Multi Agent Navigation on GPU

Avi Bleiweiss
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

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]

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

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

\[ RV_O_B^A(v_B, v_A) = \{ v'_A | 2v'_A - v_A \in VO_B^A(v_B) \} \]
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
7: compute preferred velocity
8: compute proximity scope
9: foreach velocity sample do
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
```
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

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Registers</th>
<th>Shared (B)</th>
<th>Local (B)</th>
<th>Constant (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>simulate</td>
<td>32</td>
<td>116</td>
<td>244</td>
<td>208</td>
</tr>
<tr>
<td>update</td>
<td>14</td>
<td>60</td>
<td>0</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simulate</td>
</tr>
<tr>
<td>Threads / Block</td>
<td>128</td>
</tr>
<tr>
<td>Warps / Multiprocessor</td>
<td>16</td>
</tr>
<tr>
<td>Occupancy</td>
<td>50%</td>
</tr>
</tbody>
</table>
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

```c
__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

<table>
<thead>
<tr>
<th>Property</th>
<th>GTX280</th>
<th>X7350</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>NVIDIA</td>
<td>Intel</td>
</tr>
<tr>
<td>Core Clock (MHz)</td>
<td>601</td>
<td>2930</td>
</tr>
<tr>
<td>Memory Clock (MHz)</td>
<td>1107</td>
<td>1066</td>
</tr>
<tr>
<td>Global Memory (MB)</td>
<td>1024</td>
<td>8192</td>
</tr>
<tr>
<td>Multiprocessors</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>Total Threads</td>
<td>500-20000</td>
<td>16</td>
</tr>
</tbody>
</table>
# Experiments

## Dataset Agents Thread Blocks

<table>
<thead>
<tr>
<th>Timestep</th>
<th>Proximity</th>
<th>Velocity Samples</th>
<th>Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Neighbors</td>
<td>Distance</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>10</td>
<td>15</td>
<td>250</td>
</tr>
</tbody>
</table>

## Roadmap:

- 211 segments
- 429 nodes

## Samples Frames Neighbors Distance

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Agents</th>
<th>Thread Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evacuation</td>
<td>500</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>5000</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>157</td>
</tr>
</tbody>
</table>
Running Time

![Graph showing running time for GPU No Hash and GPU With Hash. The x-axis represents the number of agents, and the y-axis represents the time in milliseconds per frame. The graph shows a clear increase in running time as the number of agents increases for both GPU configurations.](image-url)
Frame Rate

The graph compares the frame rate for different numbers of agents between two methods:

- **GPU With Hash**
- **CPU 16 Threads**

The x-axis represents the number of agents, ranging from 500 to 20,000. The y-axis on the left shows frames per second (fps) for GPU and CPU. The y-axis on the right shows speedup for GPU compared to CPU.

As the number of agents increases, the frame rate for both methods decreases, but the GPU method maintains a higher frame rate than the CPU method, especially at higher agent counts. The speedup for the GPU with hash over the CPU with 16 threads increases significantly as the number of agents increases.
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

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

- 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