PA199
Advanced Game Design

Lecture 12

Crowd Simulation

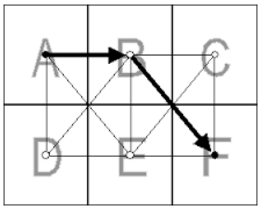
Dr. Fotis Liarokapis

12th May 2015

Path Planning

What is Path Planning

- Steer character from one point (or position) to another one



Computer Games and Path Planning

- Fast and flexible
  - Real-time planning for thousands of characters
  - Flexibility to avoid other characters and local hazards
  - Individuals and groups
- Natural Paths
  - Smooth
  - Collision-free
  - Short
  - Keep a preferred amount of clearance from obstacles
  - Flexible

Typical Algorithms

- Dijkstra's Algorithm
- Bellman-Ford Algorithm
- Johnson's Algorithm
- Greedy Best-First-Search Algorithm
- A* Algorithms

Dijkstra's Algorithm

- Finding the shortest paths between nodes in a graph, which may represent, for example, road networks
  - The algorithm exists in many variants
- Dijkstra's original variant found the shortest path between two nodes
- A more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph
  - Producing a shortest path tree

Dijkstra's Algorithm Graph

- For a graph: G = (V,E)
  - V is a set of vertices and
  - E is a set of edges
- Dijkstra's algorithm keeps two sets of vertices:
  - S the set of vertices whose shortest paths from the source have already been determined
  - V-S the remaining vertices
- The other data structures needed are:
  - d array of best estimates of shortest path to each vertex
  - pi an array of predecessors for each vertex

Dijkstra's Mode of Operation

- 1. Initialise d and pi,
- 2. Set S to empty,
- 3. While there are still vertices in V-S
  - i. Sort the vertices in V-S according to the current best estimate of their distance from the source,
  - ii. Add u, the closest vertex in V-S, to S,
  - iii. Relax all the vertices still in V-S connected to u

Dijkstra's Algorithm Unity Video

https://www.youtube.com/watch?v=dhvf9KCAsVg

Xlent Games - Pathfinding Test

https://www.youtube.com/watch?v=rIQ8-7k_kek

Bellman-Ford Algorithm

- Computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph
- Disadvantage is that it is slower than Dijkstra's algorithm
- Advantage is that it is more versatile
  - Capable of handling graphs in which some of the edge weights are negative numbers

Bellman-Ford Algorithm .

- Negative edge weights are found in various applications of graphs
  - Hence the usefulness of this algorithm
- If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path
  - Any path can be made cheaper by one more walk around the negative cycle

Bellman-Ford Algorithm ..

- It uses d[u] as an upper bound on the distance d[u, v] from u to v
- The algorithm progressively decreases an estimate d[v] on the weight of the shortest path from the source vertex s to each vertex v in V until it achieve the actual shortest-path
- Returns Boolean
  - TRUE if the given digraph contains no negative cycles that are reachable from source vertex s
  - Otherwise it returns FALSE

Bellman-Ford Algorithm ...

```
• INITIALIZE-SINGLE-SOURCE (G, s)
• 2.for each vertex i = 1 to V[G] - 1 do
• 3.    for each edge (u, v) in E[G] do
• 4.        RELAX (u, v, w)
• 5.For each edge (u, v) in E[G] do
• 6.    if d[u] + w(u, v) < d[v] then
• 7.        return FALSE
• 8.return TRUE
```

Johnson's Algorithm

- Johnson's algorithm is a way to find the shortest paths between all pairs of vertices in a sparse, edge weighted, directed graph
- It allows some of the edge weights to be negative numbers
  - But no negative-weight cycles may exist
- It works by
  - Using the Bellman-Ford algorithm to compute a transformation of the input graph that removes all negative weights
  - Allowing Dijkstra's algorithm to be used on the transformed graph

Greedy Best-First-Search Algorithm

- Greedy Best-First-Search algorithm works in a similar way to Dijkstra's, except that it has some estimate (called a heuristic) of how far from the goal any vertex is
- Instead of selecting the vertex closest to the starting point, it selects the vertex closest to the goal
- Greedy Best-First-Search is not guaranteed to find a shortest path
  - However, it runs much quicker than Dijkstra's algorithm

A* Algorithm

- A* is the most popular choice for pathfinding, because it's fairly flexible and can be used in a wide range of contexts
- A* is like Dijkstra's algorithm in that it can be used to find a shortest path
- A* is like Greedy Best-First-Search in that it can use a heuristic to guide itself
- In the simple case, it is as fast as Greedy Best-First-Search

A* Algorithm

- Combines the pieces of information that Dijkstra's algorithm uses and information that Greedy Best-First-Search uses
  - g(n) represents the exact cost of the path from the starting point to any vertex n
  - h(n) represents the heuristic estimated cost from vertex n to the goal
- A* balances the two as it moves from the starting point to the goal
- Each time through the main loop, it examines the vertex n that has the lowest $f(n) = g(n) + h(n)$

A* Algorithm with Python-Pygame

https://www.youtube.com/watch?v=FNRfSQDF7TA

Snake Game AI with A* algorithm

https://www.youtube.com/watch?v=DnyltgX2ACo

Crowds

Crowd Simulation

- Crowd simulation is the process of populating a virtual scene with a large number of intelligent agents that display distinct collective behaviours

A simulation with large crowds of intelligent agents

Crowds In Computer Games

- Crowds are very popular
- Games offer technically advanced visual and interactive simulation
- Not specific to a single genre
  - In Hitman crowds are utilised for blending into the game world to assassinate targets or hide from pursuers
  - In Grand Theft Auto, crowds are a living part of the city, simulated to act as normal pedestrians

China Town level of Hitman: Absolution

Grand Theft Auto 4

Crowds and Sociology

- Crowd simulation can also refer to simulations based on group dynamics and crowd psychology
  - Often in public safety planning
- In this case, the focus is just the behavior of the crowd, and not the visual realism of the simulation
  - Crowds have been studied as a scientific interest since the end of the 19th Century
- A lot of research has focused on the collective social behavior of people at:
  - Social gatherings, assemblies, protests, rebellions, concerts, sporting events and religious ceremonies

