24

Efficient Crowd Simulation for Mobile Games

Graham Pentheny

24.1 Introduction

Crowd simulation is a topic of ongoing exploration and experimentation in the game AI industry [Pelechano et al. 07, Sung et al. 04]. Modern games are filled with more and more AI-controlled agents. It is therefore imperative to create a movement system that is realistic, robust, and designer-friendly.

Traditional pathfinding approaches compute separate paths for individual agents, even though many paths may have similar sections. These redundant path calculations inhibit simulations of large numbers of units on mobile hardware.

The mobile tower defense game *Fieldrunners 2* used a combination of vector flow fields and steering behaviors to efficiently simulate thousands of agents, referred to as units. This article will describe the systems of flow-field generation, flow sampling, and unit movement employed by *Fieldrunners 2*. The process of constructing and balancing a dynamic crowd simulation system will be described in detail from the ground up.

24.2 Grid

The grid provides a discretization of the game world and defines the areas within which units may travel. For *Fieldrunners 2,* a grid cell was sized slightly wider than the widest unit, so that every computed path was traversable by every unit. Each grid cell can either be *open*, indicating that a unit may pass through it, or *blocked* indicating that the cell is impassible.

24.3 Flow Field

Units move through the grid following a static vector *flow field*. The flow field represents the optimal path direction at every cell in the grid, and is an approximation of a continuous *flow function*. Given a set of destination points, the flow function defines a vector field of normalized vectors, indicating the direction of the optimal path to the nearest destination. The flow function is similar to common methods for describing flows in fluid dynamics [Cabral and Leedom 93], with the difference that all flow vectors are normalized. Given this definition, we can define a flow field to be a discretization of a *flow function*.

Flow fields guide units to the nearest destination in the same manner as a standard pathfinding system; however, the units' pathing information is encoded in a flow field, removing the need for units to compute paths individually.

The vector flow field is specific to each set of potential destinations and thus can be used by all units sharing a set of destination points. Because the flow field expresses pathing information for the entire game world, it does not need to be updated unless the pathable areas of the grid or the set of destination points changes.

For example, if a bridge across a river is destroyed, the flow field only needs to be recomputed once to account for the change to pathable areas. Units following that flow field will implicitly change their respective paths in response to the change in the game world.

The flow field is comprised of a single normalized vector for each grid cell, as shown in Figure 24.1. A flow field and a unique set of destination points together are called a *path*. For example, a path corresponding to an *m* by *n* grid is a set of *m*n* normalized vectors and a set of one or more destination points. Due to the number of vectors required to represent the flow function, this approach can potentially yield prohibitively high memory usage. Memory consumption is linearly dependent on the product of the number of grid cells and the number of independent paths. Maps in *Fieldrunners 2* were restricted to three unique paths at most, and map grid sizes were small enough that flow-field memory usage did not prove to be a significant issue.

The grid size, and thus the resolution of the flow field, does not need to be high to yield believable movement characteristics. Using bilinear interpolation, a continuous flow function can be approximated from the four closest vectors in the low-resolution flow field [Alexander 06]. As the grid resolution increases, the flow field computes a higher and higher sampling of the same flow function. Bilinear interpolation of vectors in a flow field improves the continuity and organicity of unit paths.

24.4 Generating the Flow Field

The flow field is generated via a modified traditional point-to-point pathfinding function. The algorithm used in *Fieldrunners 2* was based on Dijkstra's algorithm

