Beyond Programmable Shading Introduction

Aaron Lefohn - Intel / University of Washington

Mike Houston – AMD / Stanford

Course Goals, To Learn...

- The state-of-the-art in real-time rendering hardware, programming models, and algorithms

- The open research problems in real-time rendering algorithms and programming models

Course Goals (Detail). To Learn...

- Architecture / programming models
  - How does a GPU work and what is difference between CPU and GPU?
  - How to use all CPU and GPU FLOPs
  - Parallel programming models and algorithms for graphics on CPU and GPU
  - Open research problems in parallel programming models for graphics

- Rendering
  - The modern real-time rendering pipeline (DirectX11)
  - Latest CPU and GPU parallel programming models
  - How to solve rendering problems by mixing traditional 3D pipeline with your own task- or data-parallel algorithms
  - State-of-the-art in real-time rendering algorithms
  - Open research problems in real-time rendering

Course History

- Beyond Programmable Shading SIGGRAPH courses
  - 2008, 2009, 2010
- Beyond Programmable Shading Stanford course
  - Spring 2010 (5 student “pilot” course)
- Winter 2011
  - First offered at University of Washington
  - To be offered at Stanford in spring 2011

Instructor Biographies

- Aaron Lefohn
  - Director of research for the Advanced Rendering Technologies team at Intel
    - Real-time rendering and rendering programming model research
  - Neoptica, graphics startup acquired by Intel in 2007
  - Rendering R&D at Pixar
  - Ph.D. University of California, Davis with John Owens
  - Early GPGPU researcher (beginning in 2002)
- Mike Houston
  - Principal Architect for AMD Accelerated Parallel Processing
    - One of the OpenCL and Fusion SW technical leads
    - Heavily involved in GPU and Fusion hardware for parallel computation
  - Ph.D. Stanford with Pat Hanrahan
  - Early GPGPU researcher (beginning in 2002)

Syllabus Overview

- Section 1: Review modern real-time rendering
  - DirectX 9 pipeline (ca. 2003)
  - DirectX 11 pipeline (ca. 2009)
  - Rendering effects / terms (shadow maps, cube maps, *-buffers, …)

- Section 2: GPU architecture and parallel programming
  - Interleaved lectures on arch., prog. models, rendering applications
  - Focus on GPU (because it is new to you), but discuss CPU and GPU

- Section 3: Putting it all together
  - Advanced rendering techniques that use 3D pipeline plus parallel algorithms
  - New real-time rendering pipelines
  - How games use these architectures, programming models, and techniques
  - Open research problems

Assignment (Rough) Overview

- Assignment 1 (15%, individual)
  - Warm-up “programmable shading” project using DirectX11
  - I’ll provide skeleton code, you write shaders and add rendering passes
- Assignment 2 (35%, individual)
  - Heterogeneous CPU + GPU rendering project using CPU and GPU languages
  - I’ll provide skeleton code, but this assig. will involve a lot of coding
- Assignment 3, term project (50%, ~2 people / project)
  - “Render pretty pictures using all CPU and GPU cores”
  - New solutions to open problems in rendering or parallel algorithms

Logistics I

- Computers
  - Each registered student will be loaned a new computer for the term
  - Location
    - Grad students may keep machine at your UW desk (not at home)
    - Other machines will be in Sieg 327
  - Hardware details
    - 12 CPU cores / 24 CPU threads (3.33 GHz Xeon) with 12 GB of RAM
    - AMD 5870 GPU with 1 GB of RAM
  - Software
    - Intel Parallel Studio (including Cilk and many other parallelism tools)
    - Microsoft DirectX SDK
    - AMD StreamSDK (OpenCL)
    - AMD GPU drivers (OpenGL, Direct3D, DirectCompute)
  - HW and SW donated by Intel, AMD, and Microsoft

Logistics II

- We will have several guest lecturers
  - Natasha Tatarchuk from Bungie
  - Andrew Lauritzen from Intel
  - Likely a couple of others later in the term (TBA)

- Website
  - UW: http://www.cs.washington.edu/education/courses/cse558/11wi/

- Mailing list
  - UW: https://mailman.cs.washington.edu/mailman/listinfo/cse558

A Brief History of The Real-Time Rendering Pipeline

Early Software Renderers: Fixed Function

- 1970s through mid-1980s, most graphics pipelines were fixed-function
- Must change core rendering code to change geometry/lighting/surface model

