CUDA and Multi-Core Gaming

Dan Amerson
Avi Bleiweiss

NVIDIA

Agenda

- Couple of examples of non-graphics uses for the GPU
- Dan Amerson of Emergent Game Technologies
  - Integrating CUDA into Gamebryo Floodgate
- Avi Bleiweiss of NVIDIA
  - Multi-agent Navigation on the GPU

Introduction

- Dan Amerson
- Technical Director, Gamebryo Core Runtime
- Emergent Game Technologies

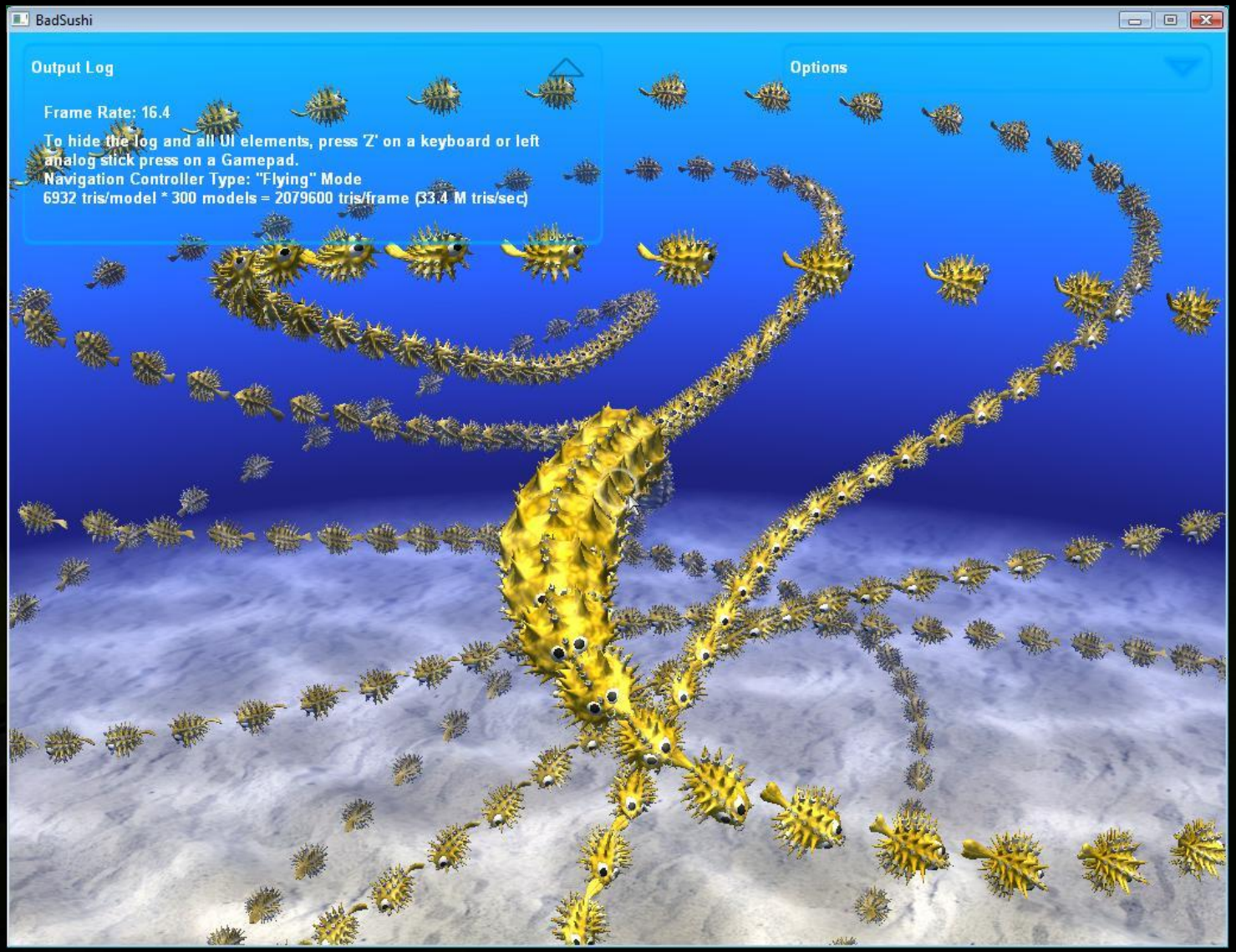
Revisiting CUDA

- At NVISION 08, I presented some investigations into CUDA.
- I posed this question.
  - Should I use CUDA in my game engine?
- The answer was:
  - Start prototyping.
  - Widespread use is not here yet.

