Real-Time Hard Shadows with Parallel Algorithms

Aaron Lefohn
(with slides from Andrew Lauritzen, Nico Galoppo, Matt Pharr)
Intel Graphics Research
Overview

• Brief history of real-time shadow rendering

• New shadow algorithms that are “beyond programmable shading” (i.e., that use general parallel algorithms)
  – Resolution Matched Shadow Maps
  – Irregular Z-Buffer

• Open problems in shadow rendering
Caveats

• This talk is NOT an exhaustive survey of real-time shadow techniques
  – There are hundreds of shadow techniques
  – This talk focuses on newer shadow methods that combine the rendering pipeline with parallel algorithms

• This talk focuses on a few recent techniques for hard shadows from opaque occluders
  – Translucent occluders a separate active area of research
  – Soft shadows an active (and harder) area of research
The Shadowing Problem
The Shadowing Problem
The Shadowing Problem
Late 1990s: Lightmaps

- Blend low-resolution precomputed shadow texture
- Motivated multi-texturing
Early 2000s: Stencil Shadows

- 2002: stencil shadows
  - Bandwidth intensive, difficult with GPU-side vertex processing

Image from Lloyd et al, "CC Shadow Volumes" EGSR 2004
(The Downside: Overdraw)
PCF Shadow Maps

• 2002-2004: PCF shadow maps
  – Render depth image from light source
  – Project camera space points into depth map to test for shadowing
Fixed-Function Shadows, RIP

• 2005-
  – Variance shadow maps, cascaded shadow maps, exponential shadow maps, ...

• 2009 and beyond
  – Irregular z-buffer, sample distribution shadow maps, adaptive volumetric shadow maps, ...
Take-Aways

• Shadows are hard

• Researchers/developers will go to great lengths to create better shadows

• General parallel compute closely coupled with the rendering pipeline opens up many new possibilities for shadows
Shadow Mapping

• Render depth image from light position
• Shadow lookup
  – Transform eye samples to shadow map
  – If shadow map value closer to light, pixel in shadow

Williams
SIGGRAPH 1978
Projective Aliasing

• Occluder normal nearly orthogonal to light rays
Perspective Aliasing

• Mismatch between sampling distribution of eye-space samples and shadow samples
Shadow Map Techniques

• Hundreds of shadow map papers address perspective, projective, and depth representation aliasing

• The only shadow-map-based approaches that directly address both perspective and projective aliasing are
  – Irregular rasterization
  – Adaptive grid-based methods

• (Techniques that address only perspective aliasing also reduce projective aliasing, but depends on surface orientation)
Resolution Matched Shadow Maps

UC Davis 2006
Neoptica 2007
Resolution Matched Shadow Maps

• Solve shadow map aliasing problems by generating a quad-tree of small shadow maps based on the shadow requests made by the pixels in current camera view

• Pros
  – Quality very close to ray tracing but using rasterization infrastructure
  – Quadtree shadow map supports filtering (cheaply fake more shadow rays)

• Cons
  – Performance is view-dependent (vary performance instead of quality)

• Caveat
  – This work was done in 2006 using DX9-class GPUs and in 2007 on PlayStation3. A DX11 implementation would look more like the PS3 version but be all on the GPU.
Resolution Matched Shadow Maps

- Interactive shadow map technique for high-quality hard shadows for dynamic scenes
Resolution Matched Shadow Maps

• Nearly alias-free interactive hard shadows
  – Generate $32,768^2$ quadtree shadow map every frame

• Key insight
  – When 1:1 resolution matching between screen space and shadow-map space, contiguous surfaces in screen space are contiguous in shadow-map space
  – Rasterizer can efficiently generate correctly-sampled shadow “rays”
Quadtree Shadow Data Structure

• Shadow map coordinates (virtual domain)

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

- Paged 2D texture memory

Virtual Domain → Physical Domain
Quadtree Address Translator

- Mipmap page table

Virtual Domain

Physical Domain
Building the Mipmapped Page Table

Virtual Domain

Physical Domain
Overview: Building Quadtree Shadow Map

- Discover which regions of quadtree shadow map are required by current frame (GPU)
  - Quadtree nodes called “virtual page numbers” (vpn)
- Transfer page allocation requests (GPU → CPU)
- Data-parallel page allocation (CPU + GPU)
- Generate shadow data for new pages (GPU)
Building Mipmapped Page Table: 1

Virtual page number (node id) for every pixel
Building Mipmapped Page Table: 2

Vpn for every pixel

One vpn per contiguous region

• Invalidate identical contiguous vpns
• Data-parallel connected component analysis
Building Mipmapped Page Table: 3

- Remove invalid vpns (node ids)
- Data-parallel compaction (scan, gather)

One vpn per contiguous region

Valid vpns only
Building Mipmapped Page Table: 4

- Sort vpns with data-parallel sort algorithm
Building Mipmapped Page Table: 5

- Invalidate duplicate vpns
- Data-parallel connected component analysis

Sorted vpns

One vpn per contiguous region
Building Mipmapped Page Table: 6

- Remove invalid vpns
- Data-parallel compaction (scan, gather)
- ...then...transfer memory page request to CPU

