Real-Time Realistic Rendering

Michael Doggett
Docent
Department of Computer Science
Lund university

30-5-2011
Visually realistic goal “… force[d] us to completely rethink the entire rendering process.”

Cook et al., 1987
The Reyes Image Rendering Architecture

Image courtesy and copyright Pixar
Slide courtesy Jonathan Ragan-Kelley
Outline

• GPU architecture
  • 2001-2009 ATI/AMD - Boston, U.S.A.
  • XBOX360, Radeon 2xxx-6xxx
• Decoupled Sampling
• Analytical Motion Blur
Graphics Group

- Tomas Akenine-Möller
- Michael Doggett
- Lennart Ohlsson
- Magnus Andersson
- Rasmus Barringer
- Per Ganestam
- Carl Johan Gribel
- Björn Johnsson
- Jim Rasmusson
- Philip Buchanan
128 fragments in parallel

16 cores = 128 ALUs, 16 simultaneous instruction streams

Slide courtesy Kayvon Fatahalian
128 \[ \text{vertices/fragments} \]

primitives

OpenCL work items

CUDA threads

] in parallel

vertices

primitives

fragments

Slide courtesy Kayvon Fatahalian
GPU design parameters

• Competition
  • Currently 2 strong competitors
    • AMD (ATI) and nVidia
  • Performance/Dollar

• Moore’s law
  • Number of transistors on a chip doubles every two years
GPU design parameters

- RTL design
  - nVidia ALUs from DX10 on are full custom
- Architecture changes hidden by API
- Fixed function uses programmable hardware
- Many custom units
- Backwards compatible
Before programmable GPUs

- ATI
  - Founded 1985 started
- nVidia
  - Founded 1993
- DirectX 6

Diagram:
- Image
  - FrameBuffer
  - Z & Alpha
  - Texturing
  - Textures
  - Rasterization
Hardware Transform, Clipping and Lighting

- Transformation requires 32-bit float 4x4 matrix multiplication
- Texturing for 4 component (RGBA) 8-bit pixels
- Low precision math
- DirectX 7
  - NV10 ’99 (Nvidia GeForce 256)
  - R100 ’00 (ATI Radeon 7500)
Programmable GPUs 1st Generation

- Shaders run programs that do what fixed function hardware did
- Makes GPUs simpler than CPUs
Programmable GPUs 1st Generation

- DirectX 8
- Multiple versions of Pixel shaders, 1.1, 1.3, 1.4
- 13-22 instructions
- assembler language
Programmable GPUs
1st Generation

- NV20 '01
  - Nvidia GeForce 3
    - [Lindholm01]
- R200 '01
  - ATI Radeon 8500
  - 2-wide Vertex shader
  - 4-wide SIMD Pixel shader
  - Fixed point ~16bits
Programmable GPUs 2nd Generation

- R300 ’02
- ATI Radeon 9700
- 4-wide Vertex shader
- 8-wide SIMD Pixel shader
- 24bit floating point math
Programmable GPUs 2nd Generation

- NV30 '03
- Nvidia GeForce FX 5800
  - 16 and 32 bit float
- Cg (C for graphics)
- NV40 '04 GeForce 6800 [Montrym05]
- DirectX 9
- High Level Shading Language (HLSL)
XBox360 GPU architecture - ’05

- Transform
- Texturing
- Vertex Shaders
- Pixel Shaders
- Rasterizer
- Index Stream Generator
- Tessellator
- Unified Shader
- Texture/Vertex Fetch
- Output Buffer
- Memory Export
- Vertex Pipeline
- Pixel Pipeline
- Display Pixels

UNIFIED MEMORY

DAUGHTER DIE
ATI Radeon 4870(R770) Die

- 2008
- 260mm\(^2\)
- 956 MTransistors
ATI Radeon 4870(R770) Die

- 260mm²
- 956 MTransistors
- Red
  - 10 SIMDs
- Orange
  - 64 z/stencil
  - 40 texture
Multi-Graphics core
NVIDIA Fermi

- GeForce GTX 480 (GF100)
- Released March, 2010
- Shader Load/Store L1/L2 cache
- 4x Triangle rate
- 4 rasterization engines
- 529mm$^2$, core i7 263mm$^2$
NVIDIA GeForce GTX 480 “SM”

- 15 Streaming Multiprocessors
- 20 Execution contexts (128 KB)
- “Shared” memory (16+48 KB)

ATI Radeon HD 5870 “SIMD-engine”

- 20 SIMD-engines
- 20 Execution contexts (256 KB)
- “Shared” memory (32 KB)
Intel’s Knights Corner/Ferry (Larrabee)

- 32 Pentium Cores
- 16 wide SIMD with 32 bit floating point MAD
- Programmable and debuggable

Images courtesy [Seiler08]
Analytical Motion Blur Rasterization with Compression

Carl Johan Gribel, Michael Doggett, Tomas Akenine-Möller

High-Performance Graphics 2010
Motion Blur Motivation

- Human visual system designed to detect motion
- This fails when presented too few images, or too much motion per image
  - motion gets jumpy
- Motion blur aids the motion detection of the visual system
Analytical Motion Blur Rasterization

