Height field ambient occlusion using CUDA

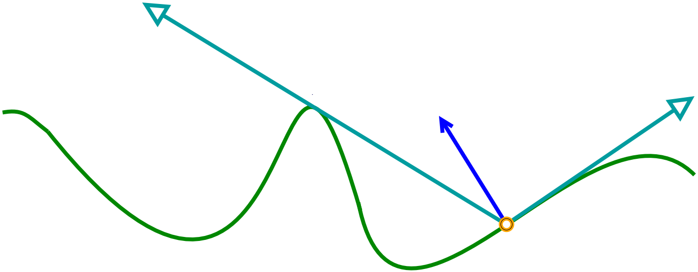
3.6.2009

Outline

1. Introduction
2. Method
   - Idea
   - Implementation
3. Data management
   - Naive solutions
   - Performance improvements
   - Preblend kernel
4. Occlusion computation
   - Theory
   - Kernel
5. Results

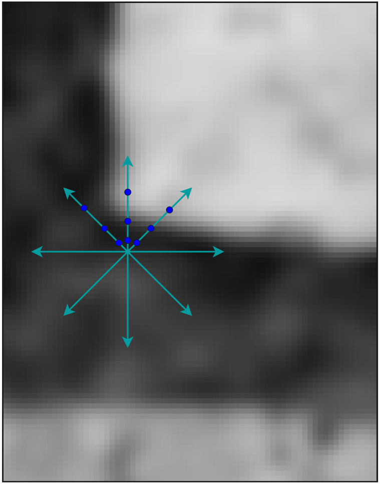
Height fields

Self occlusion



Current methods

- Marching several directions from each fragment
- Sampling several times along a direction
- Optimizations based on this approach



Outline

1 Introduction

2 Method
- Idea
- Implementation

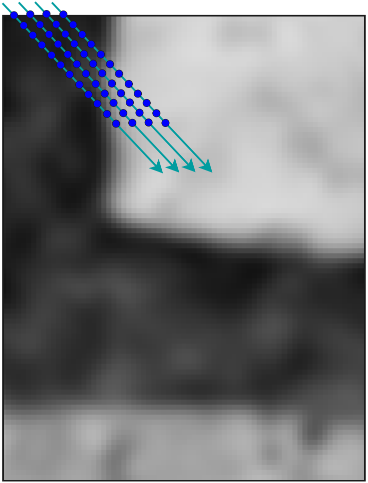
3 Data management
- Naive solutions
- Performance improvements
- Preblend kernel

4 Occlusion computation
- Theory
- Kernel

5 Results

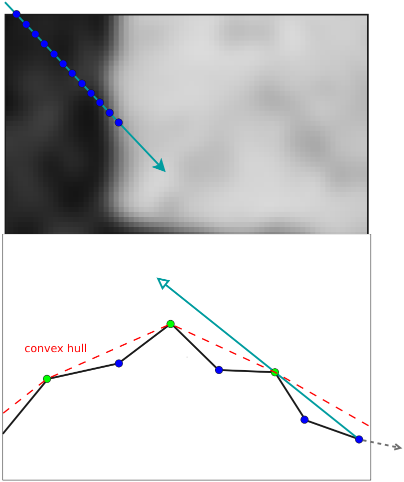
Sweeps

- We process several (e.g. 64) directions (*sweeps*)
- For a height field of n^2 (e.g. 1024x1024) a sweep consists of $n \cdots \sqrt{2}n$ lines
- One direction = infinitely high but thin light source



Spatial coherence

- Each line steps one texel at a time
- Keeps a record of previous occluders
- Line-wise coherence
- Convex hull subset of occluders is enough
- 3% of occluders required in practice
- Output: occlusion vector/value on each step
- Results for each direction blended together



Implications

- Previously bandwidth limited
- New method samples significantly less
- Bound to discrete directions, but ideally takes each pixel on them into account
- Does not map to shaders well $\Rightarrow$ GPGPU

Outline

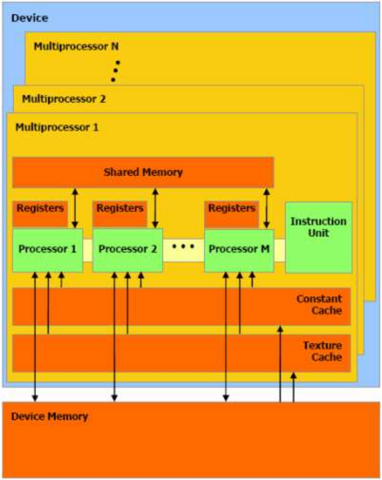
1. Introduction
2. Method
   - Idea
   - Implementation
3. Data management
   - Naive solutions
   - Performance improvements
   - Preblend kernel
4. Occlusion computation
   - Theory
   - Kernel
5. Results

Environment

- OpenGL 3.0 (Aug 2008)
- CUDA 2.2 (May 2009)
- Linux (primary), Windows
- Software (CPU) and hardware (GPU) implementations

