May 2010

Real time crowd simulations incorporating individual agent personality models and group interactions.

*A Dissertation by **Scott Bevin** submitted in partial fulfilment of the requirements of **The University of Bolton** for the degree of Bachelor of Science.*

School of Business and Creative Technologies

The University of Bolton,

Email: scott@gameXcore.co.uk

1. Abstract

This paper presents an approach to integrating per-agent based personality and intelligence into a crowd simulation, in order to create a more varied crowd. The paper first meets the basic requirements for a crowd simulation, focusing on medium density crowds, path planning, locomotion and local area obstacle avoidance algorithms are first implemented. The paper then moves on to investigate and implement ways in which agents within the simulation can be given personality and intelligence. The experiments performed on the simulation proved that it is possible to vary agents using variables mapped to OCEAN variables to create a more dynamic simulation.

2. Keywords

Crowd Simulation, autonomous agents, human behaviour models, artificial personality, OCEAN

Table of Contents

3. Introduction

Over the course of the past few decades there have been many advances in fields such as AI technique, large agent-based systems and cogitative modelling, these advances have made the modelling of large crowds of virtual, autonomous agents feasible for off-line animations and films. More recently there has been growing interest regarding the development of these crowd simulations for application in real-time video games (Reynolds, 1999) and other virtual environments (Thalmann et al., 2006). Multi-agents systems are also applicable to other areas such as the study of human and social behaviours with regard to architectural design and training for situations involving emergency evacuation.

3.1. Project background

As advances are made, games developers are starting to focus on increasingly immersive, realistic and large scale environments. However, in order to succeed, developers must first draw comparisons and contrasts between the real world and the in-game environments being created. Analysing our environments in this way will aid us in a more towards more immersive game play.

An area of great interest when creating these immersive environments is population. Crowds of people are ubiquitous in real life. Everywhere you look, you will find groups congregating, regardless of whether they are small, large, sparse or dense, within these crowds the people all interact with each other in intricate ways and where able show individuality.

Staying within the scope of this dissertation project, crowds in games more often than not focus on size and the basic constraint that agents should move around without walking into things. The end result is that the crowd often represents that of a crowd of zombies, which risks lowering the player's immersion within the environment.

3.2. Aims

The aim of this dissertation is to add to the immersion of the player within a crowd situation by giving agents within the crowd individual personalities, as well as creating agents that interact in a realistic manner.

The project is broken down into 3 areas;

1. Agents need to navigate around the world, while intelligently avoiding obstacles and other agents.
2. Agents should express individuality.
3. Agents should interact with each other in a realistic manner.

3.3. Report structure

From this point on the report will be structured as follows. First is a review of related work and literature that was researched prior to the development of the artefact. The design and implementation of the crowd simulation software is then discussed, following this discussion the closing sections of the report will talk about the testing methods used on the software and provide critical analysis and evaluation of any results gained. Finally recommendations and suggestions are made for ways in which this work could be furthered.

4. Related Work

The following section reviews background research conducted prior to the development of the project. The research was split into two sections:

- How can agents navigate safely around the world?
- What options are available for giving agents the illusion of personality?

4.1. Movement and local navigation

Many efficient algorithms have been developed for the purpose of navigating agents around virtual environments (Lamarch & Donikian, 2004) (Sud et al., 2007) (Pettre, Laumond, & Thalmann, 2005) each succesfully performing this complex task at interactive speeds. Navigation of large groups of agents around each other in medium-density scenerios is however still a problem in its infancy with the proposed algorithms all demonstrating some degree of problematic or unrealistic and unwanted behaviour.

At this point an explanation as to the differences between decentralized multi-agent *local navigation* and centralized multi-agent *planning* should be made. Multi-agent local navigation seeks to avoid collision with other agents on a per-time-step basis, the algorithm analyses the current state of the world around the agent and then chooses the correct way for the agent to move. In multi agent *planning* all agents within the space are considered and a path for each is planned centrally, these focus on a different problem, namely that of optimal coordination between agents and are therefore mostly not suitable for real-time application.

4.1.1. Velocity Obstacles

Velocity obstacles (Fiorini & Shiller, 1988) were introduced as an algorithm to allow an agent to safely navigate around other [passively] moving obstacles, such as other agents, independently and without the need for explicit communication.

