GPU algorithms**
  - Scan
  - Sort
  - Compact
  - ...

Aaron Lefohn
University of California, Davis
Conclusions

• **GPU data structure abstraction**
  - Enable new data structures
  - Spur development of complex GPU algorithms
  - Parallel iteration model

• **Applications**
  - Dynamic quadtreers/octrees possible on GPUs
  - Recursive filters possible on GPUs
  - Data-parallel algorithms and interactive rendering
Conclusions

- **Final note**
  - If a problem looks impossibly sequential, use scan
Acknowledgements

- John Owens
- Committee: Nina Amenta, Bernd Hamann, Ken Joy, Oliver Staadt
- Coauthors: Michael Kass, Joe Kniss, Shubhabrata Sengupta, Robert Strzodka
- Dartmouth: Fabio Pellacini
- NVIDIA: Nick Triantos, Craig Kolb, Cass Everitt, Mark Harris, Ian Buck, Randima Fernando, David Kirk
- Los Alamos: Jim Ahrens, Pat McCormick
- Stanford: Daniel Horn, Kayvon Fatahalian, Mike Houston
- Pixar: Kiril Vidimce, Mark Meyer, Tony DeRose, Rob Cook
- Neoptica: Craig Kolb, Matt Pharr
- IDAV: Chris Co, Serban Porumbescu, Yong Kil, Adam Moerschell, Louis Feng, Taylor Holliday

- Funding
  - National Science Foundation Graduate Fellowship

- Karen, Kaia, Allen, and Phyllis Lefohn
Questions?

• More Information?
  • Google “Glift”
  • Google “Aaron Lefohn”