One vpn per contiguous region

Only unique vpns
For each mip level of quadtree page table
- Allocate new page by mapping page from physical memory into virtual address space of quadtree
- Data-parallel write of new page table entries (PTEs)
Building Mipmapped Page Table: 7b

Quadtree node (virtual page) allocation request

Level 1 of mipmap page table
Building Mipmapped Page Table: 7c

Quadtree node (virtual page) allocation request

Level 2 of mipmap page table
Building Mipmapped Page Table: 7...
Generating Quadtree Shadow Data

Virtual Domain

Physical Domain
Generating Quadtree Shadow Data: 1

Virtual shadow domain (one of N levels)

Three new pages to render
Generating Quadtree Shadow Data: 2

Cluster pages into superpage

Render superpage
Generating Quadtree Shadow Data: 3

Rendered superpage

Physical Memory (Quadtree node storage)
Quadtree Shadow Map

Virtual Domain

Physical Domain
Apply Shadows to Scene

• Render from eye
• Compute shadow map coordinates \((s, t, \text{lod})\)
• Shadow map lookup in quadtree
  – Page table converts \((s, t, \text{lod})\) to physical addresses
  – 2x2 hardware percentage closer filtering works fine
  – Trilinear PCF between LODs to filter discrete LODs
• “Just like standard shadow map”
RMSM Results: Furball
RMSM Results: Furball
RMSM Results: Furball
RMSM Results: Tree Scene

32,768²  16,384²  8,192²
RMSM Results: Town Scene

65K x 65K RMSM shadow from moving spotlight above bridge
RMSM Results: Town Scene

65K x 65K RMSM shadow from moving spotlight above bridge
RMSM Results: Town Scene

RMSM shadows from two levels of chain-link fence created as alpha-tested geometry
RMSM Movie
• Why a new algorithm?
  – Faster, use less memory, less per-frame variability
  – Leverage emerging many-core architectures (PS3, Larrabee, etc.)
    – Efficient execution of small number of threads per core
    – Efficient core-to-core communication

• What changed?
  – Build page table with task-parallel algorithm
    – Remove variable-time and costly data-parallel algorithm
    – Performance now only depends on number of superpages
  – Stream shadow data using single $1024^2$ scratch buffer
    – Greatly reduced memory requirement (now fixed memory instead of variable)
  – Use $1024^2$ page size
    – Superpage is the page size
2.0: Algorithm Highlights

• Memory requirements greatly reduced because quadtree data structure is generated-and-consumed one page at a time

• Optimize by computing shadow-map-space and screen-space bounding boxes for each shadow page

• Performance variability only based on number of rendered superpages
  – Capturing on-chip r/w locality gives large bandwidth savings
  – Replace many-pass data-parallel memory allocation algorithm with single-pass task-parallel version
2.0: Task-Parallel Page Table Building

• Why?
  – Remove variable-time and costly data-parallel algorithm
    – ~20x bandwidth reduction
  – Performance now only depends on number of superpages
    – Single pass over framebuffer instead of many

• Hardware requirements
  – Use 8+ CPU-like cores plus GPU
  – Requires fast single-task performance and large per-task storage (at least 128 Kb per task)
  – Requires fast CPU-GPU interconnect or device with CPU + GPU capabilities
2.0: Step 1 is Identical to 1.0

- Render from eye
- Compute shadow map coordinates and LOD
- Compute quadtree vpn for each pixel

Virtual page number (node id) for every pixel
2.0: Steps 2-7 Combined into Single Pass

• Spawn N tasks
  – N is minimum required for all cores to run at peak performance

• Each task
  – Streams over 1/Nth of page request framebuffer
  – Builds mipmap page table in on-chip memory
  – For each requested page
    – Find (s, t) bounding box for each requested page
    – Find (x, y) screen-space bounding box
2.0: Steps 2-7 Combined into Single Pass

• Parallel merge of all tasks’ page tables

• Single (final) task writes out
  – List of shadow pages to render
  – (s, t) and (x, y) bounding boxes for each pages
2.0: Generating & Using Shadow Data

- Combine generation and application of shadow map data

- For each requested page
  - Render shadow data cropped to (s, t) bounding box
    - Always render into same $1024^2$ scratch buffer
  - Apply shadow data to scene
    - Render screen-space bounding box
    - Perform shadow lookups into shadow page buffer
RMSM Summary

• Assume 2.0 algorithm
  – Scene analysis and building page table is fixed-cost based on screen resolution and will become nearly free on near-term architectures

• Dominant cost
  – Generate 2-7x more shadow texels than screen-space pixels
  – Multiple geometry passes

• Still have per-frame performance variability because of varying number of geometry passes
  – Can use heuristics to limit # of passes, but introduces errors
Irregular Z-Buffer Shadows