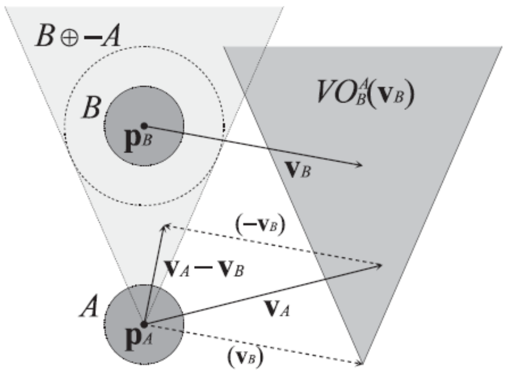


Fig. 1. The velocity obstacle of obstacle B to agent A.

During each time step of the simulation, the agent constructs the set of velocities which would result in a collision with another agent; the velocities within the constructed set represent the velocity obstacle. Once a velocity obstacle is constructed for each surrounding obstacle a set can be created which combines all of the velocity obstacles and a velocity outside of this collective set can be chosen, this velocity is guarunteed to be collision free.

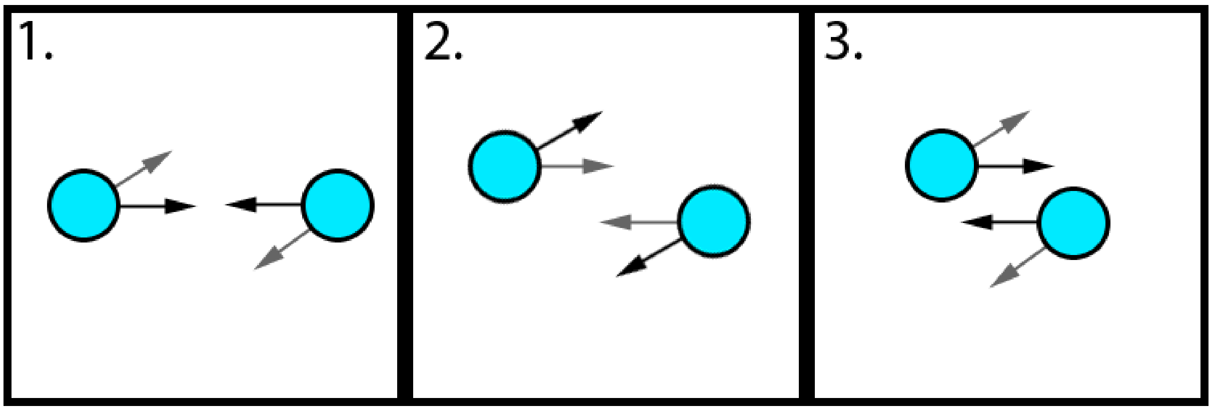


Fig. 2. A Diagram showing how two agents moving towards each other cause oscillation.

While velocity obstacles can successfully be used to navigate agents around each other, a major problem arises in the form of oscillation. Oscillation occurs when two agents need to travel directly through each other to get to their goal: Consider two agents *A* and *B*, moving along their respective velocities v_A and v_B. Assume that $v_A \in VO^A_B(v_B)$ and $v_B \in VO^B_A(v_A)$. If the agents continue along their current trajectories this would result in a collision, thus, agent *A* alters its velocity to v'_A such that $v'_A \notin VO^A_B(v_B)$ and agent *B* alters its velocity to v'_B such that $v_B \notin VO^B_A(v_A)$. If both agents were to prefer their old velocity's for some reason such as it leading directly to their goal they will then switch back, this series of events will likely continue, causing an oscillation (see **Fig. 2.**).

Oscillation is caused by the velocity obstacle algorithm not taking into consideration how the other agent will react, there have been many attempts to address this problem; Abe & Matsuo attempted to solve this problem using Common Velocity Obstacles (2001). Common Velocity Obstacles attempted to define all the combinations of velocities that could exist between two agents in 4-dimentional space and then select the best. While Common Velocity Obstacles addressed the oscillation problem, they failed to scale well with large numbers of agents. Recursive Velocity Obstacles (Kluge & Prassler, 2007) were later proposed. Using recursive velocity obstacles agent *A* would decide upon a velocity based upon the expected behaviour of agent *B*, who, in turn, would generate a velocity based upon the expected behaviour of agent *A*, and so on until some level of recursion has been reached. Recursive velocity obstacles failed to solve the problem properly.

