Path Planning System on the GPU

Avi Bleiweiss
NVIDIA Corporation
Reasoning

- Explicit
  - State machine, serial
- Implicit
  - Compute intensive
  - Fits SIMT well
- Path planning
Motivation

• GPU accelerated AI
• Congestion games
• Effective team tasks
  – Virtual robots, humans
• Scalable, real time
Problem

Planner
- Efficient roadmap construction
  - From 3D virtual environment
- Searches a global, optimal path
  - From start to goal
- Locally, avoids collisions with
  - Static, dynamic objects

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

• Compact, quality roadmap
• Heterogeneous agents
• Velocity Obstacles
• GPU optimizations
  – Spatial hash
  – Nested parallel
Outline

• Algorithm
• Implementation
• Results
• Takeaways
Pipeline

• 3D level, $C_{space}$ mesh input
• Inline computed roadmap
• Goals, roadmap decoupled
• Discrete time simulation
Roadmap Construction

• An existed $C_{\text{free}}$ path
  – Guaranteed in roadmap
• Predictable termination
• 3D grid operators
  – Highly parallelizable

[Geraerts and Overmars 2005]
Visibility

• Two sets of edges
  – Visible roadmap node pairs
  – Goals to unblocked nodes
• Static obstacles outline
• A* search, shortest path
  – From goal to any node
Velocity Obstacles

• Well defined, widely used
• Avoidance velocity set\(^1\)
• Reciprocal Velocity Obstacles\(^2\)
  – Oscillation free motion
• Agents moving in 2D plane

1 [Fiorini and Shiller 1998]
2 [Van Den Berg et al. 2008]
Multi Agent Simulation

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

\[
\text{do}
\]
\[
\begin{align*}
\text{hash} & \quad \text{construct hash table} \\
\text{simulate} & \quad \text{compute preferred velocity} \\
& \quad \text{compute proximity scope} \\
& \quad \text{foreach velocity sample do} \\
& \quad \quad \text{foreach neighbor do} \\
& \quad \quad \quad \text{if OBSTACLE then VO} \\
& \quad \quad \quad \text{elseif AGENT then RVO} \\
& \quad \quad \quad \text{resolve new velocity} \\
\text{update} & \quad \text{update position, velocity} \\
& \quad \text{resolve at-goal} \\
\text{while} & \quad \text{not all-at-goal}
\end{align*}
\]
Challenges

- Hiding memory latency
- Divergent, irregular threads
- Small agent count (≤32)
- Hash construction cost
Workflow

- Roadmap static for
  - 100s simulation steps
- Dependent resources
  - Linear, pitched 3D
- Dozen compute kernels
- Split frame, multi GPU
Medial Axis Transform

- Serial running time $O(kn^3)$
- $n^3$ GPU threads, per pass
  - $O(k)$ time for CDT
  - $O(1)$ for qualifier $T$
  - $O(1)$ for resolve

[Lee and Horng 1996]
Distance Transform

• Squared Euclidian distance
• Serial running time $O(n^3)$
• Parallel linear time $O(n)$
  – Slice, column, row passes
  – $n^2$ GPU threads, per pass

[Felzenszwalb and Huttenlocher 1996]
Flood Fill

- Obstacle aware
  - 3D line drawing
- Parallel guards
- Single cell, private stack
- Scan line stack smaller
  - Runs slower!

```
push guard on to stack
while stack not empty do
  pop stack
  if guard not visible from cell continue
  add guard to cell's coverage set
  foreach adjacent neighbor cell do
    if neighbor in \( \mathcal{C}_{\text{free}} \) && not covered do
      push neighbor on to stack
```

6 face neighbors
Data Layout

• Persistent resources
  – Reside in global memory
• Thread aligned data
  – Better coalescing
• Consistent access pattern
  – Improves bandwidth