Mid’80s – Early 90s: Programmable Shading

- Most of graphics pipeline is fixed-function, highly optimized architecture, but shaders allow users to customize small portions of pipeline with simple language
  - Shade Trees, Cook 1984
  - An Image Synthesizer, Perlin 1985
  - Renderman Shading Language, Hanrahan/Lawson 1990

Cook Approach

Shader:

- Basic operations: dot products, norms, etc.
- Operations organized into trees

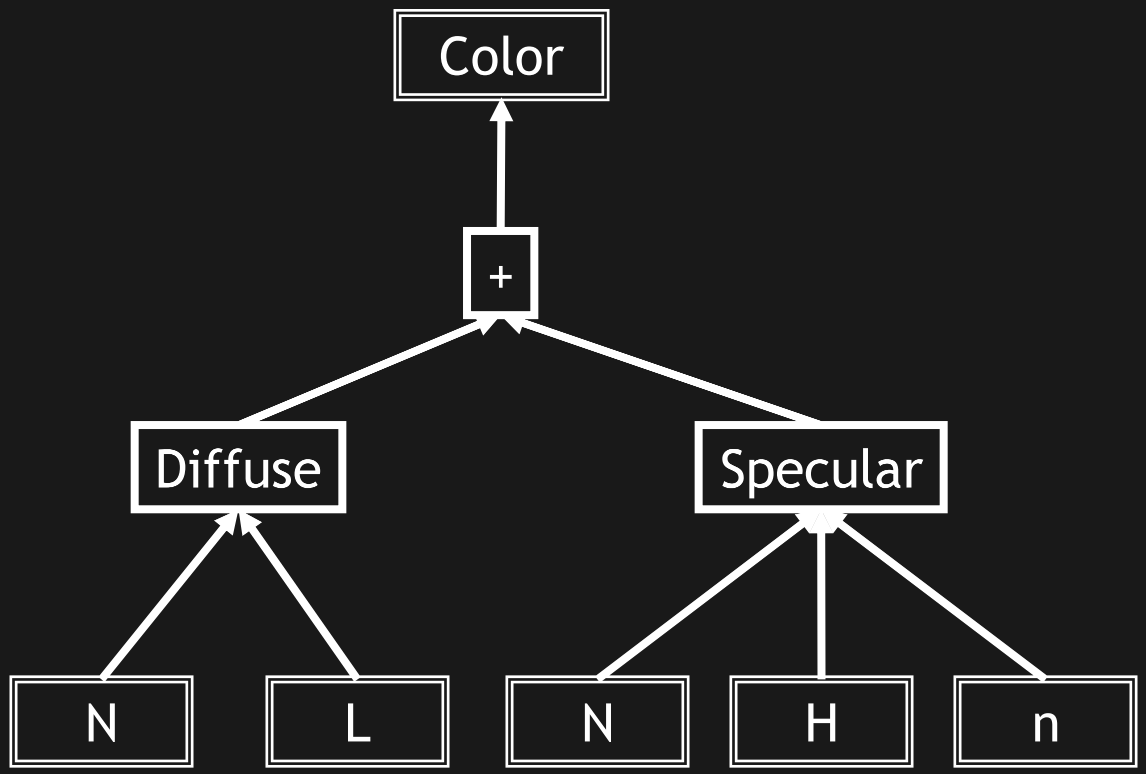
- Separated light source specification, surface reflectance, and atmospheric effects
- Multiple trees: shade trees, light trees, atmosphere trees, displacement maps

- Simple language

Slide courtesy of John Owens, UC Davis

Shade Trees

Phong shading model:



Slide courtesy of John Owens, UC Davis

Cook: Arbitrary Trees

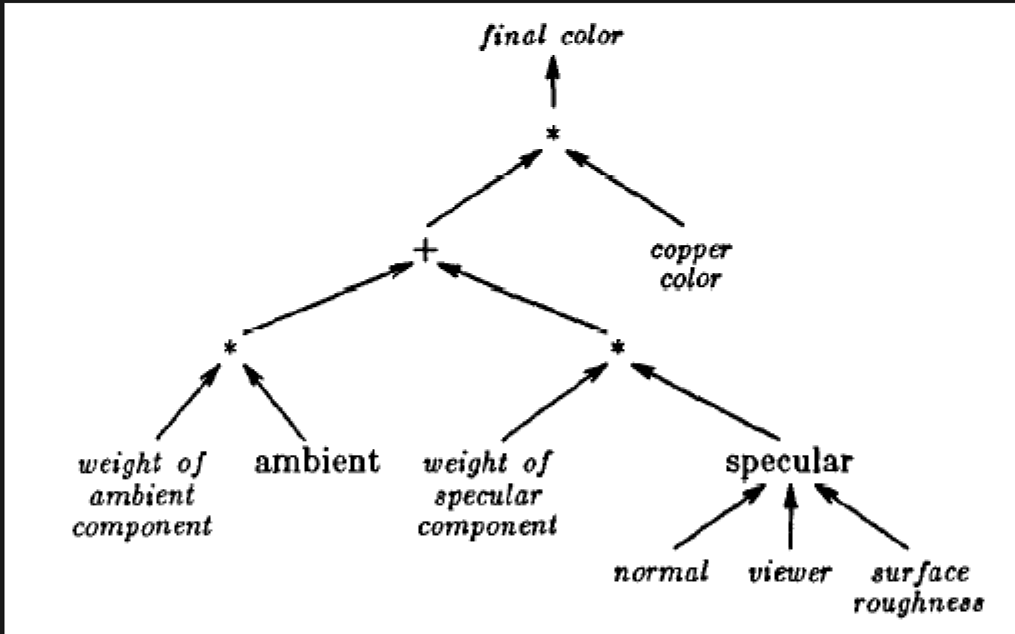
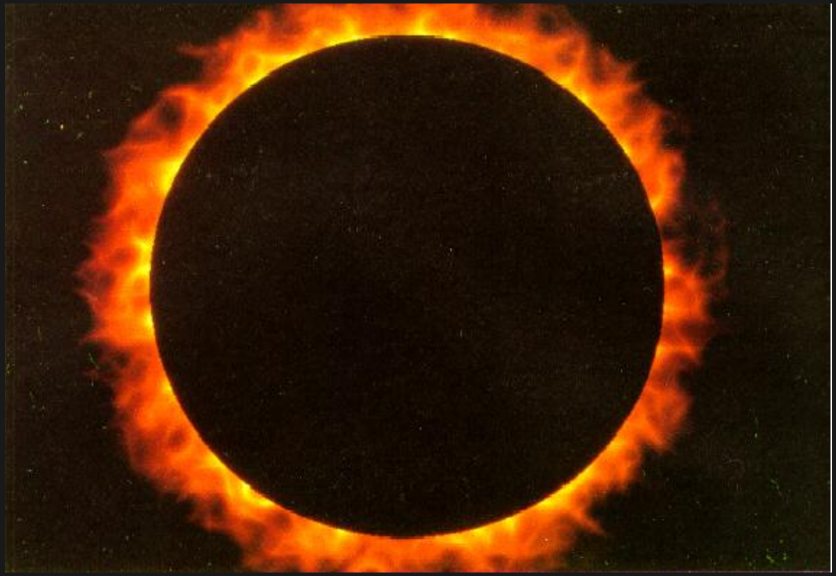
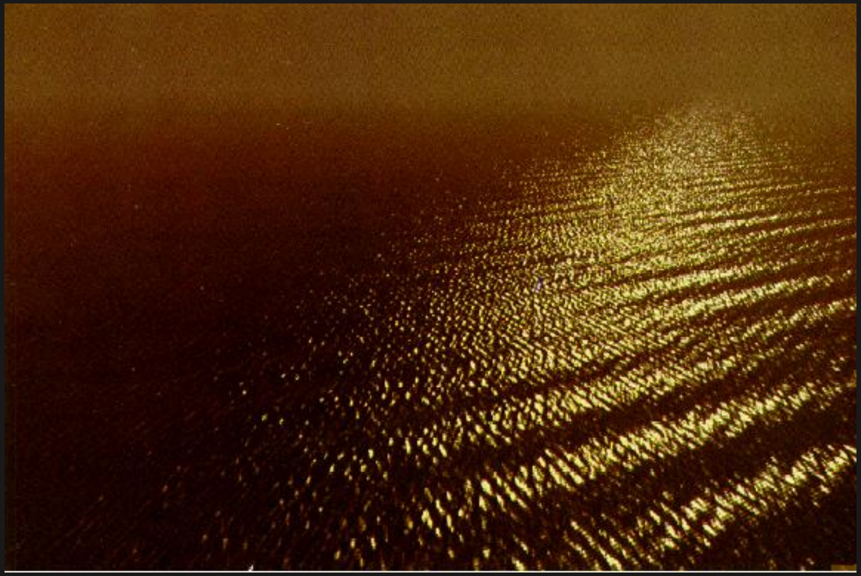


Figure 1a. Shade tree for copper.