Social Simulation Approaches

- Two different approaches, with totally different results
- Macroscopic
  - i.e. fluid dynamics
- Microscopic
  - i.e. Multi-agent based simulation

Microscopic Simulation

- Also know as agent-based simulation
- Motion is computed separately for each individual member
- Some well known approaches:
  - Particle motion
  - Steering Behaviors For Autonomous Characters
  - Social force model
  - Crowd AI
  - Reciprocal Velocity Obstacles

Microscopic Simulation - Advantages

- Each individual character makes independent decisions
- Capture each character's unique situation: visibility, proximity of other pedestrians, and local factors
- Simulation parameters may be defined for each character -> yield complex heterogeneous motion

Microscopic Simulation - Drawbacks

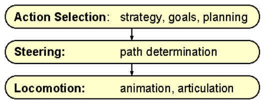
- Not easy to develop behavioral rules that consistently produce realistic motion
- Global path planning for each agent is computationally expensive
  - Particularly in simulating a large amount of agents
- Most agent models separate local collision avoidance from global path planning and conflicts inevitably arise between these two competing goals
- Local path planning often results in less realistic crowd behavior
- The problems tend to be exacerbated in areas of high congestion or rapidly changing environments

Particle Methods

- The characters are attached to point particles, which are then animated by simulating wind, gravity, attractions, and collisions
- The particle method is usually inexpensive to implement, and can be done in most 3D software packages
- The method is not very realistic because it is difficult to direct individual entities when necessary, and because motion is generally limited to a flat surface

Steering Behaviors For Autonomous Characters

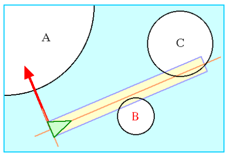
- In games, autonomous characters are sometimes called non-player characters
- The behavior of an autonomous character can be better understood by dividing it into several layers


Action Selection: strategy, goals, planning
Steering: path determination
Locomotion: animation, articulation

http://www.red3d.com/cwr/steer/gdc99/

Obstacle Avoidance

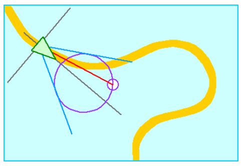
- Obstacle avoidance behavior gives a character the ability to maneuver in a cluttered environment by dodging around obstacles
- The implementation of obstacle avoidance behavior assumes that both the character and obstacle can be reasonably approximated as spheres
- Considers each obstacle in turn (perhaps using a spatial portioning scheme to cull out distance obstacles) and determines if they intersect with the cylinder



http://www.red3d.com/cwr/steer/gdc99/

Wander

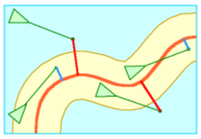
- Type of random steering
- Easy implementation generates a random steering force each frame
  - Produces rather uninteresting motion
- Better approach retains steering direction state
  - Make small random displacements to it each frame
- Another way is to use coherent Perlin noise to generate the steering direction



http://www.red3d.com/cwr/steer/gdc99/

Path Following

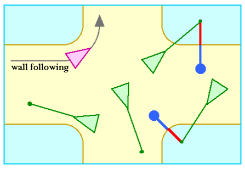
- Path following behavior enables a character to steer along a predetermined path
  - i.e. a roadway, corridor or tunnel
- The individual paths remain near, and often parallel to, the centerline of the corridor
  - But are free to deviate from it
- To compute steering for path following, a velocity-based prediction is made of the character's future position
- The predicted future position is projected onto the nearest point on the path spine



http://www.red3d.com/cwr/steer/gdc99/

Wall Following and Containment

- Variations on path following include 'wall following' and 'containment'
- Wall following means to approach a wall (or other surface or path) and then to maintain a certain offset from it
- Containment refers to motion which is restricted to remain within a certain region



http://www.red3d.com/cwr/steer/gdc99/

Social Forces

- A social force model is a microscopic, continuous time, continuous space, phenomenological computer simulation model of the movement of pedestrians
- Many models exist!

Helbing's Social Force Model

- Treats all agents as physical obstacles
- Solves a = F/m where F is "social force":

Desired Velocity Current Velocity

$$m_i \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = m_i \frac{v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t)}{\tau_i} + \sum_{j(\neq i)} \mathbf{f}_{ij} + \sum_{W} \mathbf{f}_{iW}$$

Avoiding Other Pedestrians Avoiding Walls

- Fij – Pedestrian Avoidance

$$\mathbf{f}_{ij} = \{A_i \exp[(r_{ij} - d_{ij})/B_i] + kg(r_{ij} - d_{ij})\}\mathbf{n}_{ij} + \kappa g(r_{ij} - d_{ij})\Delta v_{ji}^t \mathbf{t}_{ij}$$

- FiW – Obstacle (Wall) Avoidance

$$\mathbf{f}_{iW} = \{A_i \exp[(r_i - d_{iW})/B_i] + kg(r_i - d_{iW})\}\mathbf{n}_{iW} - \kappa g(r_i - d_{iW})(\mathbf{v}_i \cdot \mathbf{t}_{iW})\mathbf{t}_{iW}$$

Social Force Model – Pedestrian Avoidance

- rij – dij ← Edge-to-edge distance
- nij – Vector pointing away from agent
  - Ai*e[(rij-dij)/Bi] ← Repulsive force which is exponential increasing with distance
  - g(x) ← x if agents are colliding, 0 otherwise
- tij – Vector pointing tangential to agent
- Vtji – Tangential velocity difference
- FiW is very similar

$$\mathbf{f}_{ij} = \{A_i \exp[(r_{ij} - d_{ij})/B_i] + kg(r_{ij} - d_{ij})\}\mathbf{n}_{ij} + \kappa g(r_{ij} - d_{ij})\Delta v^t_{ji}\mathbf{t}_{ij}$$

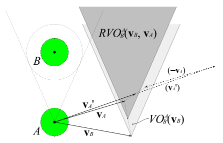
Collision   Non-penetration   Sliding Force

Crowd AI

- Agents are given artificial intelligence, which guides the entities based on one or more functions
  - i.e. sight, hearing, basic emotion, energy level, aggressiveness level, etc
- The entities are given goals and then interact with each other as members of a real crowd would
- They are often programmed to respond to changes in environment enabling them to climb hills, jump over holes, scale ladders, etc
- This is much more realistic than particle motion
  - But is very expensive to program and implement