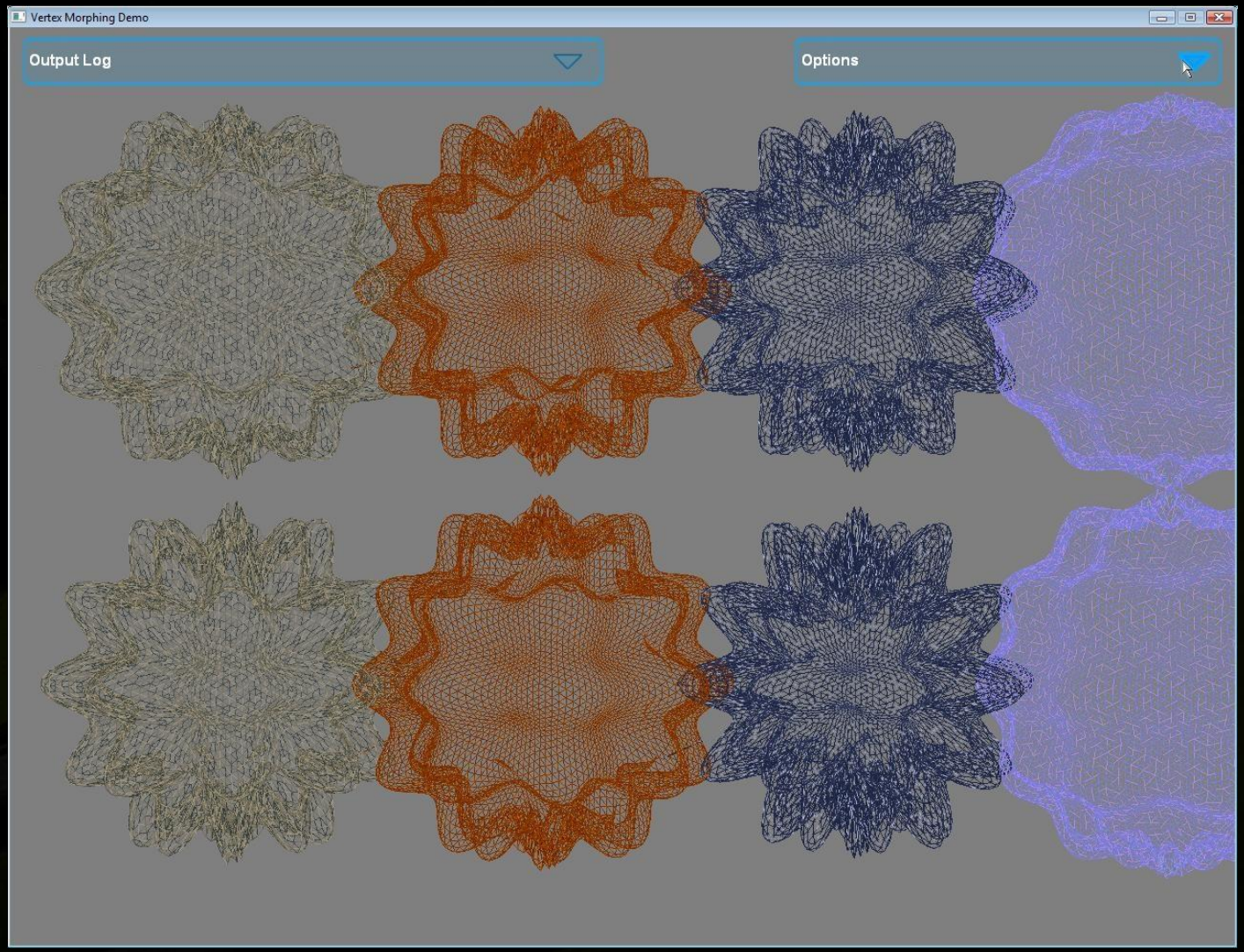
What Was The Experiment?

- Integrate Floodgate with CUDA
  - Floodgate is a stream processing engine within Gamebryo.
- Compare a number of approaches using CUDA to a CPU side approach on a quad core PC.

Floodgate in Action



The Actual Experiment



Testing Approaches

- CPU: Quad-core PC scaled from 115 fps serial processing to 180 fps with 3 worker threads.
- GPU: For GPU resident tasks, CUDA was about 2X faster.
  - Approach 1: 50 fps
  - Approach 2: 145 fps
  - Approach 3: 400 fps

GPU Test Approach 1

- Approach
  - Upload input blend shapes each frame.
  - Download results each frame.
  - Lock D3D VB and upload results again.
- Results
  - Performance scaled negatively to 50 fps.
- If inputs and outputs need to cross the bus per frame, CUDA is not necessarily a good fit.

GPU Test Approach 2

- Approach
  - Treat input blend shapes as static to reduce bus transfers.
  - Still performing two transfers per frame, read back from CUDA and D3D upload.
- Performance exceeds single threaded execution at 145 FPS.
- Opportunities exist for single-threaded code.

GPU Test Approach 3

- Approach
  - Use D3D interoperability methods.
  - No VB transfers across the bus.
- Fastest performance of any configuration at 400 FPS.
- Exceeds performance of a quad-core PC with 3 worker threads in test.

Analysis

- Time dominated by PCIe transfers in early approaches.
  - Average transfer was .261ms for a 240KB buffer.
  - We're seeing about .87 GB/s by this data much lower than peak PCIe rates.
- Possibly a result of using safer runtime APIs instead of driver APIs.
  - cudaMemcpy vs. cuMemcpyDtoHAsync et al.

Transfer Test - Stats

	Runtime	Mixed	Driver
Min	0.19952ms	0.10902ms	0.06646ms
Max	1.25872ms	3.00557ms	1.15594ms
Mean	0.26149ms	0.94669ms	0.10312ms
StDev	0.01738ms	0.03554ms	0.02232ms

That Was Then…

- At NVISION, the message was to invest at a prototype level.
  - API was finicky.
  - Public drivers didn’t have support for CUDA.
  - Lots of additional/future work was required.

Since Then…

- CUDA 2.1 was released.
  - Fixes some of the issues and improves ease-of-use. E.g, More robust D3D interoperability.
  - There's still some learning curve.
  - NVIDIA is very responsive to bug reports.
- Support is in public drivers.
  - Beginning with 177.81 drivers.
  - CUDA 2.1 requires 181.20 drivers.