Slide courtesy of John Owens, UC Davis

Images from Perlin





Contributions

Cook: Separates / modularizes factors that determine shading

- Geometric
- Material
- Environmental

Perlin: Language definition, conditionals, …

Leads to …

RenderMan—Hanrahan/Lawson

1. Abstract shading model based on optics for either global/local illumination
2. Define interface between rendering program and shading modules
3. High-level shading language

Three main kinds of shaders—light source, surface reflectance, volume

Slide courtesy of John Owens, UC Davis

Shading Language Summary

- Expert rendering engineers write highly optimized rendering architecture
- Shading languages allow non-expert users to customize renderer
- Consequence
  - Renderers specify their own compilers, language runtimes, memory models
  - User’s shading code runs as inner-most loop---JIT ends up being very important

A Few (but not all) Seminal Papers

• Depth Buffer: Catmull	1978
• A-Buffer: Carpenter	1984
• Shade Trees: Cook	1984
• Alpha Blending: Porter and Duff	1984
• REYES: Cook, Carpenter, Catmull	1987
• Renderman: Hanrahan, Lawson	1990
• Reality Engine Graphics: Akeley	1993
• …	

Early 90s, Interactive Rendering Started Over

Wolfenstein 3D, 1992

Doom I, 1993

- Interactive software renderering (no GPUs yet)
- NOTE: SGI was building interactive rendering supercomputers, but this was beginning of interactive 3D graphics on PC

By Late 90s, Graphics Hardware Emerging

- “Hey, let’s build hardware to make a highly constrained graphics pipeline go really fast”
- 3DFX, NVIDIA, ATI, and countless others
- Eventually replaced SGI’s supercomputers with plug-in boards for PC
- OpenGL and Direct3D gained dominance as they provided the *only* programming interface to GPUs

OpenGL and DirectX (90’s to early 2000s)

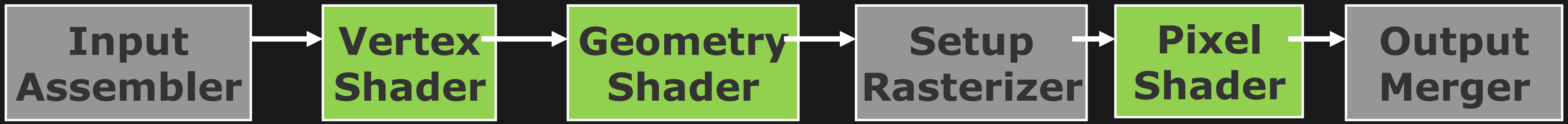
- Fixed function pipeline
  - Configure options via API
  - Fixed per-vertex lighting model
  - Fixed vertex transforms
  - Limited texturing

- Rendering pipeline designed by expert HW architects with little programmability available to end-user

2001+: GPUs “Rediscover” Programmable Shading

- NVIDIA GeForce 3 and ATI Radeon 8500
- Programmable vertex computations
  - Up to 128 instructions
- Limited programmable fragment computations
  - 8-16 instructions

2008 (DirectX 10) Programmable Pipeline



- High-level shading languages
  - GL Shading Language (GLSL) for OpenGL
  - High Level Shading Language (HLSL) for DirectX
- 1000s of instructions permitted per stage
- Flow control, integers, arrays, temporary storage, large number of textures, …

Beyond Programmable Shading: Parallel Programming for Graphics

Completing the Circle

- Beginning in 2001/2002, researchers realized that programmable GPUs could do more than graphics
  - GPUs were becoming data-parallel co-processors
  - A research field was born: General Purpose Programming on GPUs (GPGPU)
  - Scores of papers about data-parallel algorithms on GPUs
    - Finance, physical simulation, medical imaging, …