Reciprocal Velocity Obstacles

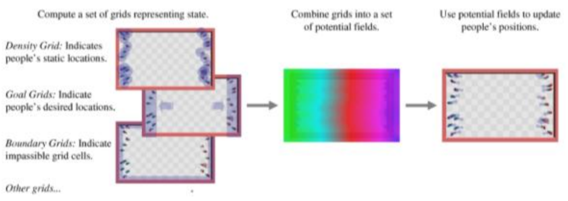
- Applied ideas from robotics to crowd simulations
- Basic idea:
  - Given n agents with velocities, find velocities will cause collisions
  - Avoid them!
- Planning is performed in velocity space



Macroscopic Simulation

- Simulate a large amount of crowd
- Treat motion as a per particle energy minimization
- Adopt a continuum perspective on the system
- This formulation yields a set of dynamic potential and velocity fields over the domain that guide all individual motion simultaneously

Methods


Compute a set of grids representing state.
Combine grids into a set of potential fields.
Use potential fields to update people's positions.
Density Grid: Indicates people's static locations.
Goal Grids: Indicate people's desired locations.
Boundary Grids: Indicate impassible grid cells.
Other grids...

Continuum Crowds, ACM Transactions on Graphics, Volume 25 Issue 3, July 2006, [Adrien Treuille et al.]

Examples

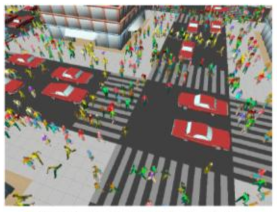


Figure 5: People in yellow shirts evacuate a building.

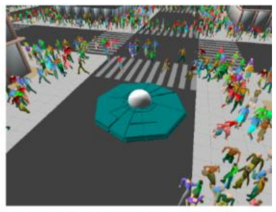


Figure 6: A continuum crowd reacts to a user-driven flying saucer agent in real-time.

Continuum Crowds, ACM Transactions on Graphics, Volume 25 Issue 3, July 2006, [Adrien Treuille et al.]

Interactive Manipulation of Large-Scale Crowd Animation

https://www.youtube.com/watch?v=DYLfFa6CNRU

Simulating Dynamic Features of Escape Panic Case Study

Aim

- A model of pedestrian behaviour to investigate the mechanisms of panic and jamming by uncoordinated motion in crowds

http://angel.elte.hu/panic/

Features of Escape Panic

- People move or try to move considerable faster than normal
- Individuals start to pushing, and interactions become physical
- Moving becomes uncoordinated
- At exist, arching and clogging are observed
- Jams build up
- Pressure on walls and steel barriers increase
- Escape is further slowed by fallen or injured people acting as ‘obstacles’

http://angel.elte.hu/panic/

The Problem & Solution

Crowd stampedes can be deadly
People act in uncoordinated and dangerous ways when panicking
It is difficult to obtain real data on crowd panics

Model people as self-driven particles
Model physical and socio-psychological influences on people’s movement as forces
Simulate crowd panics and see what happens

http://angel.elte.hu/panic/

Acceleration of Simulated People

- $v_i^0(t)$ = desired speed
- $e_i^0(t)$ = desired direction
- $v_i(t)$ = actual velocity
- τ_i = characteristic time
- m_i = mass

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i \frac{v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t)}{\tau_i} + \sum_{j(\neq i)} \mathbf{f}_{ij} + \sum_{W} \mathbf{f}_{iW}$$

http://angel.elte.hu/panic/

Forces from Other People

- Force from other people's bodies being in the way
- Force of friction preventing people from sliding
- Psychological "force" of tendency to avoid each other
- Sum of forces of person j on person i is f_{ij}

http://angel.elte.hu/panic/

Total Force of Other People

- $A_i\exp[(r_{ij} – d_{ij})/B_i]n_{ij}$ is psychological "force"
- A_i and B_i are constants

sum of the people's radii

distance between people`s centers of mass

$$\mathbf{f}_{ij} = \{\underbrace{A_i \exp[(r_{ij} - d_{ij})/B_i]}_{\text{psychological force}} + kg(r_{ij} - d_{ij})\}\mathbf{n}_{ij} + \kappa g(r_{ij} - d_{ij})\Delta v^t_{ji}\mathbf{t}_{ij}$$

normalized vector from j to i

http://angel.elte.hu/panic/

Physical Forces

- g(x) is 0 if the people don't touch and x if they do touch
- k and κ are constants

tangential velocity difference

tangential direction

$$\mathbf{f}_{ij} = \{A_i \exp[(r_{ij} - d_{ij})/B_i] + \underbrace{kg(r_{ij} - d_{ij})\}\mathbf{n}_{ij}}_{\text{force from other bodies}} + \underbrace{\kappa g(r_{ij} - d_{ij})\Delta v^t_{ji}\mathbf{t}_{ij}}_{\text{force of sliding friction}}$$

http://angel.elte.hu/panic/

Forces from Walls

- Forces from walls are calculated in basically the same way as forces from other people

$$\mathbf{f}_{iW} = \{A_i \exp[(r_i - d_{iW})/B_i] + kg(r_i - d_{iW})\}\mathbf{n}_{iW} - \kappa g(r_i - d_{iW})(\mathbf{v}_i \cdot \mathbf{t}_{iW})\mathbf{t}_{iW}$$

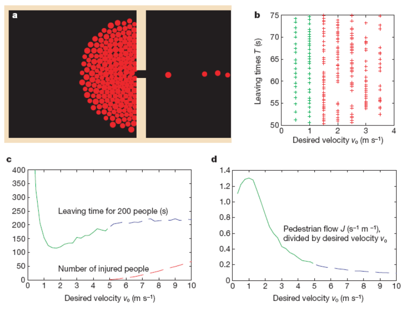
http://angel.elte.hu/panic/

Values Used for Constants and Parameters

- Insufficient data on actual panic situations to analyze the algorithm quantitatively
- Values chosen to match flows of people through an opening under non-panic conditions

http://angel.elte.hu/panic/

Simulation of Clogging



http://angel.elte.hu/panic/

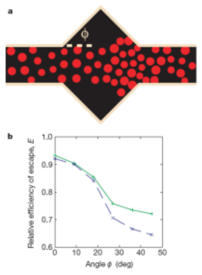
Simulation of Clogging .