4.1.2. Reciprocal Velocity Obstacles

Reciprocal Velocity Obstacles (Berg, Ming, & Manocha, 2008) introduced a simple and elegant approach to solve the oscillation problem. Instead of choosing a velocity that is outside the velocity obstacle, a velocity is chosen such that it is the average of the agents' current velocity and a velocity outside of the VO.

Definition 1. Reciprocal Velocity Obstacle

$$RVO_B^A(\mathbf{v}_B, \mathbf{v}_A) = \{\mathbf{v}'_A | 2v'_A - v_A \in VO_B^A(v_B)\}$$

As described in **definition 1** the reciprocal velocity obstacle $RVO_B^A(\mathbf{v}_B, \mathbf{v}_A)$ between two agents; agent *B* to agent *A* contains all velocities for *A* that are the average of $\mathbf{v}_A$ *and a velocity inside* $VO_B^A(\mathbf{v}_B)$. The RVO can be represented geometrically as the velocity obstacle $VO_B^A(\mathbf{v}_B)$ translated such that its apex lies at $\frac{\mathbf{v}_A + v_B}{2}$ (Fig. 3.).

Reciprocal Velocity Obstacles provide a guarantee of collision-free and oscillation-free motion for each individual agent, however do not guarantee that the motion of two agents will be free from "reciprocal dances", reciprocal dances are however noted in the movement of humans amongst each other, occurring when two people cannot come to an agreement as to which side to pass each other on.

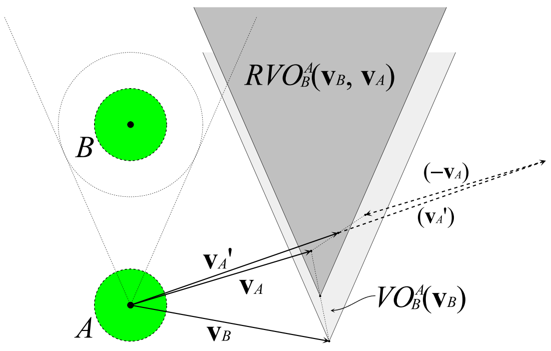


Fig. 3. The RVO of obstacle B to agent A.

4.1.3. Continuum Crowds

Research into obstacle avoidance is not only based around per-agent models. Continuum crowds (Treuille, Cooper, & Popovic, 2006) is a completely different and novel approach to handing the problem based on research into continuum dynamics and implemented using global influence maps.

Whereas models such as velocity obstacles are agent based, this is to say motion is computed separately for each agent, continuum crowds treats the navigation and obstacle avoidance as a per-particle energy minimisation problem.

Within the simulation a map is stored; the map can be structured simply as a grid lattice, or as a more advanced subdividing structure. Regardless of the data structure used, at its finest resolution each small area within the world is assigned a weight, this weight is calculated by considering agents that currently occupy that space and their respective velocities, each agent within the world exerts some “influence” on the map increasing that areas weight. While moving around agents will always try to move into areas of “low influence” thus avoiding collisions with other agents.

While an interesting avenue of research to look at, the continuum crowd’s method was deemed inappropriate for use within this particular crowd simulation. While the method does help to produce an overall more natural look and feel to the movement of agents, due to the algorithm not working on a per-agent basis it has been discredited for its ability to be extended to include individual agent personality models.

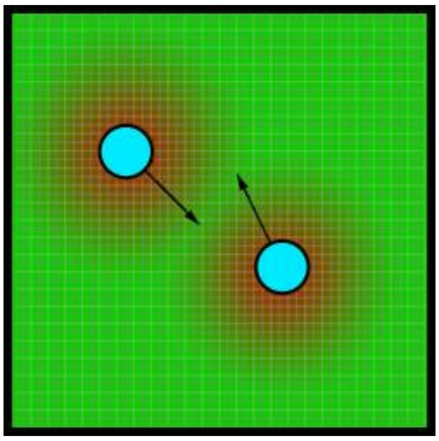


Fig. 4. A Diagram showing how agents affect the influence map and attempt to move to areas of low influence (Red areas).