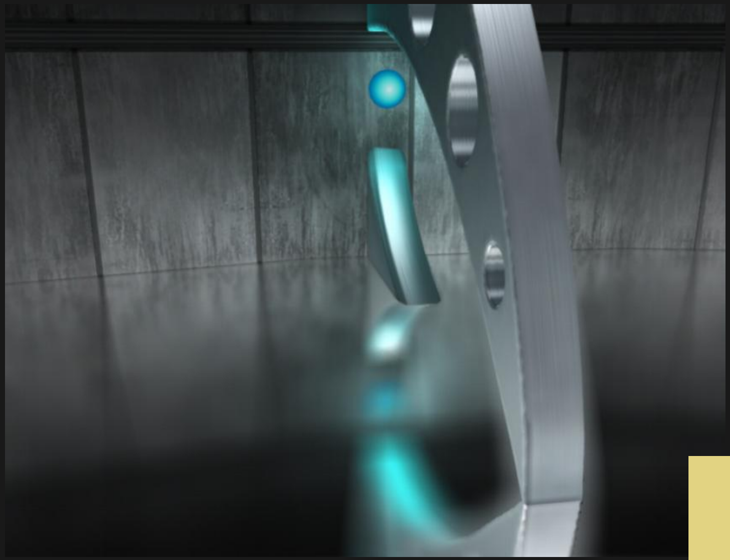
Non-Graphics GPU Programming Models

- From GPGPU, arose parallel programming models that let users program GPUs without using the 3D APIs (OGL / D3D)
  – Brook, Sh / RapidMind, PeakStream, CUDA, OpenCL, DirectCompute, …
- Focus on data-parallelism but task-parallelism is coming
  – Nascent task parallelism in OpenCL
  – Researchers have found ways to build task systems in GPU compute languages

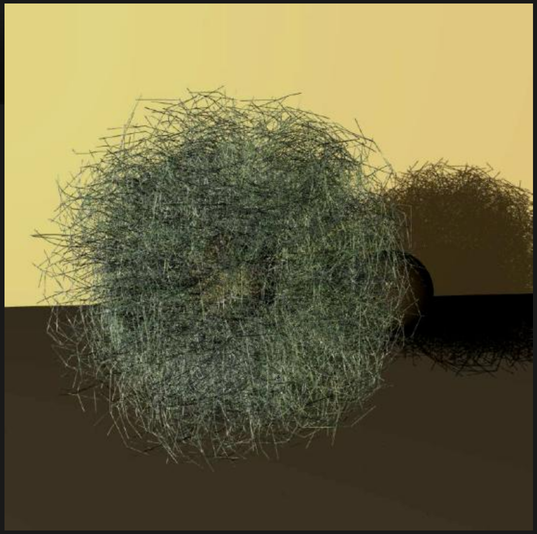
“The Killer App of GPGPU is Graphics”

- But then researchers starting writing rendering papers that combined data-parallel GPU algorithms with the GPU rendering pipeline
  - Ray tracing and photon mapping, Purcell 2002-2003
  - Summed Area Table Generation, Hensley 2005
  - Approximate global illumination, Bunnell 2005
  - Resolution Matched Shadow Maps, Lefohn 2007
  - …

Early “Beyond Programmable Shading”



“Fast Summed-Area Table Generation and its Applications,”
Hensley et al., Eurographics 2005



“Resolution Matched Shadow Maps,”
Lefohn et al., ACM Transactions on Graphics 2007

