UU Crowd Simulation Software: Theory and Implementation

Wouter van Toll

Summerschool “Multidisciplinary Game Research”
August 23, 2017
About me

- **Education at Utrecht University**
  - BSc Computer Science
  - MSc Game and Media Technology
  - PhD: Path planning and crowd simulation
  - Main programmer of our crowd simulation software

- **Now: Lecturer at UU**
  - Game programming (1st year BSc)
  - Geometric algorithms (MSc)
This presentation

- Overview of our theoretical **framework**
  - Five levels of crowd simulation

- Overview of how our **software** works
  - Later, you’ll use our Unity3D plugin...
  - ...and you’ll know what is happening in the background!
Software with two main tasks
- Constructing the ECM **navigation mesh**
- Simulating crowds in real-time

Written in C++

API in C
- DLL can be plugged into other software
- Unity3D integration: topic of many UU software projects
UU Crowd Simulation Software
Algorithms in a crowd simulation framework

Theory
Crowd simulation framework

- **Overview paper**

- **Our software focuses on levels 4/3/2**
  - Geometric components
4: Global path planning

- We use the **Explicit Corridor Map**
  - Compact navigation mesh
  - Supports any character radius
  - Multi-layered environments
  - Dynamic updates

- Other options could also work
  - As long as characters can compute
3: Path following

▪ Smoothly follow a desired path
  – Input: indicative route, non-smooth indication of the path
  – In each simulation step, compute an attraction point
  – Leads to a preferred velocity for the next level

▪ Indicative Route Method (IRM, 2009)

▪ MIRAN (#7): improvement by Jaklin et al. (2013)
  – Supports weighted regions
  – Better smoothness / shortcut control
2: Local movement

- Roughly move in the preferred direction, while...
  - ...avoiding future collisions with other characters
  - ...responding to collisions that have already happened
  - ...adapting to the surrounding streams of people
  - ...maintaining social group behavior
  - ...
UUCS in action

Demo time
Demos

- ECM
- Single agent
- Crowd
- Circle
Implementation details
It’s hard to judge a method in practice

- Not all parameter settings are described in a paper
- Lots of details are up to the programmer
- Some necessary hacks are not mentioned
- Small changes can have huge impacts
Simulation step

- Core of the crowd simulation engine
- `performStep(Δt)`
  - Increase total time by `Δt`
  - For each character: *path following*
    - Update pointers along indicative route (if any)
    - Compute new attraction point, preferred velocity
  - For each character: *collision avoidance*
    - Compute new velocity `vNew`
    - Smoothen `vNew` (optional)
    - Compute collision forces `F` (optional)
  - For each character
    - Update velocity: `v := vNew + Δt \cdot F/mass`
    - Update position: `p := p + Δt \cdot v`
  - Update nearest-neighbor data structure

(There are actually many more *substeps*)

Why separate loops?

→ **Order** of characters does not matter

→ Characters are independent, each loop can be parallelized
Simulation loop

- Update the simulation per time step: `performStep(\Delta t)`

- Possible approach: do as many steps as possible
  - Use time between frames as step size
  - Use FPS to measure efficiency

- We use a fixed time step (0.1 s)
  - Use computation time to measure efficiency
  - Behavior does not depend on the framerate
  - Fast-forward by calling `performStep()` more/less often
  - Visualization framerate can be much higher

Is this a good idea? 😐
Plug-in / API

- “Black box” with basic entry functions
  - Compute navigation mesh
  - Add/Remove character
  - Plan a path for a character
  - Set character parameters
  - Advance the simulation by a single step
  - ...

- Written in C
  - Unity3D: C# wrapper around the API
Closing comments
Crowd simulation is a complex problem

We split it into easier subproblems (levels)
  – E.g. route planning, route following, local movement
  – Many possible algorithms for each level

Implementation in C++
  – ECM navigation mesh
  – Real-time crowd simulation
  – API in C for integration into Unity3D