- As desired speed increases beyond 1.5m s^{-1}, it takes more time for people to leave
- As desired speed increases, the outflow of people becomes irregular
- Arch shaped clogging occurs around the doorway

http://angel.elte.hu/panic/

Widening Can Create Crowding

- The danger can be minimized by avoiding bottlenecks in the construction of buildings
- However, that jamming can also occur at widenings of escape routes



http://angel.elte.hu/panic/

Mass Behavior

- Panicking people tend to exhibit either herding behavior or individual behavior, or try to mixture of both
- Herding simulated using "panic parameter" p_i

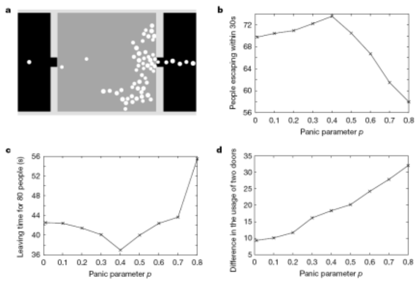
$$\mathbf{e}_i^0(t) = \text{Norm}[(1-p_i)\mathbf{e}_i + p_i\langle\mathbf{e}_j^0(t)\rangle_i]$$

Individual direction

Average direction of neighbors

http://angel.elte.hu/panic/

Effects of Herding



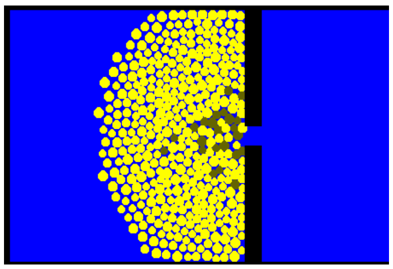
http://angel.elte.hu/panic/

Effects of Herding .

- Neither individuals nor herding behaviors performs well
- Pure individualistic behavior:
  - Each pedestrian finds an exit only accidentally
- Pure herding behavior:
  - Entire crowd will eventually move into the same and probably blocked direction

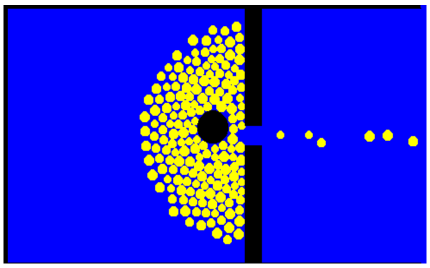
http://angel.elte.hu/panic/

Injured People Block Exit



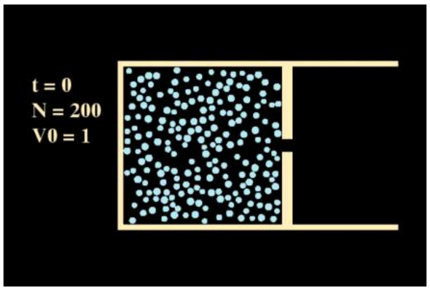
http://angel.elte.hu/panic/

A Column Can Increase Outflow



http://angel.elte.hu/panic/

Scalable Crowd Simulation Case Study



http://angel.elte.hu/panic/

Scalable Crowd Simulation

- Large Crowds
  - Scalable performance
- Large Complex Environments
  - Scalable Authoring
- Rich, Complex Behaviors
  - Scalable Behaviors

http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Conflicting Goals

- Large, Complex World
- Rich Behaviors
- But…
- Fast performance (simple agents)
- Reasonable authoring

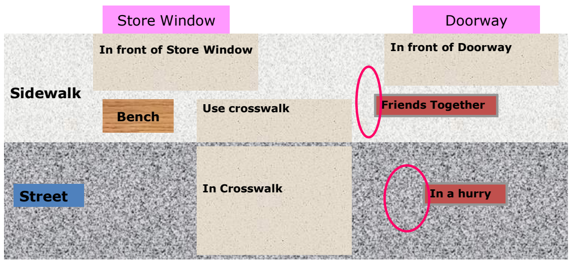
http://research.cs.wisc.edu/graphics/Gallery/Crowds/

An Environment

- Example:
  - Model of Street
- Simulation:
  - Real time, Reactive
- Rendering:
  - Unreal Game Engine for playback

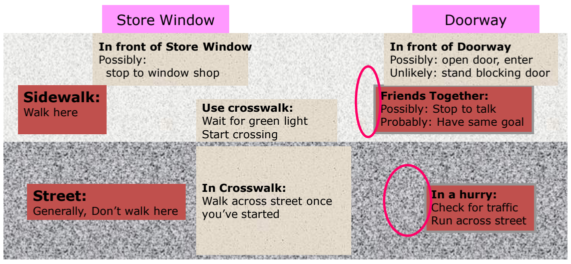
http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Scalability: Complex Environments



http://research.cs.wisc.edu/graphics/Gallery/Crowds/

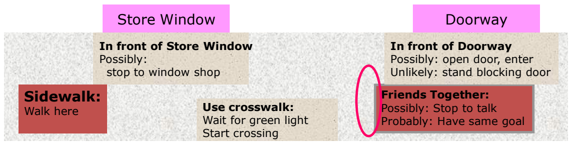
Observation: Behavior Depends on Situation



http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Managing Environmental Complexity: Situation-Based Approach

- Many different situations
- Each has a different set of local behaviors
- An agent only needs a few at a time
- Blend situations/behaviors together



http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Observation: Crowds are Crowds

- Individuals are anonymous
  - Doesn't matter what any one does
  - At any given time, do something reasonable
  - Aggregate behavior
- Stochastic Control
- Short term view of agent

An Individual

http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Key Ideas

- Situation-Based Approach
  - Breaks behavior into small pieces
  - Extensible agents kept simple
- Situation Composition
  - Probabilistic scheme to compose behaviors
- Painting Interface
  - Place behaviors in world, not agents
- Use Motion-Graph-based runtime
  - Based on Gleicher et al 2003

http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Situation-Based Approach: Agent Architecture

- Agents:
  - Discrete set of actions
  - Randomly choose from distribution
  - Behavior functions provide distributions
- All aspects of agents can be updated dynamically

http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Situation-Based Approach: Simple Default Agents

- Default agents very simple
  - Wander, don't bump into things, …
- Extend agents as necessary to achieve complex behaviors

http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Situation-Based Approach: Extensible Agent