For these reasons the continuum crowd's method is best utilized in situation involving very high density crowds, where it is much harder for each agent to show individuality and instead crowd dynamics govern individual behaviour.

4.2. Personality and Interaction

There is a great amount of literature covering personality and interaction between people in the real world, spanning a great many subjects. The interaction between people within a crowd is very complex however always follows similar patterns.

4.2.1. Composite Agents

Composite agents (Yeh et al. 2008) presented a new and interesting way of describing an agent, which allows for agents to interact in novel and realistic ways. Within the simulation, instead of an agent being a single entity the entity is described as a "main" entity and a collection of "sub" entities. Using this method, agents can exert limited control over their surroundings and other agents using explicitly programmed functionality.

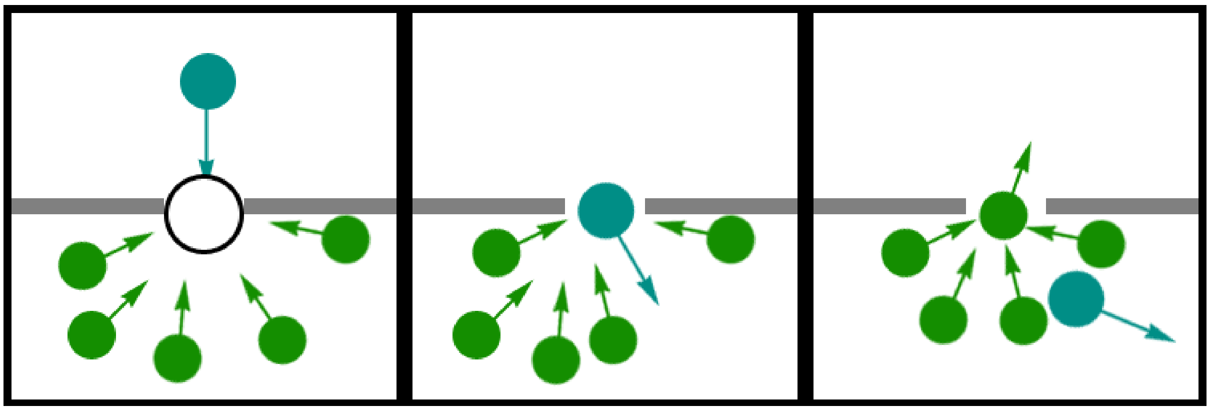


Fig. 5. 1) The agent wishing to depart the train has placed a blocker in the door way. 2) The blocker is removed once the agent is through the door. 3) The other agents can board.

One example of the functionality provided by composite agents is the situation of an agent wishing to depart a train. The agent would place a "sub-entity" in the doorway of the train blocking other agents from boarding. Once the agent has cleared the train the sub-entity would be removed, allowing other agents to then board, thus mimicking the behaviour demonstrated in a real life situation.

The major flaw with composite agents is that functionality needs to be explicitly programmed, and therefore direct and seamless integration with the existing algorithms wouldn't be possible.

4.2.2. Crowd Variation using OCEAN

Durupınar et al. attempted to solve the problem of a lack of personlity within crowd simulations in their 2008 paper "Creating crowd variation with the OCEAN personality model". They proposed integrating the OCEAN personality model (Costa & McCrae, 1996) in order to provide a visible dynamic throughout the agents.

The OCEAN model was chosen as it has become popular in a field where there is still much controversy as a reliable method of describing personality. The OCEAN model describes an individuals personality as how *open, concientious, extrovert, agreeable and neurotic* they are and is based on the work by Goldberg (1990) as well as many others.

The work involved modifying existing crowd simulation software and went on to prove that individual personality traits within agents, that are mapped to real-world psychological research produce interesting dynamics within the crowd and was therefore a great inspiration for this work. The main difference between this work and my proposal is that the integration was at a higher level then the basic movement model.

5. Design & Implementation

This section discusses, describes and attempts to justify the design and implementation of the software. The section is structured as follows; first the development of a rapid prototyping environment which allows for quick iteration and development is outlined. Following this is a discussion based on the development of the crowd simulation software, which focuses on creating a medium-density crowd and having agents successfully move around a world. The final section discusses the integration of personality models, and other algorithms which help to give the agents within the simulation the appearance of intelligence and personality.