And Now…

- D3D11 is much more imminent.
  - D3D11 Compute Shaders offer a model very similar to CUDA.
  - Getting started on CUDA prepares you for D3D11 and Win7.
  - There are other options in the short term, but CUDA has fewer limitations.
    - CS 4.x on D3D10.
    - D3D11 betas.

Ask the Question Again…

- Should I use CUDA in my game engine?
- Yes, the time is now.
  - CUDA is the best way to put this technology in a game today and prep for tomorrow.
  - D3D11 is around the corner.

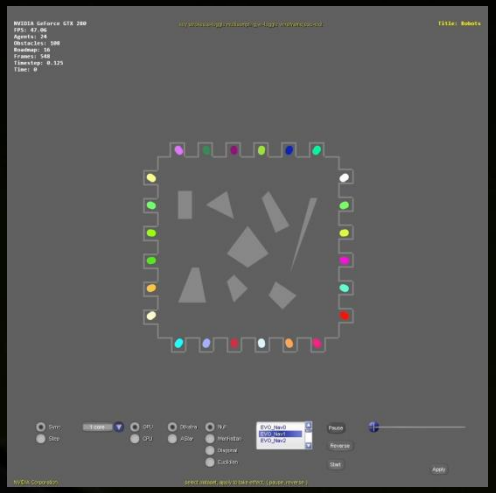
Thank you

- Thank you to:
  - Randy Fernando, NVIDIA developer relations, and Vincent Scheib for help with original NVISION talk.
  - Ignacio Castano and Cem Cebenoyan at NVIDIA for help with the GDC version.
- Come by Booth 5818 in North Hall
- amerson@emergent.net


Multi Agent Navigation
on the GPU
Avi Bleiweiss
NVIDIA

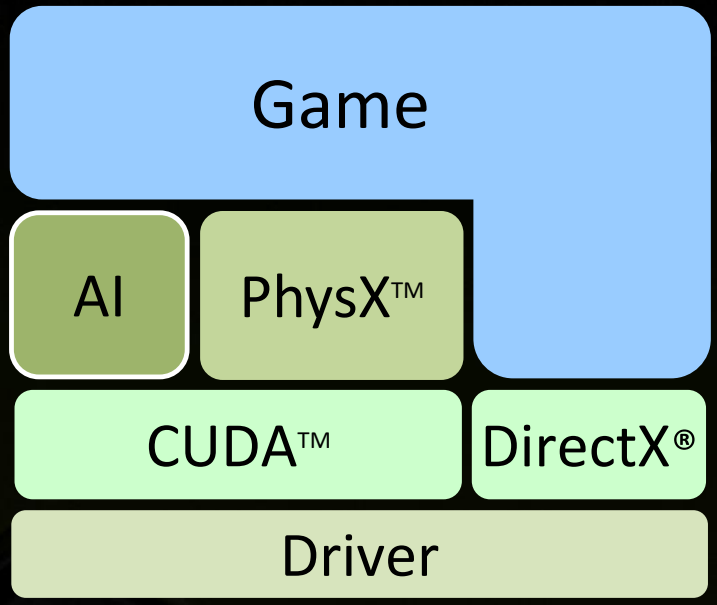
Reasoning

- Explicit
  - Script, storytelling
  - State machine, serial
- Implicit
  - Compute intensive
  - Fits SIMT architecture well
- Navigation planning
  - Collision avoidance



Motivation

- Computational intelligence
  - On CUDA platform
- Alternative pathfinding
  - Intuitive multi threading
  - Flat, nested parallel
- Scalable, real time
  - Dense environments



Problem

Planner

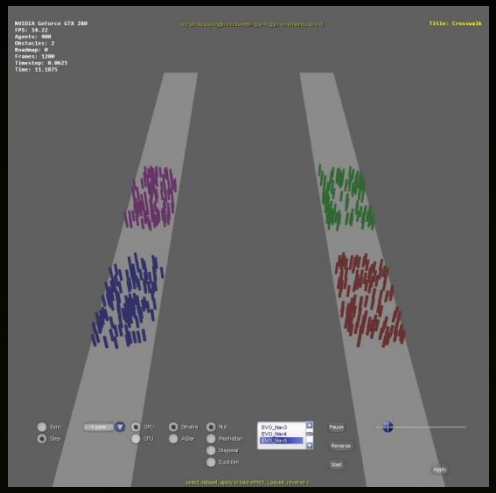
- Searches a global, optimal path
  - From start to goal
- Locally, avoids collisions with
  - Static, dynamic objects
- Exploits autonomous sensing

Simulator

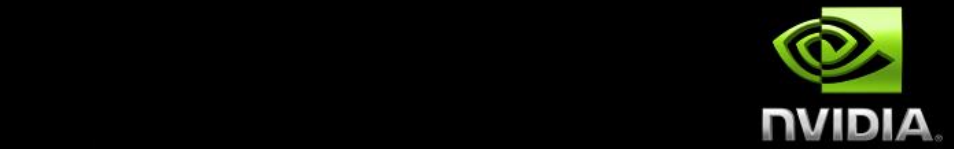
- Visually compelling motion
- Economical memory footprint
- A subset of compute units
- Linear scale with # characters

Solution

- Multi agent model
- Pre-computed roadmap
- Extended Velocity Obstacles
  - Global path integration
  - No explicit communication
- GPU specific optimization
  - Nearest neighbors search



Outline



- Algorithm
- Implementation
- Results
- Takeaways

Paper: Bleiweiss, A. 2009. Multi Agent Navigation on GPU

Algorithm

Visibility

- Two sets of edges
  - Visible roadmap node pairs
  - Goal to unblocked nodes
- A* search, shortest path
  - From goal to any node
- Line segment obstacles
  - Efficient sweep line method

A point is visible from another point -

If the connecting line doesn't intersect any static obstacles.