- Situations extend agents
  - Add Actions
  - Add Behavior Functions
  - Add Sensors and Rules that inform Behavior Functions

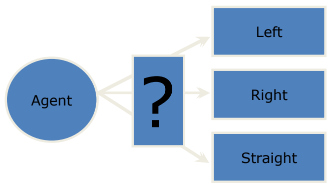
http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Simple Example

- Default agent can't cross the street
- How an agent crosses the street…
  - Enters a Crosswalk Situation
  - Crosswalk situation extends agent
    - Sensor to see traffic light
    - Behavior Functions to cross street
    - Behavior Functions to stop
    - Rules to wait for light to change
  - Remove extensions when done

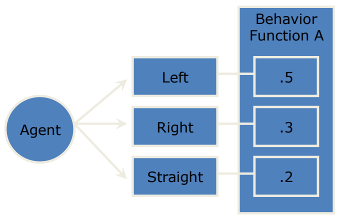
http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Composing Behaviors: Action Selection



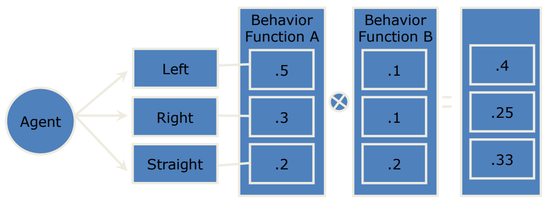
http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Composing Behaviors: Probability Scheme



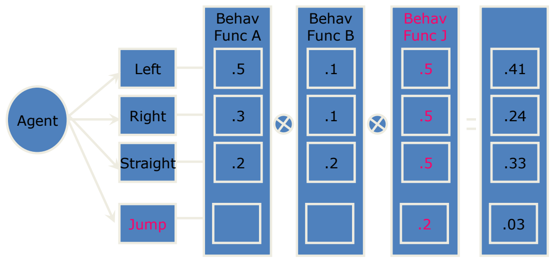
http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Composing Behaviors: Probability Scheme



http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Composing Behaviors: Extending Agents



http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Video

http://research.cs.wisc.edu/graphics/Gallery/Crowds/

Character Modeling

3D Animation

- Rendering
  - 3D Scene and Motion
  - Sequence of Frames
    - Rates: Video 30fps, Film 24fps
  - Persistence of Vision
- Animator must create
  - Illusion of Life
  - Weight

Animation

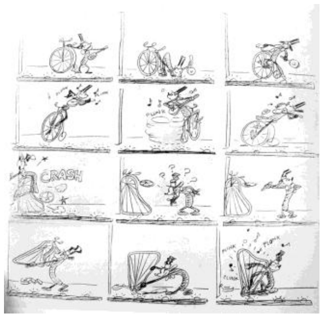
- Almost every property of every object in the scene can be animated changed through time
  - Models, cameras
  - Transformations (Move, Rotate, Scale)
- Modifications/Deformation:
  - Edits, bends, twists, manipulating a skeleton
- Materials, colors, textures

Animation .

- 3D Scene does not have
  - Gravity
  - Weight
  - Force
  - Complete interactions between objects
    - Sometimes it has…
- You must make it seem so

Preproduction Phases

- Screen-play
- Storyboards
- Character development



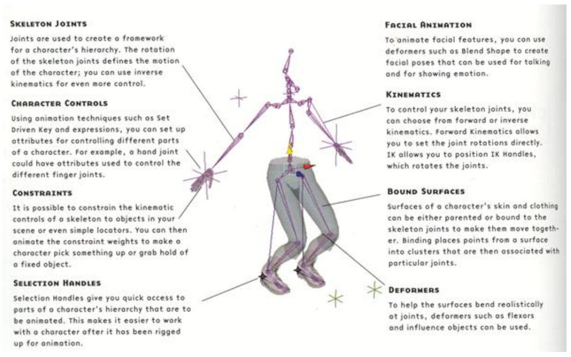
3D Characters

- Digital actor
  - Tin can
  - Sack of flower
  - Butterfly, beetle
  - Bird
  - Flower
  - Robot
  - Humanoid
  - Etc…

Typical Character

- Mechanics of movement must be convincing
- Skin and clothing moves & bends appropriately
- This process of preparing character controls is called rigging (see next slides)
  - Fully rigged character has:
    - Skeleton joints, surfaces, deformers, expressions, Set Driven Key, constraints, etc

Typical Character .



Skeleton Joints

Joints are used to create a framework for a character's hierarchy. The rotation of the skeleton joints defines the motion of the character; you can use inverse kinematics for even more control.

Character Controls

Using animation techniques such as Set Driven Key and expressions, you can set up attributes for controlling different parts of a character. For example, a hand joint could have attributes used to control the different finger joints.

Constraints

It is possible to constrain the kinematic controls of a skeleton to objects in your scene or even simple locators. You can then animate the constraint weights to make a character pick something up or grab hold of a fixed object.

Selection Handles

Selection Handles give you quick access to parts of a character's hierarchy that are to be animated. This makes it easier to work with a character after it has been rigged up for animation.

Facial Animation

To animate facial features, you can use deformers such as Blend Shape to create facial poses that can be used for talking and for showing emotion.

Kinematics

To control your skeleton joints, you can choose from forward or inverse kinematics. Forward Kinematics allows you to set the joint rotations directly. IK allows you to position IK Handles, which rotates the joints.

Bound Surfaces

Surfaces of a character's skin and clothing can be either parented or bound to the skeleton joints to make them move together. Binding places points from a surface into clusters that are then associated with particular joints.

Deformers

To help the surfaces bend realistically at joints, deformers such as flexors and influence objects can be used.