5.1. Development environment

In order to facilitate the development of the crowd simulation software a rapid prototyping engine was first created. This engine, named "XFramework", takes care of the start-up and running of the software, enabling the quick turnaround and development of new ideas.

XFramework is designed using a component based plug-in architecture for speed of expansion and change and comes with components for; rendering, game-state management and debugging tools.

The rendering component within XFramework utilises the Ogre[1] rendering engine and handles Ogre's start-up and shutdown sequences, as well as exposing a simple API for rendering simple geometric shapes for quick visualisation of the software.

Game screens can be used to manage the state of the game, for example a "pause menu screen" can be pushed on top of the "game play screen" to create an in game pause menu.

The debugging tools component is another component that allows debugging tools to be created and used within the framework, debug tools are registered and will automatically be added to a toolbar at the top of the screen so that they can be used. Standard debug tools included as standard in XFramework which helped greatly with the development of the crowd simulation are a logging window for displaying output sent to the logger and a variable tracker which can display the value of variables within the simulation.

Many utility classes and functions are included within XFramework; examples of these are classes for handling input, posting logs and reading in initialization variables from an outside file.

5.2. Crowd Simulation

The crowd simulation software can be best described by splitting it into its four major sections.

- The simulator
- The agents
- Obstacles
- The spatial partitioning structure

The software is developed in a fashion that will allow for it to be plugged into an existing project and used for the crowd simulation needs. The main focus of the software is on the simulation as opposed to how the crowd is rendered, to accommodate different rendering approaches the simulation is completely decoupled using the model-view-controller pattern, included with the artefact is a debug renderer which will render the scene in two dimensions using the OGRE rendering engine.

5.3. The Simulator

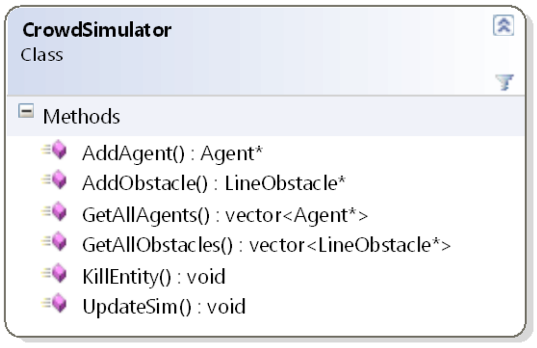
Class: ***XCS::CrowdSimulator***

This is the core of the crowd simulation. It is responsible for creation and destruction of everything within the simulation, including but not limited to agents and obstacles. The crowd simulation should be updated once per frame and passed the time elapsed since the previous frame[2]. A stable frame rate is not required, however very low frame rates will cause the simulation to not run smoothly, resulting in undesirable behaviour such as agents colliding.

During an update the simulation first partitions the world space and then updates each agent within the simulation.

[1] http://www.ogre3d.org/

[2] The expected scaling of time is a floating point number where 1.0f is equal to 1 second.


CrowdSimulator
Class

Methods

AddAgent() : Agent*
AddObstacle() : LineObstacle*
GetAllAgents() : vector<Agent*>
GetAllObstacles() : vector<LineObstacle*>
KillEntity() : void
UpdateSim() : void

Fig. 6. XCS::CrowdSimulator - Important methods

The crowd simulator class uses multiprocessing provided by OpenMP in order to process all agents in parallel. This provides a speed boost on multi-core systems helping to keep the simulation running at interactive rates.

5.4. Agents

*Class: **XCS::Agent** : derived from **XCS::Entity***

Within the simulation an agent is an object representing an individual person within a crowd. Agents are by far the most complex object within the system. An agent's primary task is to move to a target location, this movement should appear "intelligent", for example: collision with other agents and obstacles should be avoided. Agents should also seek to display individual personality and group behaviour.

Agents are updated once per frame, during an update they first obtain the set of all obstacles (static obstacles and other agents) that they must avoid. They then calculate the velocity which would take them directly to their target location, after which they modify the velocity based on other agents and obstacles that must be avoided eventually coming to a conclusion of which velocity is the best to take. The final step is to scale this velocity based on factors including whether or not they have changed direction and how many potential obstacles are in the immediate vicinity and then modify their position based on their velocity and the time-step.