“Dynamic Ambient Occlusion and Indirect Lighting,” Bunnell, GPU Gems II, 2005

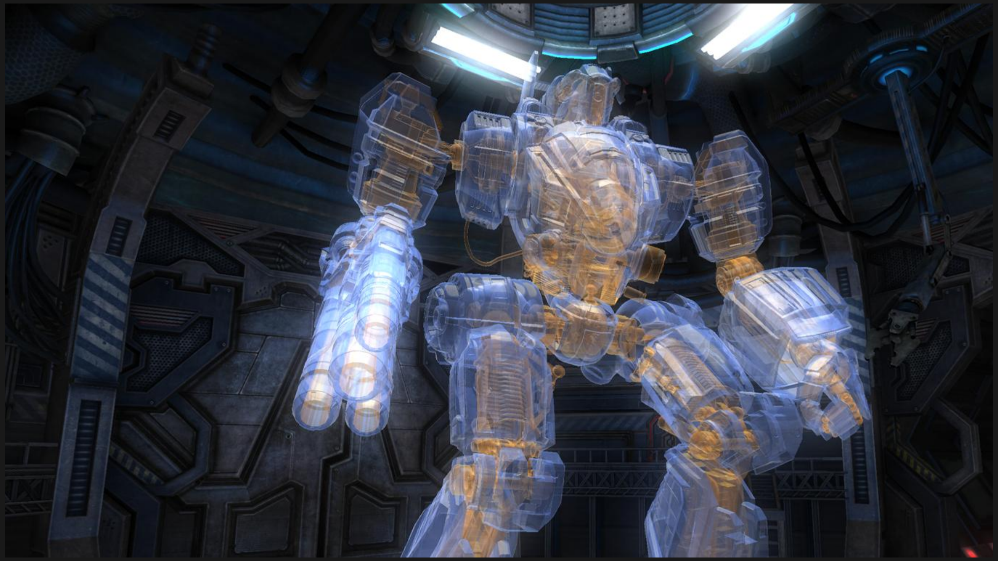
Today: Creating New Rendering Algorithms

- What?
  - Design interactive rendering algorithms that
    - Adapt dynamically by performing per-frame analysis of rendering results
    - Build dynamic per-frame data structures
    - Use an alternate graphics pipeline

- How?
  - Tools
    - Use OpenGL/Direct3D for “standard rendering” portion of algorithm
    - Use OpenCL/DirectCompute for “GPU parallel algorithm” portion
    - Use Cilk, OpenCL, ArBB, … for “CPU parallel algorithm portion”

  - Graphics programming uses mix of
    - Data-parallelism
    - Task-parallelism
    - Pipeline parallelism

"Sample Distribution Shadow Maps,"
Lauritzen et al., I3D 2011
"Render-analyze-render"



"Real-Time Concurrent Linked List Construction on the GPU,"
Yang et al., EGSR 2010
"Render to user-defined data structure"

Building New Real-Time Rendering Pipelines

"OptiX: A General Purpose Ray Tracing Engine", Parket et al., *SIGGRAPH 2010*

"Real-Time Stochastic Rasterization on Conventional GPU Architectures," *McGuire et al., High Performance Graphics 2010*

Mainstream “Beyond Programmable Shading”

- In late 2009, Microsoft and Apple/Khronos released cross-architecture APIs for programming GPUs in ways other than through the 3D rendering API
  - DirectX11: Direct3D plus DirectCompute
  - OpenCL, interoperates with OpenGL
- DirectCompute designed to work closely with 3D rendering API
  - Same language (HLSL)
  - Same CPU-side API
  - Same memory space

DirectCompute in Graphics Products (as of August 2010)

- Screen-Space Ambient Occlusion
  - BattleForge
  - Colin McRae Dirt 2
- Depth of Field
  - Metro 2033
  - Just Cause 2
- FFT Lens Effects
  - FutureMark 3DMark 11

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

DirectCompute in Shipping Graphics Games/Demos

DirectCompute Use in Real-Time Rendering Products

Ambient Occlusion

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Screen-Space Ambient Occlusion

- Conventional technique uses pixel shaders
  - http://sites.google.com/site/perumaal/ao.pdf

- DirectCompute shaders enable more control of convolution filter cache

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

HDSSAO Off

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Image credit:
Codemasters

HDSSAO On

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Image credit:
Codemasters

DirectCompute Use in Real-Time Rendering Products

Depth of Field

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Diffusion Post-Process Depth-of-Field

- Model the problem as a heat-flow simulation
  - Blur radius analogous to heat distance traveled
  - Controlled by ‘thermal conductivity’ of image, which is defined by distance from focal plane
  - Solve thousands of tridiagonal linear systems in parallel each frame

- Original work at Pixar
  - http://graphics.pixar.com/library/DepthOfField/paper.pdf
  - Michael Kass, Aaron Lefohn, John Owens

- Metro 2033
  - GDC 2010 Presentation by OlesShishkovtsov, 4A Games and AshuRege, NVIDIA
  - http://developer.nvidia.com/object/gdc-2010.html#metro