Facial Animation Video

https://www.youtube.com/watch?v=z86YsS-pVsQ

Character Resolution

- Use low resolution character that has surfaces 'parented' to skeleton
  - Allows interactive animations
  - Switch to full resolution character later

Typical Character Animation Workflow

- Character Design
- Model
- Skeleton Rigging
- Binding
- Animation
- Integration
- Rendering

Rigging

- Rigging refers to the construction and setup of an animatable character
  - Similar to the idea of building a puppet
- A 'rig' has numerous degrees of freedom (DOFs) that can be used to control various properties

Primary Methods of Animation

- Keyframe
- Procedural
  - Expressions
  - Scripting
- Dynamics/Simulation
  - Physics
- Motion Capture
- Combinations of the above

Keyframe Workflow

- Set Keys
  - Usually extreme positions
  - Less is more: Keys only the properties being animated
- Set Interpolation
  - Specify how to get from one key to another
  - Secondary, but a necessary step
- Scrub Time slider and refine motion curve

Setting Keys

- Start with extreme positions
- Add intermediate positions
  - Secondary motion
- Less is more
  - Don't add keys for properties that you are not animating
  - Easier to manage/edit fewer keys

Skeletal Hierarchy

- The Skeleton is a tree of bones
  - Often flattened to an array in practice
- Top bone in tree is the "root bone"
  - May have multiple trees, so multiple roots
- Each bone has a transform
  - Stored relative to its parent's transform
- Transforms are animated over time
- Tree structure is often called a "rig"

Bone Masks

- Some animations only affect some bones
  - Wave animation only affects arm
  - Walk affects legs strongly, arms weakly
    - Arms swing unless waving or holding something
- Bone mask stores weight for each bone
  - Multiplied by animation's overall weight
  - Each bone has a different effective weight
  - Each bone must be blended separately
- Bone weights are usually static
  - Overall weight changes as character changes animations

Variable Delta Extraction

- Uses root bone motion directly
- Sample root bone motion each frame
- Find delta from last frame
- Apply to instance pos+orn
  - Instance pos+orn is the root bone!
- Root bone is ignored when rendering

Animation Storage Problem

- 4x3 matrices, 60 per second is huge
  - 200 bone character = 0.5Mb/sec
- Consoles have around 32-64Mb
- Animation system gets maybe 25%
- PC has more memory
  - But also higher quality requirements

Decomposition

- Decompose 4x3 into components
  - Translation (3 values)
  - Rotation (4 values - quaternion)
  - Scale (3 values)
  - Skew (3 values)
- Most bones never scale & shear
- Many only have constant translation
- Don’t store constant values every frame

Interpolation

- Specify how to get from one key to the other (in between)
- Common types
  - Step: stay at the same value, then suddenly switch
  - Linear: change at constant rate
  - Spline/Smooth: make it smooth
- All of these (and more) are useful and appropriate in the right circumstance

Mesh Deformation

- Find Bones in World Space
- Find Delta from Rest Pose
- Deform Vertex Positions
- Deform Vertex Normals

Find Bones in World Space

- Animation generates a “local pose”
  - Hierarchy of bones
  - Each relative to immediate parent
- Start at root
- Transform each bone by parent bone’s world-space transform
- Descend tree recursively
- Now all bones have transforms in world space
  - “World pose”

Find Delta from Rest Pose

- Mesh is created in a pose
  - Called the "rest pose"
- Must un-transform by that pose first
- Then transform by new pose
  - Multiply new pose transforms by inverse of rest pose transforms
  - Inverse of rest pose calculated at mesh load time
- Gives "delta" transform for each bone

Deform Vertex Positions

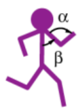
- Deformation usually performed on GPU
- Delta transforms fed to GPU
  - Usually stored in "constant" space
- Vertices each have n bones
- n is usually 4
  - 4 bone indices
  - 4 bone weights 0-1
  - Weights must sum to 1

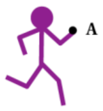
Deform Vertex Normals

- Normals are done similarly to positions but use inverse transpose of delta transforms
  - Translations are ignored
  - For pure rotations, inverse(A)=transpose(A)
  - So inverse(transpose(A)) = A
  - For scale or shear, they are different
- Normals can use fewer bones per vertex
  - Just one or two is common

Motion of characters

- Along with key frame animation we can use kinematics
  - Kinematics = study of motion without regard to the forces that cause it





Forward: $A = f(\alpha,\beta)$ **Inverse:** $\alpha,\beta = f^{-1}(A)$

Specify fewer degrees of freedom

More intuitive control

FK & IK

- Most animation is "forward kinematics"
  - Motion moves down skeletal hierarchy
- But there are feedback mechanisms
  - Eyes track a fixed object while body moves
  - Foot stays still on ground while walking
  - Hand picks up cup from table
- This is "inverse kinematics"
  - Motion moves back up skeletal hierarchy

Skeleton Puppet Game Video (FK & IK)

https://www.youtube.com/watch?v=QDwo9d8Fa5M

User Control of Kinematic Characters

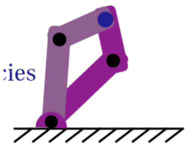
- Joint Space
  - Position all joints
    - Fine level of control
- Cartesian Space
  - Specify environmental interactions easily
    - Most DOF computed automatically

Inverse Kinematics

- Balance = keep center-of-mass over support polygon
- Control
  - i.e. position vaulter's hands on line between shoulder and vault
  - i.e. Compute knee angles that will give runner the right leg length

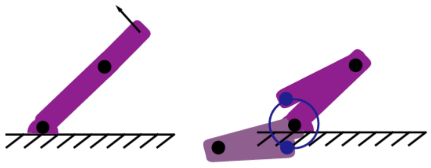
Inverse Kinematics is Hard

- Redundancy



Inverse Kinematics is Hard .

- Singularities



Momentum-based Inverse Kinematics with Motion Capture Video

https://www.youtube.com/watch?v=FJTBMnP6oCM

Single Bone IK

- Orient a bone in given direction
  - Eyeballs
  - Cameras
- Find desired aim vector
- Find current aim vector
- Find rotation from one to the other
  - Cross-product gives axis
  - Dot-product gives angle
- Transform object by that rotation

Multi-Bone IK

- One bone must get to a target position
  - Bone is called the "end effector"
- Can move some or all of its parents
- May be told which it should move first
  - Move elbow before moving shoulders
- May be given joint constraints
  - Cannot bend elbow backwards

Cyclic Coordinate Descent

- Simple type of multi-bone IK
- Iterative
  - Can be slow
- May not find best solution
  - May not find any solution in complex cases
- But it is simple and versatile
  - No pre-calculation or pre-processing needed

Cyclic Coordinate Descent .