UT Austin and Intel
(Slides by Nico Galoppo, Intel)
Irregular Z-Buffer Shadows

• RMSMs handle irregular light-space sample requirements by building a quadtree and taking advantage of coherence to use rasterizer (reasonably efficiently)
  – But there are cases when a rasterizer is the wrong tool (e.g., 10% utilization of pixels in quadtree node/page)

• Irregular z-buffer shadows give up on regular rasterization and rasterize exactly the correct samples in light space
  – Custom ray tracer only for shadow rays with 2D acceleration structure
  – Implemented as mix of standard rasterization pipeline and parallel compute

  – Pros
    – Quality matches ray tracing using (some of the) rasterization infrastructure

  – Cons
    – Performance is unproven: “Turn quality problem into a load balancing problem”
    – Shadow data cannot be filtered in light space (no faking more shadow rays)
Irregular Shadows

Light-view Classical Z-buffer

Light-view Irregular Z-buffer

Classical Shadow

Irregular Shadow

Figures by Greg Johnson
Problem: Light View Aliasing

- **In light view**, due to misalignment between sample locations
  - Traditional shadow mapping algorithms store samples at grid locations

*Figure by Greg Johnson*
Solution: Irregular shadow mapping

- Irregular shadow mapping [Johnson04] is free from light view aliasing
  - Store shadow samples in 2D irregular data structure (irregular z-buffer)
  - Ray-traced quality shadows

Figure by Greg Johnson
Irregular Z-Buffer Shadows (IZB)

- Summary
  - Ray-trace quality hard shadows using rasterization-based algorithm for opaque and alpha-tested hard shadows

- Algorithm stages
  - Capture shadow receiver sample positions
    - Render shadow receiver depths from eye
    - Project shadow receiver sample positions into light space (parallel data structure build)
  - Intersect occluders with receivers
    - Render occluders from light (with conservative rasterization)
    - Intersect occluders with receiver samples (traverse grid-of-lists)
  - Store eye-space “visibility” result buffer
Irregular Z-Buffer Shadows
Problem: Even Ray Tracing Aliases

Single Shadow Sample per Pixel
Single Shadow Sample per Pixel
Brute Force Super-Sampling
Adaptive Multi-Sampling

• Brute force super-sampling: impractical compute & storage costs

• Multiple shadow samples only required on shadow silhouette (in eye view)
  – Improve computational & storage cost

• Determining eye view shadow silhouette is inverse problem
  – Conservative estimate is sufficient
Conservative Shadow Silhouettes Estimate

• Screen – space stencil

  "Is there any chance that this eye view pixel is partially occluded by geometry in the projected light space texel?"

• Conservative answer
  – Use probability / occluder depth-from-light distribution $\chi$
  – Variance shadow map (VSM) [Lauritzen06]
  – Compute fraction of occluder distribution that is farther from light than receiver $t$ (Chebyshev)
Multi-Sampled Irregular Shadow Mapping

- Multi-sampled, not brute force super-sampled
- Simple extension of irregular shadow mapping for hard shadows [Johnson04]
- Insert multiple shadow samples in irregular z-buffer (IZB)
  - For potentially aliased pixels only, from conservative shadow silhouette stencil
  - Add eye-view shadow samples to light-view IZB
- Average eye-view occlusion values in final gather
Single Sample
Conservative Shadow Silhouettes
Single Sample
4 Samples / Potentially aliased pixel (rotated grid positions)
Conservative Shadow Silhouettes
4 Samples / Potentially aliased pixel (rotated grid positions)
IZB Performance Summary

– Can be implemented w/DX11 geometry+compute shaders, but...
  – Lack of reconfigurable pipeline results in lower performance
    – GS implementation of conservative rasterization
    – Inflexible work balancing due to fixed pipeline

– Looks more promising with a customizable pipeline
  – Conservative rasterization is free (modified rasterizer)
  – Custom pipeline enables flexible work (re-)balancing
    – Custom “fragment” backend to efficiently traverse irregular data structure
IZB Summary

• Advantages
  • Guaranteed pixel-perfect shadow quality
    • Fully automatic (no artist input / partitioning)

• Disadvantages
  • No filterable shadow representation (single shadow sample/pixel). So to compete with filterable techniques, must actually evaluate multiple shadow rays per pixel
  • Load balancing hard (convert quality challenge to perf. challenge)
Conclusions / Open Problems

• Real-time applications require “authorable performance and memory usage”
  – Very challenging design requirement for adaptive algorithms because must target “worst case” rather than “hope for best case”

• Flexible parallel programming of modern and near-term GPU architectures is opening up new possibilities for shadow rendering
  – Analyze required shadow samples before generating them
  – Build and use irregular data structures for shadow samples
References

• Quadtree shadow maps

• Irregular z-buffer shadows
  – Johnson et al, “The irregular Z-buffer: Hardware acceleration for irregular data structures” ACM Transactions on Graphisc, 2005