S4229. Generation, Simulation and Rendering of Large Varied Animated Crowds

Benjamin Hernandez  Isaac Rudomin

March 2014
We are almost 7,000,000,000 humans with a limited surface of habitable land

- Confirms the need to cope with crowded environments

Observation and measurements of the crowd phenomenon are feasible

- But experimentation and formulation of hypotheses are not.
  
  Large-scale emergency evacuations.

Crowds produce lots of energy

- During gathering and scattering
- Some of them produce entropy, i.e. wasted energy
Why Crowd simulation?

Applications:
1. Videogames,
2. Special events or Emergency simulations,
3. Vehicular traffic,
4. Health etc…

What we learn here can be used in other applicationa like largescale simulations and visualization
Realtime crowd simulation and rendering of large crowds is inherently parallel, but even so, stresses the compute and graphics hardware of GPUs. We have developed methods for simulating and rendering animated crowds of varied aspect and a diversity of behaviors that work in PCs with consumer level graphic cards (GPUs).

- scalable multi agent system architecture
- supports simulation of hundreds or thousands of agents
- rendering all those as diverse, animated characters,
- all within good frame rates

We are working on applying similar principles in heterogeneous CPU/GPU clusters for large scale problems. Use parallelism on clusters for real-time simulation and visualization.
GOD – Generation of Diversity

- **Enables** geometrical and visual diversity
- Characters are generated **from body-part meshes**
- Takes **advantage** of **texture space parameterization**
  - Rig and skinning information is embedded into this space
  - Extra texture are used for visual and texture diversity
- Maintains **memory requirements low**
- XML scripting for authoring
- Rig and skinning information is embedded into **texture space parameterization**
- They can be used in **any of the characters** with compatible UV parameterization regardless of topology.
- Also for different **LODS**.
We begin by reducing a data set of virtual characters into simple body parts.
Crowds of similar characters
Impostors instancing

Varied crowds discrete LOD
Hierarchical LOD

MOBS: a skeleton + octree hierarchy for varied animatable characters, rendering polygons or points

Static quadtrees for the crowd, combining faraway characters
Following the approach in “Fragment shaders for agent animation using finite state machines” we use world space images to store: forces, distances, velocities, list of neighbors.

We can use
- Rasterizer vs Compute Hardware
- CUDA vs Compute Shader

We implement different collision avoidance methods:
- Boids (Reynolds),
- Social Forces (Helbing),
- Reciprocal Velocity Obstacles,
- Synthetic Vision

Using different proximity query techniques
- Discrete Truncated Voronoi
- Scatter vs Gather

Physical (speed) and psychological (personality) characteristics added to the agents, for accuracy.
COLLISION AVOIDANCE

Helbing + gather

Helbing + scatter
NAVIGATION USING MDP

\[ M = \langle S, A, T, R \rangle \]

**S** finite set of states, number of cells

Penalties and Rewards \( R(s) \).

Set of actions \( A \), i.e. agent directions

\( T(s, a, s') \), the probability with which an agent will choose state \( j \) from state \( s \) through action \( a \)

Optimal Policy \( \pi^* \)

Agent Density
We attached group behaviors to the simulated humans to get a plausible reaction from real people.
World: 2D grid of cells
- empty or
- occupied by an agent.

- 8 directions radius 5
- If another agent in path has same direction its cell is considered a free cell

main computation is agent and world updates:

with a single GPU once data is in GPU updating & rendering happens on the GPU without data transfers speedup is significant.
Parallel simulation needed for more agents
- world can be any size: used 16384x16384 or 27852x16384 and 4, 9, 16 nodes
- One MPI process is started per node
  - subdivide into equal sized (2D) tiles
  - assign each tile to a node.
  - interchange of agents at borders
  - areas manage their own agents

Example:
World divided into 9 tiles with 3 rows and 3 columns.
- To calculate the next position of agent requires the information in the cells around it
- i.e. agent located at coordinates (10, 9) in zone 5 needs to know if the cell at coordinate (11, 9) is empty

CUDA single node is better than sequential and better than multiple node:
- Updating 64 million agents takes 3 microseconds in one node but 5 seconds in 4
- However we can simulate 289 million agents with 4 nodes and 529 million agents with 9 nodes.
### Visualization

#### Filter data in frustum send to master or client for render

#### Nodes in frustum render their part and send images to compositor

<table>
<thead>
<tr>
<th>Frame Rates</th>
<th>Frame Size Mbits</th>
<th>Net Limit Gbit</th>
<th>Net Limit 100 Mbit</th>
<th>Net Limit 10 Mbit</th>
<th>Net Limit 5 Mbit</th>
<th>Net Limit 1 Mbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGL Q95</td>
<td>2.0</td>
<td>500</td>
<td>50</td>
<td>5.0</td>
<td>2.5</td>
<td>0.50</td>
</tr>
<tr>
<td>VGL Q80</td>
<td>0.85</td>
<td>1200</td>
<td>120</td>
<td>12</td>
<td>6.0</td>
<td>1.2</td>
</tr>
<tr>
<td>VGL Q3</td>
<td>0.43</td>
<td>2300</td>
<td>230</td>
<td>23</td>
<td>12</td>
<td>2.3</td>
</tr>
<tr>
<td>VGL RGB</td>
<td>25</td>
<td>40</td>
<td>4</td>
<td>0.40</td>
<td>0.20</td>
<td>0.04</td>
</tr>
<tr>
<td>T-VNC Q95</td>
<td>2.2</td>
<td>450</td>
<td>45</td>
<td>4.5</td>
<td>2.3</td>
<td>0.45</td>
</tr>
<tr>
<td>T-VNC Q80</td>
<td>1.0</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>5.0</td>
<td>1.0</td>
</tr>
<tr>
<td>T-VNC Q30</td>
<td>0.57</td>
<td>1800</td>
<td>180</td>
<td>18</td>
<td>8.8</td>
<td>1.8</td>
</tr>
<tr>
<td>T-VNC RGB</td>
<td>34</td>
<td>29</td>
<td>2.9</td>
<td>0.29</td>
<td>0.15</td>
<td>0.029</td>
</tr>
</tbody>
</table>
VISUALIZATION (CLIENT)

System overview:

- Laptop: Pandora Framework
- Minotaur Supercomputer: Crowd Simulation on MPI and CUDA
- Streaming Agent Positions
- OSC Protocol
- Visualization Server (Desktop NV TITAN)
  - Crowd Visualization Engine
  - OpenGL Rendering
VISUALIZATION (IN SITU + COMPOSITING)
CONCLUSIONS

We have a scalable multi agent system architecture

- Supports the simulation of hundreds of thousands of autonomous agents

- The crowd rendering engine enables geometrical, visual and animation diversity while maintaining memory requirements low.

- We have used GLSL/CUDA for data parallelism for systems with one GPU

- We are developing techniques for large scale simulations:
  - Taking advantage of heterogeneous computing clusters with multiple CPUs and GPUs
  - Enabling real-time simulation and in-situ visualization on the clusters using MPI.
Questions?
- email:
  benjamin.hernandez@bsc.es
  isaac.rudomin@bsc.es

Team Members
Sergio Ruiz (ITESM-CCM, Mexico)
Leonel Toledo (ITESM-CEM, Mexico)
Oriam De Gyves (ITESM-CEM, Mexico)
Jorge Rivalcoba (ITESM-CEM, Mexico)
Hugo Perez (UPC-BSC, Spain)