Variable Length Vector Access
K-Nearest Neighbor

• Naïve, exhaustive search
  – $O(n^2)$ system running time

• Spatial hash
  – 3D point to a 1D index

• Per frame table build
  – Current agents’ position

$h(p) = \text{determinant}(p, p_{ref})$
Nested Parallel

• Flat parallel limiting
  – Independent grids
• Thread grid DAG
  – Same kernel per level
• Thread amplification
  – Improved occupancy

update

simulate

candidate

agent_{n-1}

... 

agent_2

agent_1

agent_0

n grids, each of \( v \) velocity threads

data dependency

\( \text{agent}_0 \) \( \text{agent}_1 \) \( \text{agent}_2 \) \( \text{agent}_{n-1} \)
Velocity Threads

- Hundreds of threads
- Graceful grid sync
- Fine reduce-min
  - Into Shared memory
- Global atomic CAS
  - Inter thread block

```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);
    reduceMinAtomicCAS(a, p);
    if(p == a.minpenalty) a.candidate = v;
}
```
Methodology

• CUDA 3.1 Beta
• GPU properties

<table>
<thead>
<tr>
<th>GPU</th>
<th>SMs</th>
<th>Warps/SM</th>
<th>Clocks (MHz)</th>
<th>L1/Shared (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX480</td>
<td>15</td>
<td>2</td>
<td>723/1446/1796</td>
<td>48/16</td>
</tr>
<tr>
<td>GTX285</td>
<td>30</td>
<td>1</td>
<td>648/1476/1242</td>
<td>NA</td>
</tr>
</tbody>
</table>

• Fermi scale\(^1\)

<table>
<thead>
<tr>
<th></th>
<th>compute</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.98</td>
<td>1.08</td>
</tr>
</tbody>
</table>

\(^1\) More info in appendix
Views

- Three views per stage
  - vs. GTX285
  - Relative throughput
  - vs. CPU
- Running time, frame rate
- Speedup vertical bars

<table>
<thead>
<tr>
<th>Property</th>
<th>GTX480</th>
<th>GTX285</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads / SM</td>
<td>1024</td>
<td>512</td>
</tr>
<tr>
<td>L1 Cache (KB)</td>
<td>48</td>
<td>None</td>
</tr>
<tr>
<td>L2 Cache (KB)</td>
<td>768</td>
<td>None</td>
</tr>
<tr>
<td>Parallel Kernels</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>
# Roadmap Construction Experiments

<table>
<thead>
<tr>
<th>Level, $C_{free}$ Vertices</th>
<th>Faces</th>
<th>Grid Resolution</th>
<th>GPU Threads Distance</th>
<th>Medial Axis</th>
<th>Graph Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>82800</td>
<td>34750</td>
<td>33</td>
<td>1089</td>
<td>35937</td>
<td>114</td>
<td>109</td>
</tr>
<tr>
<td>161463</td>
<td>64451</td>
<td>40</td>
<td>1600</td>
<td>64000</td>
<td>287</td>
<td>286</td>
</tr>
<tr>
<td>347223</td>
<td>170173</td>
<td>55</td>
<td>3025</td>
<td>166375</td>
<td>782</td>
<td>764</td>
</tr>
</tbody>
</table>

Images:
- **Mesh**
- **MAT**
- **DT**
- **Graph**
Roadmap Construction - vs. GTX285
Roadmap Construction - Throughput

![Graph showing throughput versus thread count for GTX480 (48/16). Higher throughputs are represented by a green line. The graph indicates an increasing trend with thread count.]
Path Searching Experiments

<table>
<thead>
<tr>
<th>Graph</th>
<th>Nodes</th>
<th>Edges</th>
<th>Agents</th>
<th>CTAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>5706</td>
<td>39156</td>
<td>1024—65536</td>
<td>4—256</td>
</tr>
</tbody>
</table>

Agents of random start and goal pair configurations
Path Searching - vs. GTX285

average query time = total running time / agent #
Path Searching - Throughput

A* Throughput vs Thread Count

- GTX480 (48/16)

Higher is good
Multi Agent Simulation Experiments

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Segments</th>
<th>Nodes</th>
<th>Agents</th>
<th>CTAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evacuation</td>
<td>211</td>
<td>429</td>
<td>500—20000</td>
<td>4—157</td>
</tr>
</tbody>
</table>
Multi Agent Simulation – vs. GTX285

![Graph showing performance comparison between GTX480 (48/16) and GTX285. The x-axis represents the number of agents, ranging from 500 to 20,000. The y-axis on the left shows frames per second, ranging from 0 to 100, and the y-axis on the right shows speedup, ranging from 0 to 1.8. The graph indicates a decrease in frames per second and an increase in speedup as the number of agents increases.]
Multi Agent Simulation - Throughput
Multi Agent Simulation - Distribution

Always fill SMs first, then threads in CTA!