- Start at end effector
- Go up skeleton to next joint
- Move (usually rotate) joint to minimize distance between end effector and target
- Continue up skeleton one joint at a time
- If at root bone, start at end effector again
- Stop when end effector is "close enough"
- Or hit iteration count limit

Cyclic Coordinate Descent ..

- May take a lot of iterations
- Especially when joints are nearly straight and solution needs them bent
  - e.g. a walking leg bending to go up a step
  - 50 iterations is not uncommon!
- May not find the "right" answer
  - Knee can try to bend in strange directions

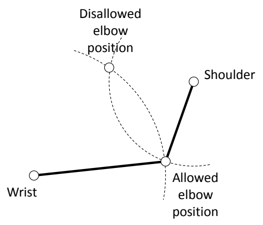
Two-Bone IK

- Direct method, not iterative
- Always finds correct solution
  - If one exists
- Allows simple constraints
  - Knees, elbows
- Restricted to two rigid bones with a rotation joint between them
  - Knees, elbows!
- Can be used in a cyclic coordinate descent

Two-Bone IK .

- Three joints must stay in user-specified plane
  - e.g. knee may not move sideways
- Reduces 3D problem to a 2D one
- Both bones must remain same length
- Therefore, middle joint is at intersection of two circles
- Pick nearest solution to current pose
- Or one solution is disallowed
  - Knees or elbows cannot bend backwards

Two-Bone IK ..



IK by Interpolation

- Animator supplies multiple poses
- Each pose has a reference direction
  - e.g. direction of aim of gun
- Game has a direction to aim in
- Blend poses together to achieve it
- Source poses can be realistic
  - As long as interpolation makes sense
  - Result looks far better than algorithmic IK with simple joint limits

IK by Interpolation .

- Result aim point is inexact
  - Blending two poses on complex skeletons does not give linear blend result
- Can iterate towards correct aim
- Can tweak aim with algorithmic IK
  - But then need to fix up hands, eyes, head
  - Can get rifle moving through body

Attachments

- For example character holding a gun
- Gun is a separate mesh
- Attachment is bone in character's skeleton
  - Represents root bone of gun
- Animate character
- Transform attachment bone to world space
  - Move gun mesh to that pos+orn

Attachments .

- For example person is hanging off bridge
- Attachment point is a bone in hand
  - As with the gun example
- But here the person moves, not the bridge
- Find delta from root bone to attachment bone
- Find world transform of grip point on bridge
- Multiply by inverse of delta
  - Finds position of root to keep hand gripping

Collision Detection

- Most games just use bounding volume
- Some need perfect triangle collision
  - Slow to test every triangle every frame
- Pre-calculate bounding box of each bone
  - Transform by world pose transform
  - Finds world-space bounding box
- Test to see if bounding box was hit
  - If it did, test the this bone influences

NCCA Video 1

NCCA Video 2

Case Study For Crowd Modeling

Aim

- Aim
  - Feedback into improving the development processes of simulations and video games, implementing virtual crowds, especially those within urban environments
- Research Question
  - What Features of Behaviour Positively Influence the Perceived Realism of Agents Within a Virtual Urban Environment, and to What Degree?

Complexity Fallacy

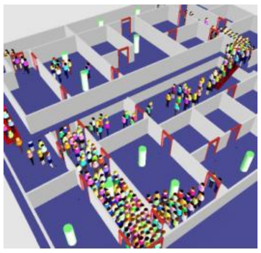
- Over complex AI's do not necessarily produce better behaviour
- There have been cases where over complex AI's have not produced good results whereas simple techniques have
- To quantify the effectiveness of crowd behaviour within a simulation, a method other than the judging the complexity is required

Types of Realism

- Realism is an important aspect in the majority of crowd simulations, however the type of realism required depends upon a simulations purpose
- Several definitions of realism within the context of crowd simulation relating to specific aspects
  - i.e. Graphical fidelity
- Two important types
  - Virtual Realism
  - Perceived Realism

Virtual Realism

- Virtual Realism is how close a simulation or specific feature is to reality
- This type of realism is important for serious simulations
  - Evacuation procedures
  - Urban planning


The HiDAC system showing high density crowds

Perceived Realism

- Perceived Realism is how realistic a simulation or specific feature is perceived to be by human viewers
- Important for entertainment based simulations
  - i.e. Video games
- Perceptual Experiments can be utilised to gauge the Perceived Realism of a simulation or feature
- This is the primary realism type investigated in this research

Virtual crowds in the Assassin's Creed 3 video game published by Ubisoft

Methodology

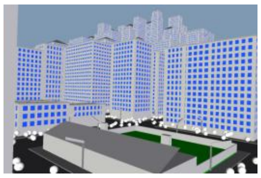
- Analysis: Identify a feature and inform algorithm construction, by analysing real-world and similar instances of crowd behaviour
- Synthesis: Synthesise a new generation simulation with further refinement and the behaviour impacting feature that was identified in analysis
- Perception: Conduct the psychophysical experiment for gauging the perceived realism values of the implemented feature

Urban Crowd Simulation

- The primary purpose for developing the urban crowd simulation was to create a platform with alterable parameters capable of customising agent behaviour for the purposes of perceptual evaluation
- Due to the perceptual aspect, the standard modelling and behaviour approaches had to be altered to accommodate the fact that more stimuli were needed than just the configuration that appeared most realistic to the developer

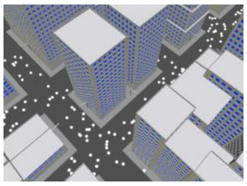
Procedural City Generation

- A procedural approach was selected to generate the virtual urban city
  - Allows for the possibility of multiple layouts and setups
  - Produces complex geometry without having to manually model and place object


A procedurally generated city in the urban crowd simulation

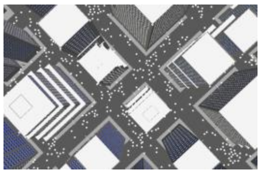
Core AI Components

- Four core components implemented for the agents in the urban crowd simulation:
  - Decision making utilising finite-state machines
  - Pathfinding with the A* algorithm
  - Steering with the crowd path-following mechanic
  - Perception utilising radial local neighbourhoods