Velocity Obstacles

- Avoidance velocity set for
  - Dynamic agents among
  - Passively moving obstacles
- Prone to oscillations
- Reciprocal Velocity Obstacles
  - Identical, collision free mind
- Complement set
  - Admissible agent velocities

Velocity Obstacles:
[Fiorini and Shiller 1998]

$$VO_B^A(v_B) = \{ v_A \mid \lambda(p_A, v_A - v_B) \cap B \oplus -A \neq \emptyset \}$$

Reciprocal Velocity Obstacles:
[Van Den Berg et al. 2008]

$$RVO_B^A(v_B, v_A) = \{ v'_A \mid 2\,v'_A - v_A \in VO_B^A(v_B) \}$$

Simulation

- Simulator advances until
  - All agents reached goal
- Path realigned towards
  - Roadmap node or goal
- Agent, velocity parallel

```
1:  VO = velocity obstacle
2:  RVO = reciprocal velocity obstacle
3:  do
4:      hash                                  ┐
5:          construct hash table              │
6:      simulate                              │ per
7:          compute preferred velocity        │
8:          compute proximity scope           │
9:          foreach velocity sample do        │ frame
10:             foreach neighbor do           │
11:                 if OBSTACLE then VO       │
12:                 elseif AGENT then RVO     │
13:             resolve new velocity          │
14:     update                                │
15:         update position, velocity         │
16:         resolve at-goal                   ┘
17: while not all-at-goal
```

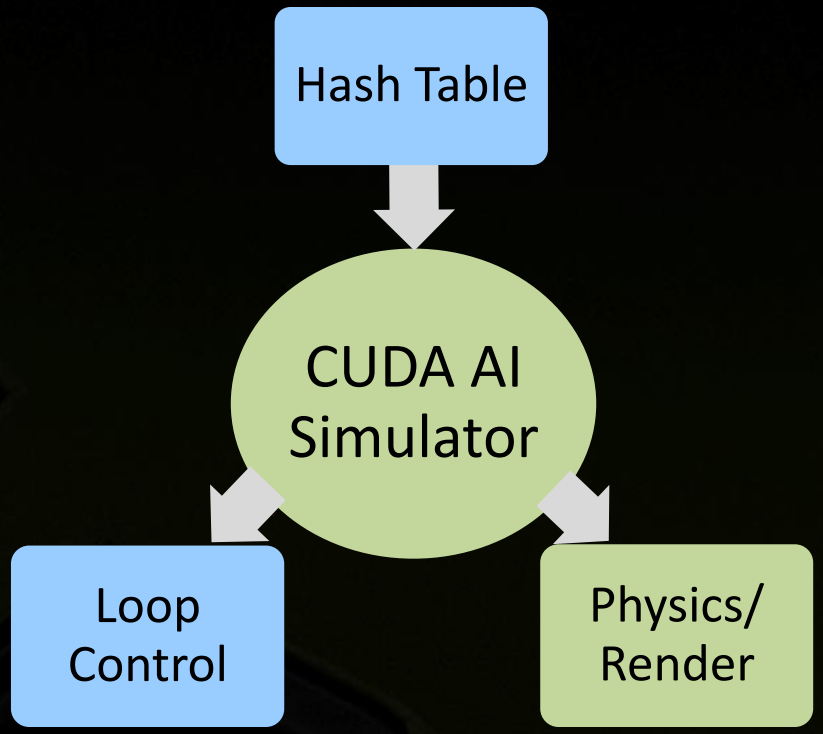
nested

flat

Implementation

Workflow

- CUDA kernel pair
  - *simulate* and *update*
- Deterministic resources
  - Allocated at initialization
- Per frame output
  - At-goal, path waypoints
- Split frame, multi GPU
  - Device-to-device copy

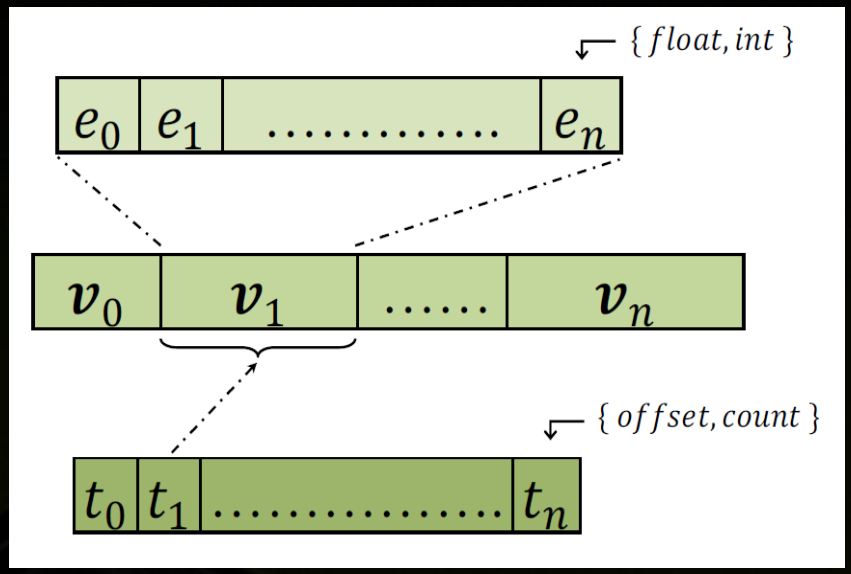


Challenges

- Hiding memory latency
- Divergent threads
- Hash construction cost
- Thread safe RNG