3.3X speedup: 13, single warp SMs vs. 13 warps, single SM
Limitations

- Flood fill large stack
- A* I/O limited
- One thread, hash build
- Hash under sampling
- Thread load imbalance
  - Non, at-goal agent mix
## Fermi Performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Roadmap Construction</th>
<th>Path Searching</th>
<th>Multi Agent Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup vs. GTX285 (up to)</td>
<td>2.07X</td>
<td>1.52X</td>
<td>1.59X</td>
</tr>
<tr>
<td>Arch Gain vs. GTX285 (%)</td>
<td>91</td>
<td>40</td>
<td>47</td>
</tr>
<tr>
<td>Hash vs. Naïve (up to)</td>
<td>NA</td>
<td>NA</td>
<td>4X</td>
</tr>
<tr>
<td>Nested vs. Flat (up to)</td>
<td>NA</td>
<td>NA</td>
<td>6.2X</td>
</tr>
</tbody>
</table>

*Nested parallel limited to agent count <32*
Future Work

• 3D collision avoidance
• Shorter path extractions
• Complex behavior, flocking
• Parallel hash build
• Zero-Copy A*
Summary

• Multi agent solution
  – Compact, scalable
  – Fermi speedup
• Nested parallel potential
• Broad application set
Thank You!
Info

• SDK: foundation libraries, sample applications
  – Technology Preview

• Papers:
  – Scalable Multi Agent Simulation on the GPU, RA09
  – GPU Accelerated Pathfinding, GH08

• Video:
  – Simulation Clips
Appendix

• Compute scale

\[
\frac{SM_{Clk,GTX480}}{SM_{Clk,GTX285}} \cdot \frac{(Warp/SM)_{GTX480} \cdot SM_{S,GTX480}}{(Warp/SM)_{GTX285} \cdot SM_{S,GTX285}}
\]

• Memory scale

\[
\frac{Mem_{Clk,GTX480}}{Mem_{Clk,GTX285}} \cdot \frac{Mem_{BusWidth,GTX480}}{mem_{BusWidth,GTX285}}
\]

• GTX480 L1/Shared (KB) config
  – Up to 1.35X faster in 48/16 vs. 16/48
Backup
CPU

• Properties

<table>
<thead>
<tr>
<th>CPU</th>
<th>Cores</th>
<th>Clocks (MHz)</th>
<th>L1/L2 (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i7-940</td>
<td>8</td>
<td>2942/(3*1066)</td>
<td>32/8192</td>
</tr>
<tr>
<td>Intel X7350</td>
<td>4</td>
<td>2930/1066</td>
<td>32/8192</td>
</tr>
</tbody>
</table>

• C++ code
  – Not highly optimized

• Multi threading
  – OpenMP, Windows threads
Roadmap Construction - vs. CPU

![Chart comparing GPU and CPU performance]

- **GTX480 (48/16)**
- **i7 8 Threads**

The chart illustrates the performance of GPU and CPU in terms of **MAT Running Time** and **Speedup** across different **Level Mesh** values. The right side of the chart indicates that higher is good.
Path Searching vs. CPU

average query time = total running time / agent #
Multi Agent Simulation – vs. CPU

![Graph showing performance comparison between GTX480 (48/16) and X7350 16 Threads. The x-axis represents the number of agents, ranging from 500 to 20000. The y-axis on the left represents Frames/sec, while the y-axis on the right represents Speedup. The graph illustrates that GTX480 performs better than X7350 as the number of agents increases.]