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Image credit:
A4 Games

DirectCompute Use in Real-Time Rendering Products

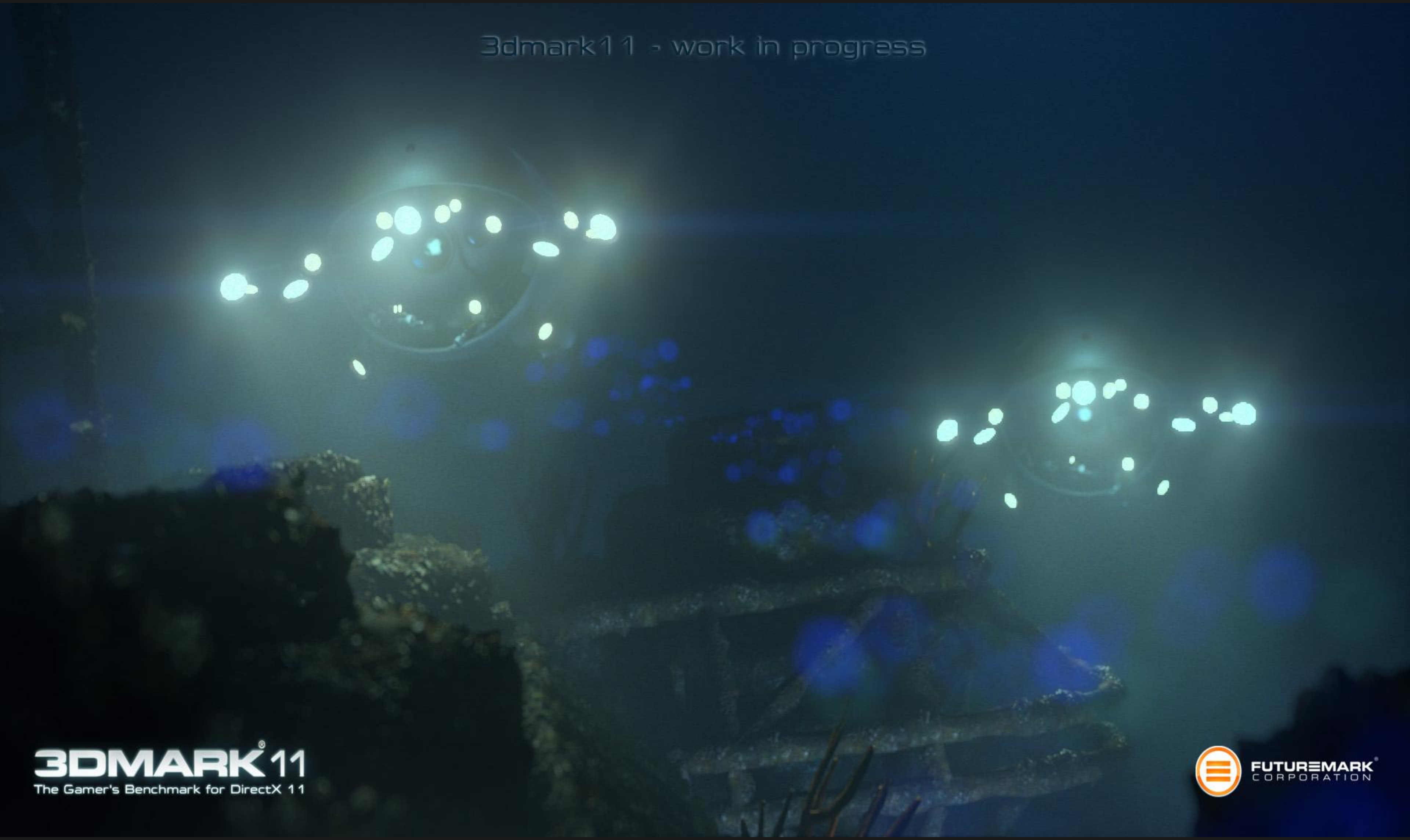
FFT Lens Effects

FFT Lens Effects in 3DMark11

- FFT rendered image
- Apply lens effect in frequency domain
- Inverse FFT

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010



Slide from Chas Boyd,
Beyond Programmable Shading, SIGGRAPH 2010

Recap:

How did/do programmers create new real-time rendering algorithms?