Data Layout

- Persistent resources
  - Reside in global memory
- Static, read-only data
  - Texture bound, linear
- Thread aligned data
  - Better coalescing
- Consistent access pattern
  - Improves bandwidth



Nearest Neighbors Search

- Naïve, exhaustive scheme
  - $O(n^2)$ total running time
- Spatial hash based
  - 3D point to a 1D index
  - Signed distance rule
- Logarithmic traversal time
- Per frame construction
  - Current agent's position

For each agent:

- Select random, 3D position samples

For each sample:

- Hash position
- Compute distance
- Insert, sort distance

Execution Model

- 1D grids and blocks
- Static shared memory
- Hide ALU ops latency
  - 10–12 cycles FMA
- Lessen memory latency
  - Independent math ops
- Per agent RNG

Kernel	Registers	Shared (B)	Local (B)	Constant (B)
simulate	32	116	244	208
update	14	60	0	56

Property	Kernel	
	simulate	update
Threads / Block	128	128
Warps / Multiprocessor	16	32
Occupancy	50%	100%

Nested Parallel

- Flat parallel limited
  - Nested more scalable
- Thread grid hierarchy
  - Independent child grids
  - All running same kernel
  - Grid global atomic sync
- Threads exceed HW max
  - No added memory

```
__global__ void
candidate(CUAgent* agents,  int index,
                CUNeighbor* neighbors)
{
   float3 v, float t;
   CUAgent a = agents[index];

   if(!getThreadId()) v = a.prefvelocity;
   else v = velocitySample(a);
   t = neighbor(a, agents, neighbors, v);

   float p = penalty(a, v, t);
   atomicMin(a.minpenalty, p);
   if(p == a.minpenalty)  a.candidate = v;
}
```

Results

Methodology

Environment

- Vista 32 bits, CUDA 2.1
- Simulation-only
- Flat parallel
- Copy to/from device included

Property	GTX280	X7350
Vendor	NVIDIA	Intel
Core Clock (MHz)	601	2930
Memory Clock (MHz)	1107	1066
Global Memory (MB)	1024	8192
Multiprocessors	30	4
Total Threads	500-20000	16