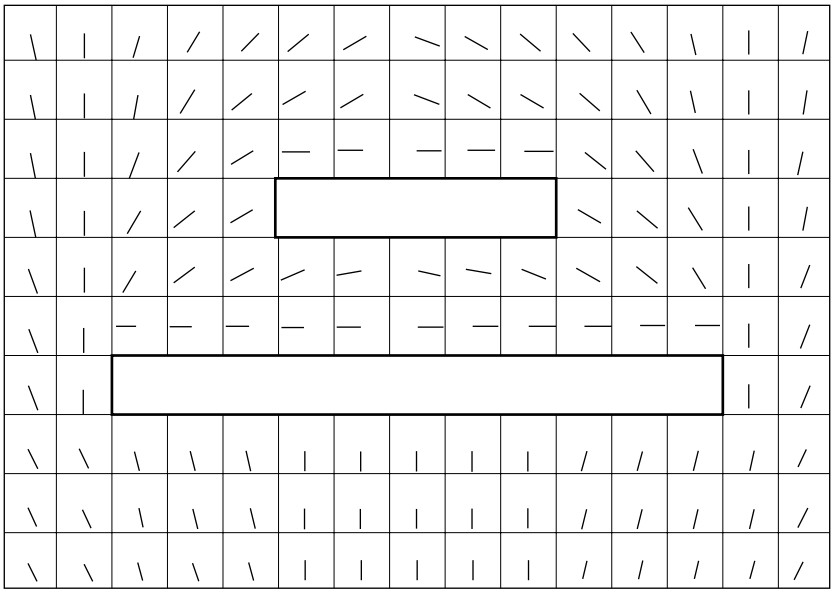

Figure 24.1

A flow field for a sample environment. The flow for a given cell is shown as a line starting at the center of the cell and pointing in the flow direction. Blocked cells are outlined with a thick black line. This particular flow field moves characters around the two rectangle obstacles and towards a destination along the bottom edge of the environment.

Dijkstra [59]; however, alternate pathfinding algorithms are aptly capable of generating a flow field.

The algorithm used in *Fieldrunners 2* begins by adding the grid cells for each of the paths' destinations to the open list. As the normal iterations of Dijkstra's algorithm progress, nodes are removed from the open list and linked to a nearby cell with the lowest computed path cost. As cells from the open list are expanded, the flow vector for the newly expanded cell is set to point in the direction of the cell it was linked to. Instead of terminating when a path is found, the algorithm expands all traversable cells added to the open list, assigning a flow vector to each, and terminating when the open list is empty. The demo code included with this article on the book's website (http://www.gameaipro.com) contains a full implementation of flow-field generation within the `GenerateFlowField()` function.

This preceding algorithm is used to generate a flow field for each path every time a change is made to either the path's destination set or the traversable area of the grid.

24.5 Units

Fieldrunners 2 required a crowd dynamics system capable of supporting dozens of different units, each with unique movement characteristics. Units in *Fieldrunners 2* are simple autonomous agents based on Craig Reynolds' *Boid* model [Reynolds 99]. Each unit has a set of both physical attributes and steering behaviors that together control its movement.

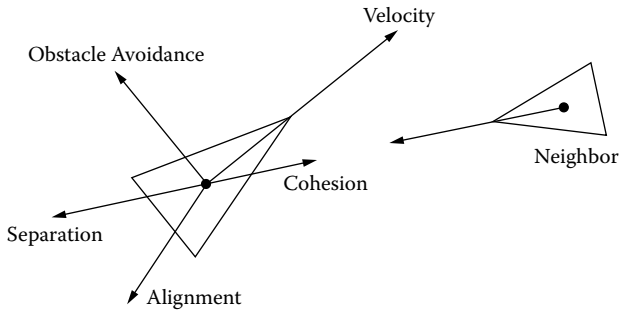

Figure 24.2

This figure shows all the steering forces acting on the left unit, with the exception of flow-field following.

The set of steering behaviors and their implementations are consistent across all unit types. A unit's physical attributes (e.g., total mass, size, agility) define its unique movement characteristics.

Units in *Fieldrunners 2* are represented as point-masses with respective velocities. A unit's steering behaviors control its point-mass by applying a set of forces to it. The prioritized combination of these steering forces imparts an acceleration on the unit, resulting in realistic, perceptively intelligent movement. Steering behaviors are widely used in games to control unit movement, and are described in numerous publications [Reynolds 99, Millington and Funge 09]. In *Fieldrunners 2*, specific modifications were made to the standard implementations of some steering behaviors to support more dynamic unit interactions.

Units in *Fieldrunners 2* use a limited, greedy, prioritized summation of five steering behaviors (four of which are shown in Figure 24.2). The five behaviors listed in descending order of priority include flow-field following, obstacle avoidance, separation, alignment, and cohesion. In each simulation step, a unit is only influenced by a specified total magnitude of steering forces. The forces resulting from steering behaviors are added to the running total in priority order until the maximum magnitude has been reached. Any steering forces that have not been added to the total are ignored.

The separation, alignment, and cohesion steering forces together describe flocking behavior [Reynolds 99]. In *Fieldrunners 2*, flocking is used to encourage units to move cohesively as a group when near other units.