The urban crowd simulation displaying crowds of agents

Social Forces Model

- The realistic motion of pedestrians would be subject to social forces
- An additional social forces component has been implemented for the altering the behaviour of agents
- The model consists several distinct forces that when presented together help to produce realistic behaviour with regards to pedestrians


A scene from the urban crowd simulation with initial social forces implemented

Social Forces Model

A) Repulsive force between the individual pedestrian and other pedestrians that are close by

B) Repulsive force between the individual pedestrian and obstructions such as buildings, close by

C) Attractive force between the individual pedestrian and other pedestrians nearby

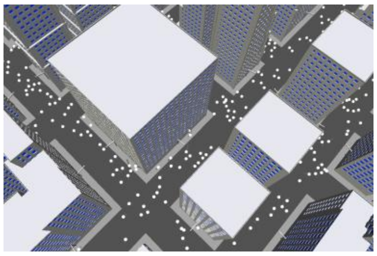
- However this has an element of randomness

D) Attractive forces between the individual pedestrian and objects, such as a store window or parked car

Quantitative Evaluation

- Along with the core components, other behaviour orientated features are identified in analysis, implemented in synthesis and then perceptually tested in perception
- These features require parameter space and customisability for perceptual evaluation
- The agent velocity feature has been fully implemented with parameter space for minimum and maximum velocities, along with velocity distribution
- An annotation mechanism has been implemented for future features based on environmental factors

Urban Crowd Simulation Demo

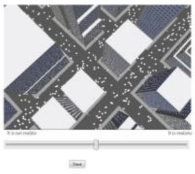


Perceived Realism Experiments

- Three psychophysical experiments planned to gauge perceived realism and identify successful feature thresholds and parameter values
- Each experiment involves participants viewing video clips of different simulation set-ups and judging the realism
- Data is collected in the form of simulation configuration ID's with linked perceived realism values
- A prototype survey platform has been used for a pilot study

Prototype Survey Platform

- Platform developed using PHP to act as a survey platform for the psychophysical experiments
  - Provides a slider for participants to choose the realism level of a given clip
  - Orders the videos with regards to the psychophysical methodology employed
  - As the platform was developed using a language for server side applications it can be extended online to reach large numbers of participants


The survey platform prototype displaying a video clip

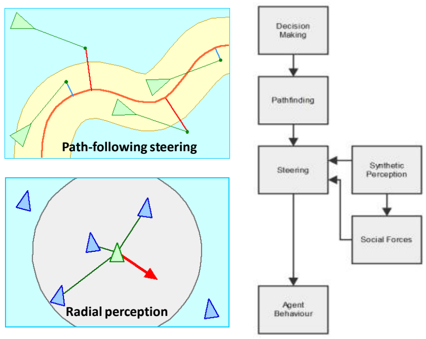
Psychophysical Methodologies

- The type and order of video clips shown to participants is dependant upon the psychophysical methodology and the experiment
- The first and primary experiment utilises the adaptive staircase procedure to establish the perceptual thresholds and perceived realism levels of a feature
- The second experiment utilises the adjustment procedure to rank features based on there perceptual effectiveness
- The third experiment utilises the constant stimuli procedure to determine the threshold and most effective number of features required for perceptual plausibility

Pilot Study

- A pilot study was conducted with three participants utilising the primary staircase methodology
- The feature perceptually tested was agent velocity and the participants were shown different configurations of velocity range and distribution
- Final results gave a velocity range of 0.3, based on a perceived realism value of 0.82, with thresholds of 0.2 and 0.5
- Velocity distribution was 0.5, based on an average perceived realism value of 0.85, with thresholds of 0.3 and 0.7

More Behavioural Features



Social Forces Scenario

- The resultant steering force is calculated from the three forces present in the algorithm
  - SF = (ar * w) + (aa * w) + (or * w)
- where
  - "SF" is the resultant calculated social force, which acts as a steering modifier
  - Agent repulsion is represented as "ar“
  - agent attraction as "aa“
  - object repulsion as "or“
  - Parameter weight is represented as the modifier "w"

Social Forces Scenario .

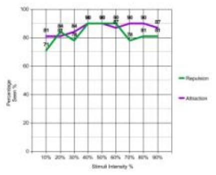
- Calculated individually these forces are:
  - ar = (a - a^n) * n, (t, r)
  - aa = (a^n - a) * n, (t, r)
  - or = (a - o^n) * n, (t, r)
- where
  - "a^n" is the position of all the agents within vision
  - "o^n" is the position of all the objects within vision
  - "a" is the current agents position
  - "n" is the normalizing factor
  - "t" is the time factor
  - "f" is the behaviour fluctuation factor

Experiment

- 32 participants
- The experiment consists of two key variables
  - One for each of the agent based social forces.
- These variables are tested at specific trials:
  - Trials 01 to 09 for agent avoidance
  - Trials 10 to 18 for agent attraction
    - This accounts to a total of 18 trials

Results

- 94% of participants found that when the agent avoidance social force is present the behaviour of the agents is more realistic
- 95% selected the videos with the agent attraction social force present to be more realistic
- 95% of the participants find a simulation with social forces to be more realistic



Crowd Grouping Experiment

- A platform for testing certain aspects of artificial intelligence (AI) using psychophysical methodologies
- The current research is looking into aspects relating to crowd based AI, most specifically for pedestrians

http://psychophysicsforai.weebly.com/

Video Example

References

- http://www.cs.cmu.edu/~jkh/
- https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html
- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/bellFordAlgor.htm
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- http://www.red3d.com/cwr/steer/
- http://research.cs.wisc.edu/graphics/Gallery/Crowds/
- http://www.fi.muni.cz/~liarokap/publications/VSGAMES2013b.pdf

Bibliography

- Social Force Model for Pedestrian Dynamics, 1998, [Dirk Helbing and Peter Molnar]
- Real-time Navigation of Independent Agents Using Adaptive Roadmaps, VRST, 2007 [Avneesh Sud et al.]
- Real-Time Multi-Agent Path Planning on Arbitrary Surfaces, Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, 2010 [Rafael P. Torchelsen et al.]
- Continuum Crowds, ACM Transactions on Graphics, Volume 25 Issue 3, July 2006, [Adrien Treuille et al.]
- Space-time Group Motion Planning, Springer Tracts in Advanced Robotics Volume 86, 2013, [Ioannis Karamouzas et al.]

Questions