- Compute Edge Equations and exact exposure intervals
  - analytic inside-test
  - visibility management

\[ e_1(t) = (p_2(t) \times p_0(t)) \cdot (x_0, y_0, 1) \]
\[ = (((1 - t)q_2 + tr_2) \times ((1 - t)q_0 + tr_0)) \cdot (x_0, y_0, 1) \]
\[ = (ft^2 + gt + h) \cdot (x_0, y_0, 1) \]

- Compute the time integral
- Store and compress the intervals
Moving triangle edge functions
Results

Ground Truth

Stochastic 12 spp

Analytical, compression 8
Decoupled Sampling for Real-Time Graphics Pipelines

Jonathan Ragan-Kelley, Jiawen Chen, Jaakko Lehtinen, Michael Doggett, Fredo Durand (collab. with MIT)

ACM TOG 2011, to be presented at SIGGRAPH 2011
Decoupled Shading

Rendering:
- Visibility - compute what is visible
- Shading - compute color for each pixel

Complex visibility
- many stochastic point samples in 5D (space, time, lens aperture)

Complex shading
- expensive evaluation can be prefiltered

Modern GPUs use multisample AA for decoupling
Our technique: Post-visibility Decoupled Sampling

1. Separate *visibility* from *shading* samples.

2. Define an explicit *mapping* from visibility to shading space.

3. Use a *cache* to manage irregular shading-visibility relationships, without precomputation.
Decoupled Sampling with motion blur

foreach primitive:
    foreach vis sample:
        skip if not visible
        map to shading sample
        if not in cache:
            shade and cache
        else:
            use cached value

Slide courtesy Jonathan Ragan-Kelley
Decoupled Sampling with motion blur

$t = 0.05$

visibility samples (screen space)

shading grid

foreach primitive:
  foreach vis sample:
    skip if not visible
    map to shading sample
    if not in cache:
      shade and cache
    else:
      use cached value

Slide courtesy Jonathan Ragan-Kelley
Decoupled Sampling with motion blur

visibility samples (screen space)

foreach primitive:
  foreach vis sample:
    skip if not visible
    map to shading sample
    if not in cache:
      shade and cache
    else:
      use cached value

Slide courtesy Jonathan Ragan-Kelley
Decoupled Sampling with motion blur

\[ t = \ldots \]

visibility samples (screen space)

foreach primitive:
  foreach vis sample:
    skip if not visible
    map to shading sample
    if not in cache:
      shade and cache
    else:
      use cached value

Slide courtesy Jonathan Ragan-Kelley
Decoupled Sampling with motion blur

foreach primitive:
  foreach vis sample:
    skip if not visible
    map to shading sample
    if not in cache:
      shade and cache
    else:
      use cached value

visibility samples (screen space)

Slide courtesy Jonathan Ragan-Kelley
Decoupled Sampling with motion blur

foreach primitive:
  foreach vis sample:
    skip if not visible
    map to shading sample
  if not in cache:
    shade and cache
  else:
    use cached value
Decoupled Sampling

- Transformed primitives
- Covered subpixels
- Visible subpixels
- Colored subpixels

Slide courtesy Jonathan Ragan-Kelley
Decoupled Sampling

- Transformed primitives
- Covered subpixels
- Visible subpixels
- Colored subpixels

Xform → Rast → Depth-Stencil → Map → FB

Shade

Cache

Shading requests → Shaded results

Slide courtesy Jonathan Ragan-Kelley
Results
Blur vs. shading rate: defocus

Half-Life 2, Episode 2
1280x720, 27 samples/pixel

4.5-43x less shading than ideal supersampling

Slide courtesy Jonathan Ragan-Kelley
Blur vs. shading rate: motion

Team Fortress 2
1280x720, 27 samples/pixel

3-40x less shading than ideal supersampling

64 samples/pixel
27 samples/pixel
8 samples/pixel

Slide courtesy Jonathan Ragan-Kelley
Blur vs. Shading Rate

Decoupled Sampling
64 visibility samples

Slide courtesy Jonathan Ragan-Kelley
Depth of field results

Valve's Half-Life 2 : Episode 2
Graphics Hardware

- Hardware on which 3D graphics is created in real-time
- Traditionally custom hardware
  - Increasing programmability
  - More complex rendering algorithms
- Programmable hardware available on a wide range
  - Mobile - PowerVR, Desktop - AMD Radeon/Nvidia GeForce
  - Heterogeneous processors - CPU with integrated graphics
    - Sandy Bridge
- Create new algorithms that make use of architectural features
Device properties

- Number and type of cores
- Task parallel
  - OOO/In-order
- Data parallel
  - SIMD/SIMT, width, VLIW/scalar
- Caches
  - Size, bandwidths, hierarchy, coherency
- Adaptable to new varieties of architectures
Challenges

• Wide range of changing hardware
• Efficient programming and utilization
  • CUDA, OpenCL, DirectCompute
  • Fixed custom scheduling (currently)
• Quest for ever increasing realism
  • Analytical visibility
• Decoupled sampling
Summary

- GPUs have evolved into massively parallel processors
- Analytical Motion Blur improves quality leading to more realistic images
- Decoupled shading enables real-time realistic rendering
- Need new approaches to adapt to wide variation and complex scheduling
Thanks for listening and Questions