Obstacle avoidance helps faster units maneuver intelligently around slower units. The implementations of the obstacle avoidance and separation behaviors differ slightly from Reynolds' original implementation [Reynolds 99]. The obstacle avoidance steering behavior generates a "side stepping" force perpendicular to the unit's velocity, and proportional to the position and relative velocity of the neighbor. The force generated by the separation steering behavior is scaled by the ratio of the kinetic energy of the neighbor to the kinetic energy of the unit the force is applied to. Units with smaller masses and velocities (presumably being more nimble) will more readily yield to larger, less maneuverable units.

Finally, flow-field following moves the unit in the direction specified by the flow field. The flow-field direction at the position of the unit is computed by linearly interpolating the four closest flow vectors.

The mass, maximum force, maximum velocity, and neighbor radius attributes describe a unit's unique behavior. The mass is used to calculate the unit's kinetic energy in addition to the accelerations resulting from steering behaviors. The maximum force value dictates the maximum combined magnitude of steering forces that can influence the unit in a single simulation step. A unit's agility value is defined as the ratio of a unit's maximum force to its mass—the unit's maximum acceleration. Finally, the maximum velocity attribute limits the magnitude of the unit's velocity, and the neighbor radius attribute restricts the set of neighbors used in calculating flocking forces to those within a certain radius.

24.6 Adjusting Unit Movement Values

It is necessary to find the correct set of attribute values for a unit to yield a desired behavior. In simulations based on steering behaviors, this is notoriously difficult and arduous. A systematic approach was developed and used by designers to balance unit attributes in *Fieldrunners 2*. For reasons of simplicity, all units used an identical set of weighted, prioritized steering behaviors, relying on their physical attributes for unique behavior.

First, the maximum velocity attribute is set to a reasonable value, and the remaining attributes are given an arbitrary base value. Because the maximum velocity of a unit is most easily visualized, it provides a good starting point. The remaining values will each be adjusted individually.

Next, the maximum force value is adjusted to yield believable movement characteristics for a single unit of that type. Because the maximum force affects the agility of the unit, it will alter visual aspects such as turning speed and braking.

Given a group of homogeneous units, the result of changes to the unit's neighbor radius attribute can easily be observed in isolation. Smaller neighbor radiuses will allow units to cluster more closely, while increasing the neighbor radius will spread units out.

Finally, all units' masses are adjusted relative to each other. When adjusting the mass of a unit, its agility must remain constant, or the previously adjusted movement characteristics of the unit will change.

24.7 Mobile Limitations and Performance Considerations

The largest runtime performance issue in this approach is the generation and processing of neighboring unit lists used in computing the flocking steering forces. In *Fieldrunners 2*, performance issues were mitigated through use of a *loose quad tree* [Ulrich 00] to reduce the neighboring unit search space. Units with large neighbor radiuses will yield a large set of neighbors to consider, decreasing performance. Combining the calculations of flocking forces can provide measurable performance improvements, as intermediary values can be reused in subsequent computations.

Floating-point operations on mobile processors can be slow, and minimizing the number of operations required in pathing and movement calculations can also yield improvements. Storing scalars that represent vector magnitudes as their respective squared values was a common optimization in *Fieldrunners 2*. This allowed vector length comparisons to use the squared vector magnitude, removing the need to compute many floating-point square root values.

Flow-field-based systems provide the greatest benefits when large numbers of units need to navigate to a set of common goals. A separate flow field is required for each unique set of goal positions among units. As the number of unique sets of goals increases, the calculations and memory required to maintain the necessary flow fields can become prohibitively complex and large. The memory required to represent a flow field grows linearly with the number of grid cells, while the pathfinding computational complexity is equivalent to the worst-case complexity of the pathfinding algorithm used. In the case of *Fieldrunners 2*, Dijkstra's algorithm was used, which yielded quasi-linear time complexity dependent on the number of grid cells. One approach to minimizing flow-field memory consumption is to save flow vectors as a specific rotation of the "north" vector (usually `<0,1>`). When accessing the flow direction for a given cell, the known basis vector is recreated and rotated the amount specific to that cell. Alternatively, if flow vectors are restricted to specific directions (e.g., cardinal directions), they can be stored as a one byte integer where its value corresponds to the specific potential direction.

As mobile hardware moves towards multicore processors, correct utilization of multithreaded algorithms becomes important. The problem of generating multiple flow fields can easily be modeled as a parallel process. The mutual independence of flow fields allows each to be computed in parallel with the rest, potentially in different threads or processes. The composition and independence of steering behaviors allows them to be computed in parallel as well, so long as they're accumulated and applied to the unit collectively. Together, the intrinsic parallelizability of flow-field generation and steering behavior computation make multithreading optimizations trivial.