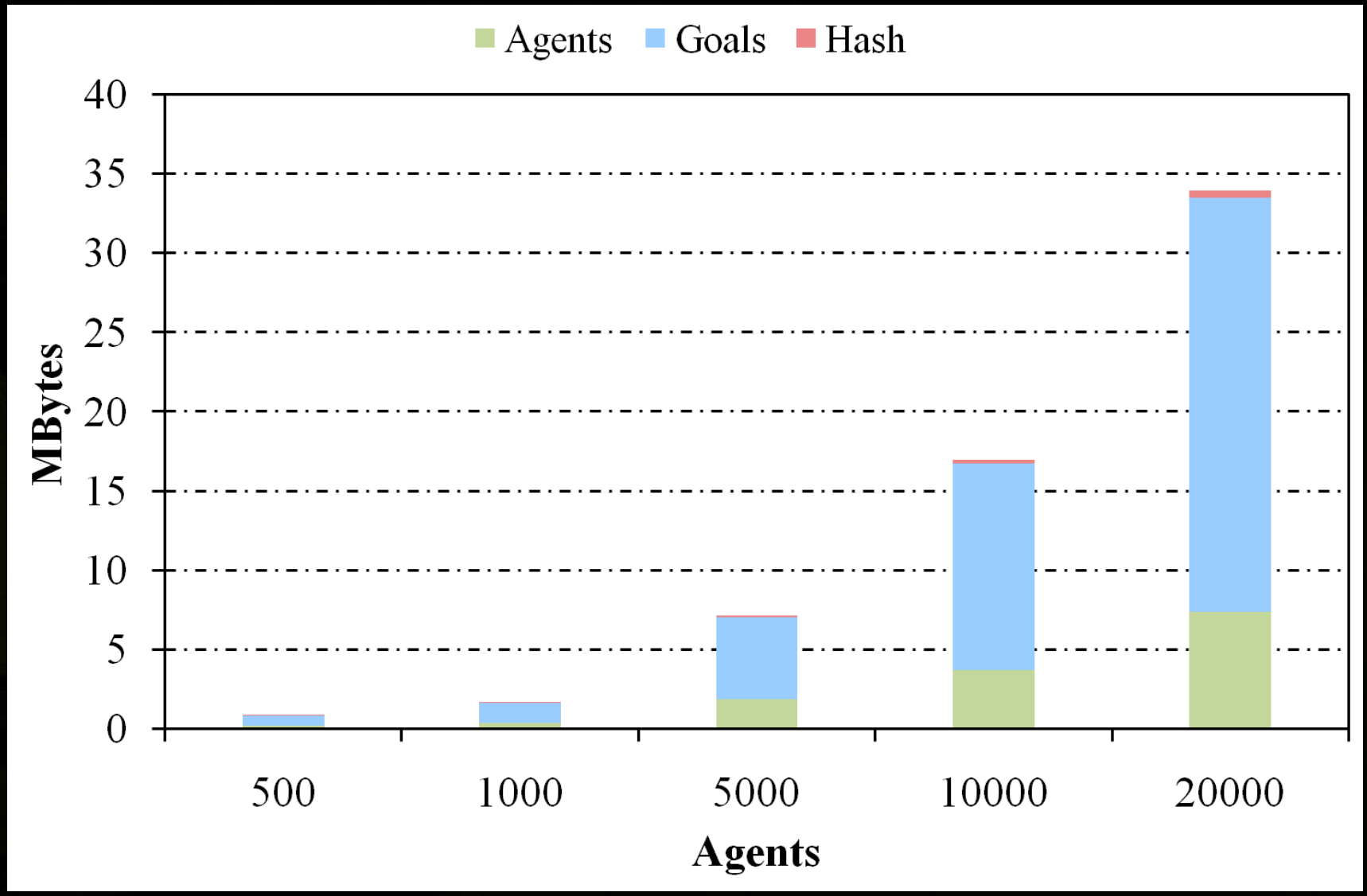
Experiments

Timestep	Proximity		Velocity Samples	Frames
	Neighbors	Distance		
0.1	10	15	250	1200

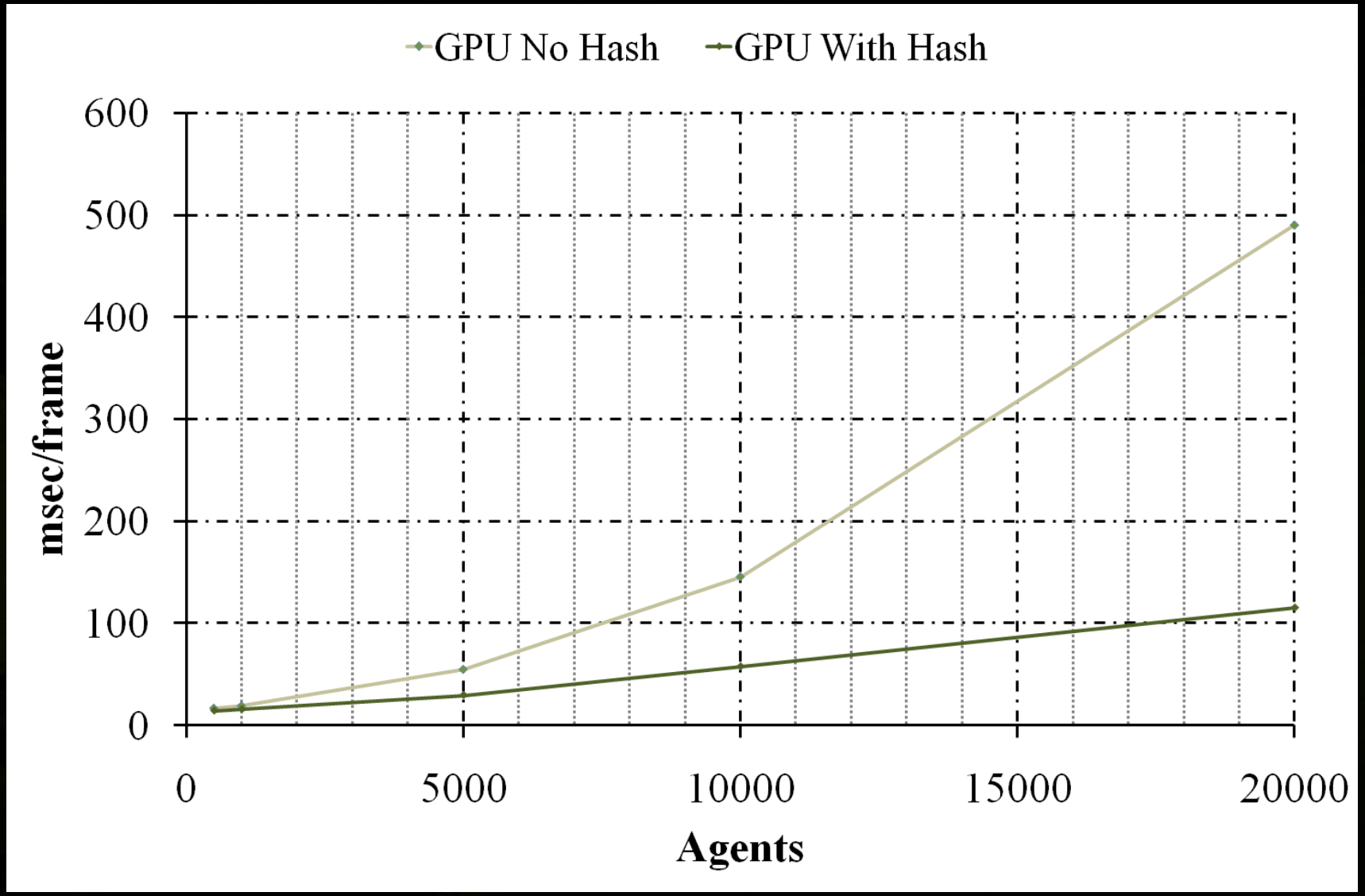
Dataset	Agents	Thread Blocks
Evacuation	500	4
	1000	8
	5000	40
	10000	79
	20000	157