Tesla architecture

- Global (device) memory on-board, various access modes
- Shared memory on-chip, for each MP
- 64kB, 16 banks, interleaved 32b
- 30 MPs on GTX280, 8 ALUs on a MP



Outline

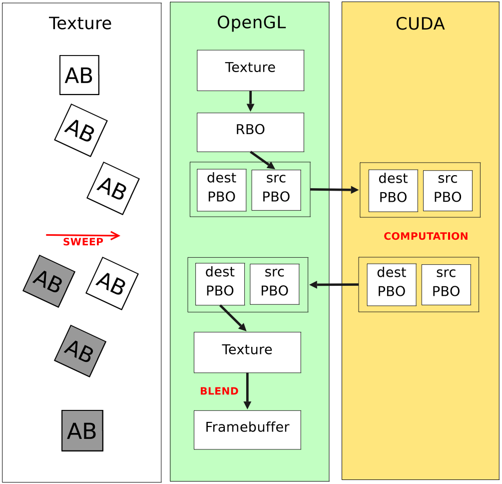
1 Introduction
2 Method
- Idea
- Implementation

3 Data management
- Naive solutions
- Performance improvements
- Preblend kernel

4 Occlusion computation
- Theory
- Kernel

5 Results

OpenGL interoperability



OpenGL interoperability

- Too much data copying
- Not enough threads
- We can transfer more sweeps at once to remedy the latter

Blending in CUDA

- Write into multiple dests as before
- Blend using CUDA
- $\Rightarrow$ still memcpies, sampling slower in CUDA than in OpenGL

Outline

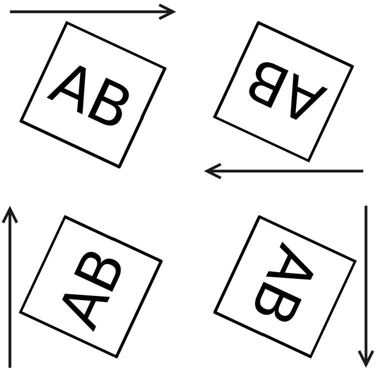
1. Introduction
2. Method
   - Idea
   - Implementation
3. Data management
   - Naive solutions
   - Performance improvements
   - Preblend kernel
4. Occlusion computation
   - Theory
   - Kernel
5. Results

Sample src PBO with CUDA

- Reduces data passed between OpenGL and CUDA
- Need to copy src PBO to CudaArray
- CUDA provides 2D bilinear filtering (read-only)

Preblending in CUDA

- Cannot write directly, atomic adds are slow
- 180° rotation is trivial
- 90°/270° needs shared mem tricks for mem coalescing



Packing texels

- CUDA memory coalescing only works for thread-consecutive 32b elements
- But we have only single luminance value for a texel
- This calls for bit operations
- Heaviest on preblend kernels, which luckily have low arithmetic density

00 01 02 03	10 11 12 13	
04 05 06 07	14 15 16 17	
08 09	18 19	

+

00 01 02 03	04 05 06 07	08 09
10 11 12 13	14 15 16 17	18 19
20 21 22 23	24 25 26 27	28 29

=

00 10 20 30	40 50 60 70	80
01 11 21 31	41 51 61 71	81
02 12 22 32	42 52 62 72	82
03 13 23 33	43 53 63 73	83

Outline

1. Introduction
2. Method
   - Idea
   - Implementation
3. Data management
   - Naive solutions
   - Performance improvements
   - Preblend kernel
4. Occlusion computation
   - Theory
   - Kernel
5. Results

Outline

1 Introduction

2 Method
- Idea
- Implementation

3 Data management
- Naive solutions
- Performance improvements
- Preblend kernel

4 Occlusion computation
- Theory
- Kernel

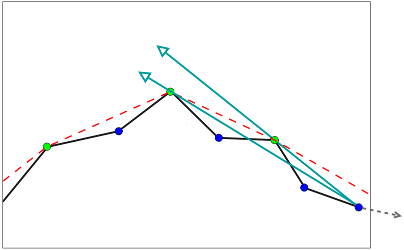
5 Results

Function of the kernel

- Processes one line, reads height values as input
- Writes aligned packed values
- Keeps a representation of the convex hull
- Outputs an occlusion vector
- Coalescing vs. length uniformity
- Thread block size and shared memory resources

Occluder processing

- Keep a vector of occluders in shared mem
- Search it backwards (linear/binary)
- Use heuristic comparison operator
- Insert new occluder
- Remove remaining (change length indicator)
- Return *last* − *current*



Outline

1. Introduction
2. Method
   - Idea
   - Implementation
3. Data management
   - Naive solutions
   - Performance improvements
   - Preblend kernel
4. Occlusion computation
   - Theory
   - Kernel
5. Results

Performance figures

- Profilers are immature
- But we know that we get 50-80 GB/s device memory rates
- And 200-400 Gops/s

Comparison

- M$ published previous state-of-the-art method in Eurographics 2008
- It scales badly, 1024x1024 @ 2.5fps, 32 directions
- Our method achieves 36-42fps, 64 directions
- Is texel-precise on sharp edges

Screen captures