24.8 Benefits

This approach to unit movement was chosen for *Fieldrunners 2* due to a specific set of unique benefits that it provided. Pathing information is precomputed and stored in the flow field; thus it is only ever calculated once for a given world configuration. This property of flow fields offered notable performance benefits in *Fieldrunners 2*, as the pathability of the world is modified infrequently.

Pathing information for all locations in the world is computed in a single pass, yielding a grid size-based complexity comparable to Dijkstra's algorithm. Compared to traditional pathfinding methods where the time complexity is linear with respect to the number of units, this approach is constant with respect to the number of units simulated. For *Fieldrunners 2*, this enabled complex scenarios with thousands of independent and diverse units to run on mobile devices at interactive frame rates.

Steering behavior-based approaches like this one provide great flexibility in defining unique unit behavior. Steering behaviors rely on composition to define complex behavior, making specializations and additions modular and encapsulated. The composition of a new steering behavior or the modification of an existing steering behavior can both easily be applied to a unit to define a unique movement style.

24.9 Conclusion and Future Work

The system used in *Fieldrunners 2* used static vector flow fields and steering behaviors to simulate thousands of units on mobile devices. Unlike traditional pathfinding techniques,

the proposed navigation system minimizes redundant path calculations by encoding pathing information from all areas in a vector flow field.

Flow-field-based pathfinding techniques provide a unique way to reduce redundant pathfinding computations by computing the optimal path from every point. The flow-field generation technique used in *Fieldrunners 2* was based on Dijkstra's algorithm for simplicity and design reasons. More advanced pathfinding algorithms, such as Theta* [Nash et al. 07], can generate smoother, more organic flow fields. Flow fields can be extended to incorporate alternate motivations and concerns for units by blending static and dynamic flow fields [Alexander 06]. Despite this potential improvement, static flow fields and steering behaviors provided a robust, realistic crowd simulation for *Fieldrunners 2*.

References

[Alexander 06] B. Alexander. "Flow fields for movement and obstacle avoidance." In *AI Game Programming Wisdom 3*, edited by Steve Rabin, pp. 159–172. Boston, MA: Charles River Media, 2006.

[Cabral and Leedom 93] B. Cabral and L. Leedom. "Imaging vector fields using line integral convolution." *SIGGRAPH '93 Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 263–270, 1993. Available online (http://www.cg.inf.ethz.ch/teaching/scivis_common/Literature/CabralLeedom93.pdf).

[Dijkstra 59] E. Dijkstra. "A note on two problems in connexion with graphs." *Numerische Mathematik* 1, pp. 261–271, 1959. Available online (http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf).

[Millington and Funge 09] I. Millington and J. Funge. *Artificial Intelligence for Games*, pp. 55–95. Burlington, MA: Morgan Kaufmann, 2009.

[Nash et al. 07] A. Nash, K. Daniel, S. Koenig, and A. Felner. "Theta*: Any-angle path planning on grids." *Proceedings of the AAAI Conference on Artificial Intelligence (2007)*, pp. 1177–1183, 2007. Available online (http://idm-lab.org/bib/abstracts/papers/aaai07a.pdf).

[Pelechano et al. 07] N. Pelechano, J. M. Allbeck, and N. I. Badler. "Controlling individual agents in high-density crowd simulation." *SCA '07 Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 99–108, 2007. Available online (http://www.computingscience.nl/docs/vakken/mpp/papers/12.pdf).

[Reynolds 99] C. W. Reynolds. "Steering behaviors for autonomous characters." *Proceedings of the Game Developers Conference (1999)*, pp. 763–782, 1999. Available online (http://www.red3d.com/cwr/papers/1999/gdc99steer.pdf).

[Sung et al. 04] M. Sung, M. Gleichar, and S. Chenney. "Scalable behaviors for crowd simulation." *Computer Graphics Forum*, Volume 23, Issue 3, pp. 519–528. September 2004. Available online (http://www.computingscience.nl/docs/vakken/mpp/papers/21.pdf).

[Ulrich 00] T. Ulrich. "Loose octrees." In *Game Programming Gems*, edited by Mark DeLoura, pp. 444–453. Hingham, MA: Charles River Media, 2000.