Roadmap:
211 segments
429 nodes

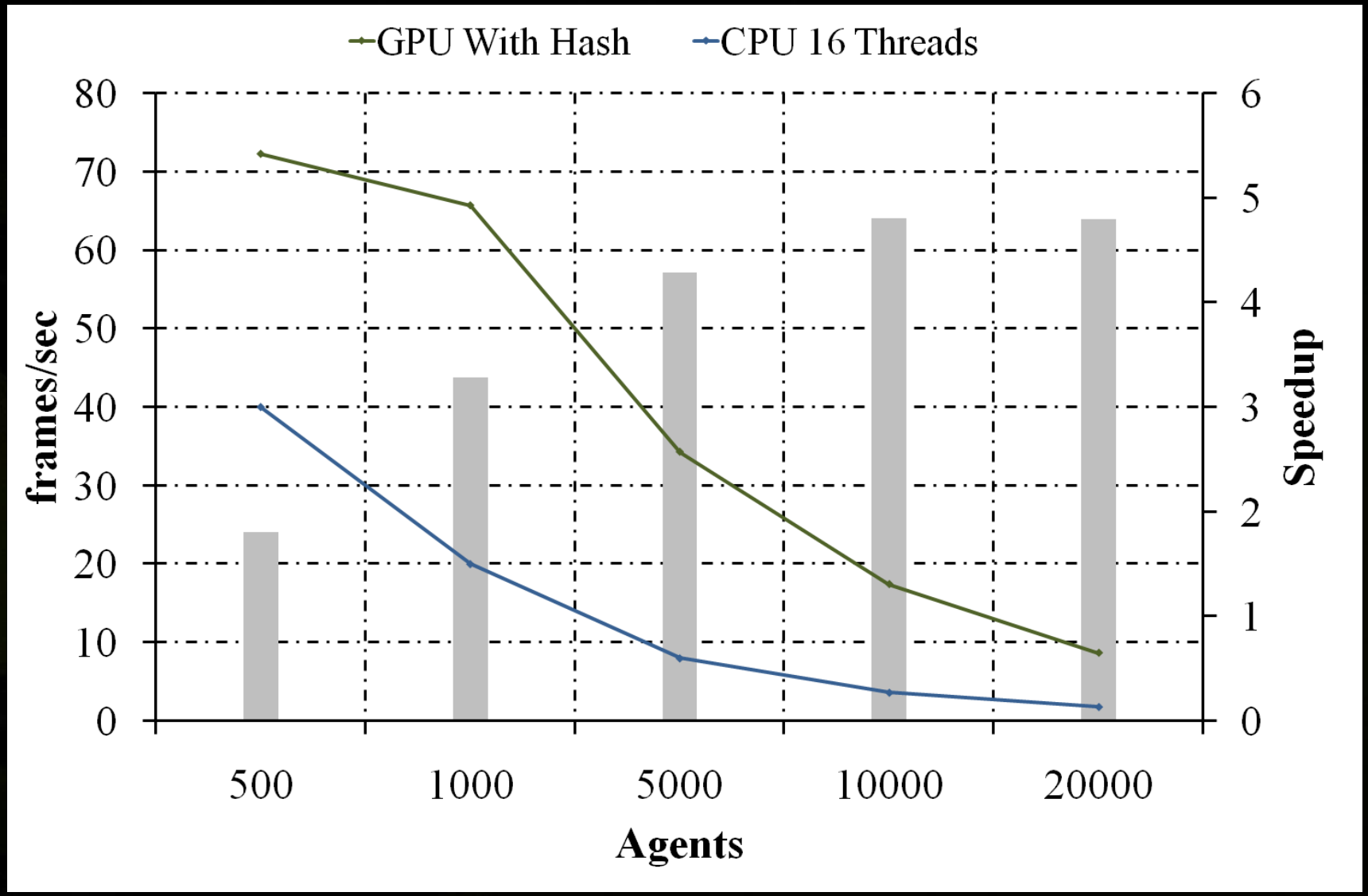
Footprint


Agents
Goals
Hash
40
35
30
25
20
15
10
5
0
MBytes
500
1000
5000
10000
20000
Agents

Running Time



Frame Rate


GPU With Hash
CPU 16 Threads
frames/sec
Speedup
80
70
60
50
40
30
20
10
0
6
5
4
3
2
1
0
500
1000
5000
10000
20000
Agents

Takeaways

Limitations

- Hash table construction
  - Single threaded
- Thread load imbalance
  - Non, at-goal agent mix
- Hash motion artifacts
  - Area under sampling
- Shared memory SW cache
  - Constraint, 32B per thread

Future Work

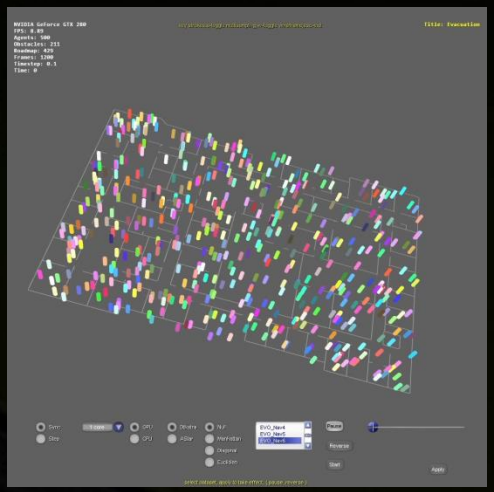
- Exploit shared memory
  - Further hide latency
- At-goal agent extraction
  - Unified thread block
- Up hash sampling quality
- Dynamic obstacles, goals
  - GPU visibility port

