GPU Crowd Simulation

Jeremy Shopf
AMD, Inc.

Christopher Oat
AMD, Inc.

Joshua Barczak
AMD, Inc.

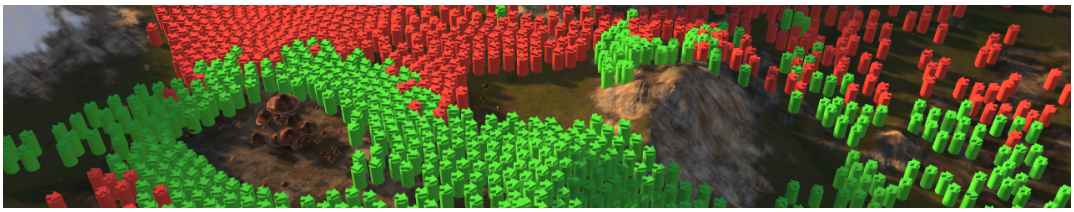


Figure 1: *Agents traverse a terrain while avoiding obstacles and each other. Color indicates agents' goal group.*

1 Abstract

We present a GPU-friendly path-planning framework for large-scale crowd simulation (Figure 1). This framework has been used to simulate 65,000 agents at real-time framerates on a single commodity GPU. By combining a continuum-based global path planner with a fine-grained avoidance model, we can perform expensive global planning at a coarse resolution and lower update rate while the local avoidance model takes care of avoiding other agents and nearby obstacles at a higher frequency. To our knowledge, this is the first massive crowd simulation performed entirely on a GPU.

2 Global Path Planning

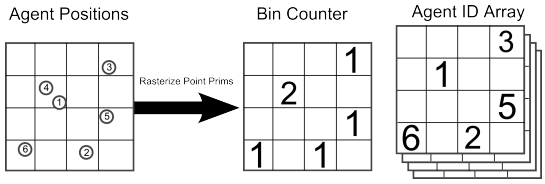
We chose a continuum-based approach similar to the Continuum Crowds work by Treuille et al. [2006]. This type of method is particularly well suited for simulating large numbers of agents because it is computed *spatially*, instead of per-agent, and results in smooth movement with no "dead-ends." The environment is represented with a cost function. This cost function is then used as input to a GPU-based eikonal solver [Jeong and Whittaker 2007] that calculates the travel-time (potential) from any location to the nearest goal. By following the gradient of this potential field, agents are guaranteed to always be moving along the shortest path to the global goal, considering the speed at which an agent can travel based on terrain features, obstacles and agent density.

3 Local Avoidance

Unfortunately, solving the eikonal equation at a resolution high enough for large numbers of agents to avoid each other with acceptable fidelity is prohibitively expensive for a real-time application. We augment our global eikonal solution with a local avoidance model that allows agents to avoid each other and small-scale obstacles. This avoidance model computes agent velocities by examining the direction determined by the global model and the positions and velocities of nearby agents. This avoidance model is based on the Velocity Obstacle formulation [Fiorini and Shiller 1998].

3.1 Spatial Queries

Determining the positions and velocities of dynamic local obstacles requires a spatial data structure containing all obstacle information in the simulation. We developed a novel multi-pass algorithm for sorting agents into spatial bins. Agents are rasterized as point primitives into a bin based on their locations. Agent IDs are stored in a depth texture array and bin loads are stored in a color buffer. The GPU's depth-test unit ensures that agent IDs are inserted into bins in sorted order and the alpha blend unit is used to increment bin load counters as agents are binned. Agents are binned in parallel but only a single agent is placed into a particular bin per iteration.



As agents are placed into bins, they are removed from the working-set using Direct3D10 *stream-out*. This prevents agents from being re-processed during subsequent iterations. We repeat this process once for each bin slot, using predicated rendering to terminate early once all elements have been binned. The spatial data structure is queried by first reading the bin load (n) from the color buffer and then loading n agent IDs from the depth texture array.

3.2 Direction Determination

Each agent evaluates a number of fixed directions relative to the goal direction determined by the global solution. We used five directions in our application but more can be used for increased motion fidelity. Each direction is evaluated to determine the time to collision with agents in the current or adjacent bins. Each direction is given a fitness function based on the angle relative to the desired global direction and the time to collision. Time to collision is determined by evaluating a swept circle-circle collision test, in which the radius of each circle is equal to the radius of the bounding circle of the associated agent. The updated velocity (Equation 3) is then calculated based on the direction with the largest fitness function result (Equation 2) and the smallest time to collision in that direction.

$$fitness(vp_i) = w_i t(vp_i) + (v_i \cdot vp_i) \cdot .5 + .5 \quad (1)$$

$$v_i = \arg\max_{vp_i \in V} fitness(vp_i) \quad (2)$$

$$v_{final} = \hat{v_i} * \min(s_a, s_a t(\hat{v_i}) / \bigtriangledown ft) \quad (3)$$

where w_i is a per-agent factor affecting the preference to move in the global direction or avoid nearby agents, $t(x)$ returns the minimum time-to-collision with all agents in direction x, V is the set of discrete directions to evaluate, v_i is the global direction, s_a is the speed of agent a, and $\bigtriangledown ft$ is time-delta since the last simulation frame.

References

FIORINI, P., AND SHILLER, Z., 1998. Motion planning in dynamic environments using velocity obstacles.

JEONG, W.-K., AND WHITTAKER, R. 2007. A fast eikonal equation solver for parallel systems. *SIAM Conference on Computational Science and Engineering*.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. *ACM Trans. Graph. 25*, 3, 1160–1168.