5.4.1. Agent Movement

Agent movement consists of three areas; path planning, the locomotion model and local obstacle avoidance.

5.4.1.1. Path Planning

Path planning involves finding a high level path through the world from the agent's current position, to its target position. The locomotion model can then take charge of movement along the path.

Path planning has been excluded from the project as unnecessary and instead agent's attempt to move directly to their target position.

5.4.1.2. Locomotion

The agent's locomotion model implements a subset of steering behaviours (Reynolds, 1999) , incorporating *arrive* and *seek* methods. While both these behaviours are similar, belonging to the same group of "move to position" behaviours, they also both behave slightly different and thus have fundamentally different purposes. Seek will always generate a vector pointing to the target location with the same length each time. Arrive differs by generating a vector pointing towards the target location, where the length of the vector scales with the distance from the target. The result is that using seek will cause the agent to overshoot its target, whereas arrive will cause the agent to slow down as it approaches and come smoothly to a stop.

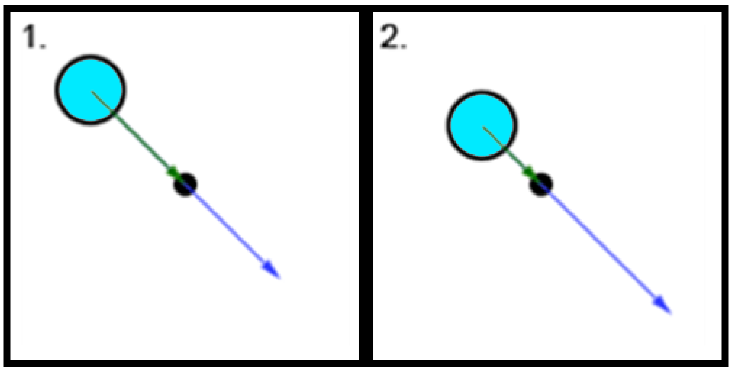


Fig. 7. Diagram showing the velocity's returned by seek (blue) and arrive (green) for an agent moving towards a target.

Even though only two steering behaviours have been implemented, the system is designed to be extensible and ready for the addition of more behaviour's, utilizing a weighted priority accumulation algorithm in preparation for the addition of new steering behaviours.

5.4.1.3. Local Obstacle Avoidance

Agents implement Reciprocal Velocity Obstacles in order to avoid local obstacles this choice was made because they are a proven per-agent obstacle avoidance method. Choosing a per-agent method is important in order to enable the integration of individual personalities. From this point onwards Reciprocal Velocity Obstacles will be referred to using the acronym RVO's.

There are two methods of implementing RVO's; time-of-impact (TOI) sampling and geometrically. Both were implemented in order to accommodate for an investigative comparison between the two methods.

Some modifications had to be made to the algorithm presented by Berg et al. in order to ensure the addition of agent personalities; the problem with the original algorithm is that agent **A** while trying to avoid collision with agent **B** would assume agent **B**'s movement based on its own movement in that situation, therefore once each agent begins to interact differently in a given situation the algorithm stops working correctly and the agents begin making mistakes.

5.4.1.4. TOI Sampling

When using the TOI sampling method rays are fired out from the agent and then scored based on two factors, these rays represent the possible velocities that are being considered.

1. How long did it take the ray to hit the first obstacle it collides with?
2. How far off the desired velocity is the ray?

The first ray to be sampled is always the ray that directly points to the agents target location. After all rays have been sampled and scored the best sample (the sample with the lowest penalty score) is chosen as the agent's new velocity.

The TOI sampling algorithm is one of the most complex algorithms in the project. Below pseudo code for the algorithm is listed and explained, this is a simplified version and assumes there are no static line segment obstacles to avoid, and also that the agent is not currently engaged in a collision with any other agent or obstacle.

```
Float minPenalty := Infinity

FOR i := 0, i < NUM_SAMPLES, i += 1
  IF i = 0
    Vector2f candVel:= steeringVelocity
  ELSE
    Vector2f candVel:= NewSample()
  END IF

  Float dV = Length(candVel,  steeringVelocity)
```

Listing. 1. Pseudo code for the first part of the TOI algorithm (sample selection).