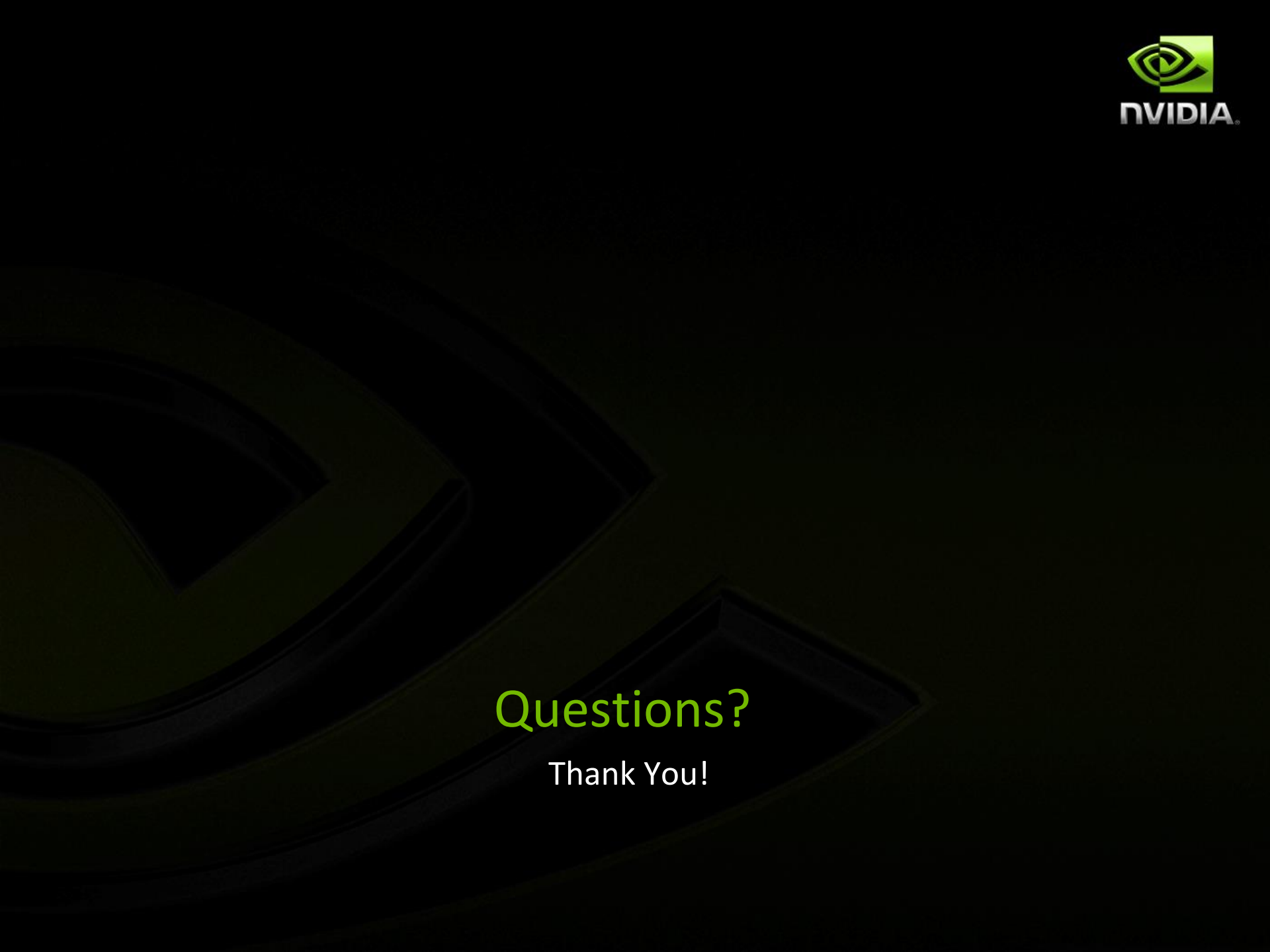
Performance

Parameter	NVIDIA GTX280	INTEL X7350
Hash Speedup	4X	Little to None
Simulation Acceleration	Up to 77X	Single thread
	Up to 4.8X	Sixteen threads
FPS (for 10K agents)	18	3.75
Nested vs. Flat	Up to 2X	Difficult to program
Cost ($)	399	2400

Summary

- Computational intelligence
  - Maps well on GPU
- Multi agent solution
  - Compact, scalable
  - Further optimization
- Nested data parallel
  - Multi GPU system
- AI, physics integration





Questions?

Thank You!

How To Reach Us

- Paper:
  - http://tinyurl.com/MultiAgentGPU-paper-2009
- During GDC
  - Expo Suite 656, West Hall
  - Developer Tool Open Chat, 1:30 to 2:30 pm (25th-27th)
- Online
  - Twitter: nvidiadeveloper
  - Website: http://developer.nvidia.com
  - CUDA: http://www.nvidia.com/cuda
  - Forums: http://developer.nvidia.com/forums