Fixed Function Pipelines (DX7)

- Writing new rendering algorithms means
  - Tricks with multitexture, stencil buffer, depth buffer, blending …
- Examples
  - Stencil shadow volumes
  - Hidden line removal
  - …

Programmable Shaders (DX8-10)

- Writing new rendering algorithms means
  - Tricks with stencil buffer, depth buffer, blending …
  - Plus: Writing shaders

- Examples
  - User-defined materials
  - User-defined lights
  - User-defined data structures (built in texture memory)

Software Graphics: Part I (DX11)

- Writing new rendering algorithms means
  - Tricks with stencil buffer, depth buffer, blending …
  - Plus: Writing shaders
  - Plus: Writing data- and task-parallel algorithms
    - Analyze results of rendering pipeline
    - Synthesize data structures

- Examples
  - Dynamic summed area table
  - Dynamic quadtree adaptive shadow map
  - Dynamic histogram-analysis shadow map
  - Dynamic ambient occlusion
  - Order-independent transparency
  - …

Software Graphics: Part II (DX11+)

- Writing new rendering algorithms means
  - Tricks with stencil buffer, depth buffer, blending …
  - Plus: Writing shaders
  - Plus: Writing data- and task-parallel algorithms
    - Analyze results of rendering pipeline
    - Synthesize data structures
  - Plus: Creating new and extended rendering pipelines

- Examples
  - Stochastic rasterization rendering
  - Ray tracing pipelines
  - …

What Next? Full Software Graphics?

- Intel's Larrabee
  - Advertised to come full-circle to "full software" real-time graphics
  - Unfortunately, the product was cancelled

- Yet
  - "Mixing SW graphics with the HW pipeline" is becoming mainstream
  - GPUs are becoming increasingly programmable
  - CPUs are becoming more parallel/powerful
  - CPUs and GPUs are getting "closer" with integrated CPU-GPU chips

- So…
  - Will fixed-function graphics hardware and fixed pipelines go away?

The Wheel of Reincarnation

Gradually the processor became more complex.... Finally the display processor came to resemble a full-fledged computer with some special graphics features. And then a strange thing happened. We felt compelled to add to the processor a second, subsidiary processor, which, itself, began to grow in complexity. It was then that we discovered a disturbing truth. Designing a display processor can become a never-ending cyclical process. In fact, we found the process so frustrating that we have come to call it the "wheel of reincarnation.”

- Myer and Sutherland “On The Design of Display Processors”, Communications of the ACM, **1968**

Will There Be Another Turn of The Wheel of Reincarnation?

- Is "the rise of SW graphics" a temporary (5-10) year window as we go around the wheel of reincarnation or has the wheel stopped turning?
- If it has stopped turning, why?
- If it hasn't stopped turning, what will be the next fixed-function?
  - *The next turn of the wheel will not look like today's graphics hardware*
  - *It is a great time to be a graphics researcher because the killer-app SW rendering pipelines/capabilities created now may define future fixed-function hardware*

(A Few) Open Real-Time Rendering Problems

- More realistic "eye rays"
  - Anti-aliasing
  - Motion blur and depth-of-field
  - Transparency
  - Curved and highly-detailed geometry
  - …

- More realistic illumination
  - Shadows (hard, soft, volumetric)
  - Global illumination
  - Reflections

From Johan Andersson's SIGGRAPH 2010 talk

(A Few) Open Programming Model Problems

- Consistent programming models for CPU and GPU
- Consistent programming model for pipeline, data, and task parallelism (“braided parallelism”)
- More flexible rendering pipelines
- (Many more will come up throughout the course as you use the state-of-the-art)

Conclusions

- SW+ HW graphics is here today (beginning “for real” in DX11)
  - Graphics programming is no longer simply a single pre-defined pipeline
  - Research is ablaze with SW real-time rendering research on GPUs and CPUs
- Future real-time rendering programming will likely consist of
  - A pre-defined (Direct3D/OpenGL) rendering pipeline
  - User-defined software pipelines
  - User-defined data- and task-parallel code tightly coupled to graphics pipelines
- Is the wheel of reincarnation still turning?

Questions?

- Email Aaron: alefohn@cs.washington.edu
- Email Mike: mhouston@graphics.stanford.edu

- Next lecture: “A trip down the 2003 rasterization pipeline”