First the algorithm initialises the minimum penalty to the largest possible value, then for each sample ray a velocity is chosen at random. The exception to the rule is the first sample, where the steering velocity is automatically chosen for the sample. The final part of this section calculates and stores how far this ray deviates off the desired path.

```
Float cT := Infinity

FOR int j := 0, j < obstacle count, j += 1
  Vector2 abVector = 2 * candVel –
                   i.   velocity -   obstacleVelocity

  Float time := CollisionTime (abVector, VO).

  IF time < cT
    cT := time;
  END IF
END FOR
```

Listing. 2. Pseudo code for the second part of the TOI algorithm (time of impact with obstacles).

During the second section all potential obstacles are iterated over in order to calculate which has the lowest collision time, i.e. the one which is closest. The penalty score can now be generated, taking into consideration the distance from the desired velocity and the minimum collision time.

```
Float penalty := cT + dV

IF penalty < minPenalty
  minPenalty := penalty
  Vector2 newVelocity := candVel
END IF
```

Listing. 3. Pseudo code for the final part of the TOI algorithm (determine if this sample is the best velocity so far).

By the end of the algorithm the sample velocity which gives the minimum penalty has been calculated and thus the decision over which velocity should be taken has been made.

5.4.1.5. Optimization - Adaptive TOI Sampling

The standard TOI sampling algorithm chooses the rays it samples as possible velocity's at random representing a brute force approach. Using the brute force approach each agent must sample around 300 – 400 possible velocity's in order to achieve smooth movement. Many of these rays are wasted, checking areas which are definitely not going to produce a possible velocity. Given the time taken to run the simulation increases linearly with the number of samples the agent takes along with how many obstacles the agent must actively avoid, the hypothesis is that minimizing the sample count whilst still maintaining a smooth simulation should provide a large performance increase.

The solution was to redesign the sampling section of the algorithm to use a smarter sampling method that can narrow down the search area and move gradually towards the best velocity.

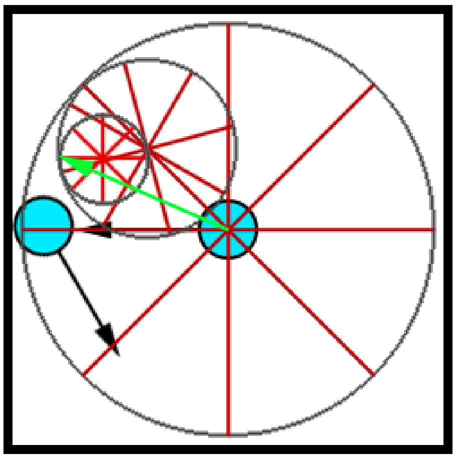


Fig. 8. Diagram showing how adaptive TOI sampling narrows in on the best choice of velocity.

The algorithm works by selecting sample points on the edge of progressively smaller circles, each circle is approximately 1/8th the size of the previous. For each circle eight points around the edge are sampled, the algorithm then proceeds as normal for this sample. For each sample, after that samples penalty has been calculated if the penalty is smaller than the previous minimum penalty then the next circles centre point is placed halfway between the current circles centre point and the sample velocity.

Whilst not mathematically guaranteed to provide the best velocity, using this new sampling method results in an agent being able to calculate an admissible velocity in around 30-40 samples, which should theoretically introduce a 10 fold speed increase over the original brute force algorithm.

5.4.1.6. Geometrical

The geometrical method involves constructing the velocity obstacle cone for each potential obstacle. This cone is then moved so that its apex lies at the average of the agent and the obstacles velocities. Fig.9. shows these cones.

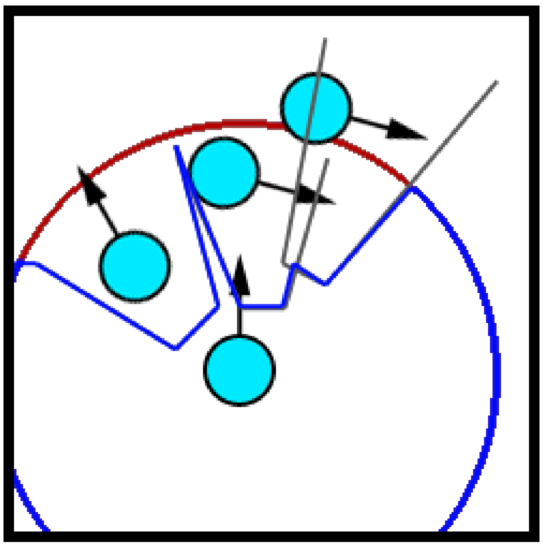


Fig. 9. Velocity obstacle cones for each potential obstacle are constructed and then a union set is created.

After a cone has been created for each obstacle we test if the desired velocity lies within any of them, if it doesn't then this velocity is safe and collision free. If not, we union all the cones together and then select the closest joint to the desired velocity as the best velocity to use.

During the development of the geometric method it became apparent that it was inferior to the TOI sampling method. The main problems started to arise in high density situations where the velocity obstacles would cover so much area that the agent had nowhere to go and a collision would take place. For this reason development of the geometrical based method was ceased in order to move focus to the development of a more advanced TOI sampling algorithm.

5.5. Obstacles

*Class: **XCS::LineObstacle :** derived from **XCS::Entity***

Within the simulation obstacles are static barriers defined as line segments with a start and end point that must be navigated around. Because a line segment is fundamentally a different obstacle to a circular object such as an agent, the velocity obstacle algorithm must handle it differently.

To integrate line segment obstacles modifications must be made to the algorithm in listing 2. The new algorithm becomes:

```
Float cT := Infinity

FOR int j := 0, j < obstacle count, j += 1
  IF obstacle IS agent
    Vector2 abVector = 2 * candVel –
                        i.   velocity -   obstacleVelocity

    Float time = CollisionTime (abVector, VO)
  ELSE IF obstacle is line obstacle
    Float time1 := CollisionTime(candVel, EndPoint)
    Float time2 := CollisionTime(candVel, startPoint)
    Float time3 := CollisionTime(candVel, side1)
    Float time4 := CollisionTime(candVel, side2)

    Float Time := min(time1, time2, time3, time4)
  END IF

  IF time < cT
    cT := time
  END IF
END FOR
```

Listing. 4. Pseudo code for the second part of the TOI algorithm, this time involving line obstacles.

The algorithm works by treating the line obstacle (LO_1) almost as if it were a swept circle that must be avoided (as shown in Fig. 10.). The time to collision is calculated for both the two circles radius *r* centred on the line segments start and end points and the two line segments parallel to LO_1 at distance *r* and –*r* away. Whichever time is the smallest indicates where on the line segment the sample ray will intersect and a penalty is applied accordingly.

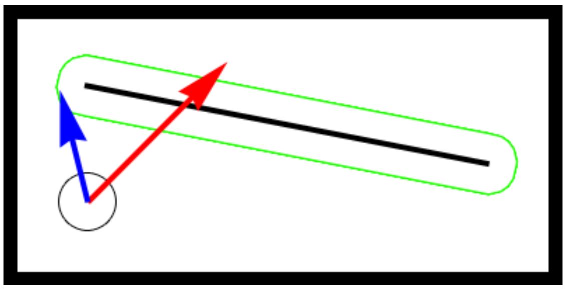


Fig. 10. Diagram showing how the agent treats the line segment as a swept circle as subsequently chooses the velocity that takes it towards the end of the segment.

5.6. Optimization – Quad-Tree Spatial Partitioning

Class: ***XCS::QuadTree*** *: composed of* ***XCS::QuadTreeNode***

A spatial partitioning structure is a way of partitioning space within the world in a logical way, the purpose of partitioning space is to allow fast spatial searches to take place. The crowd simulation software takes advantage of a quad-tree (Bentley & Finkel, 1974) structure, which is used mainly to accelerate collision checks and potential obstacle lookups for agents. The decision to use a quad tree as opposed to other data structures commonly used such as a KD-Tree (Bently, 1975) was made because fast heuristics can be used to attempt to locate highly congested areas in the world.

A Quad-tree is a data structure that utilizes recursive decomposition of the search space, extending the binary search tree from one to two dimensions, the main benefit of this search method being the reduced cost from linear $O(n)$ to logarithmic $O(n)$. Other applications exist outside of game software development, including spreadsheets and databases (Beckley et al., 1985) and image searching (Smith & Chang, 1994).

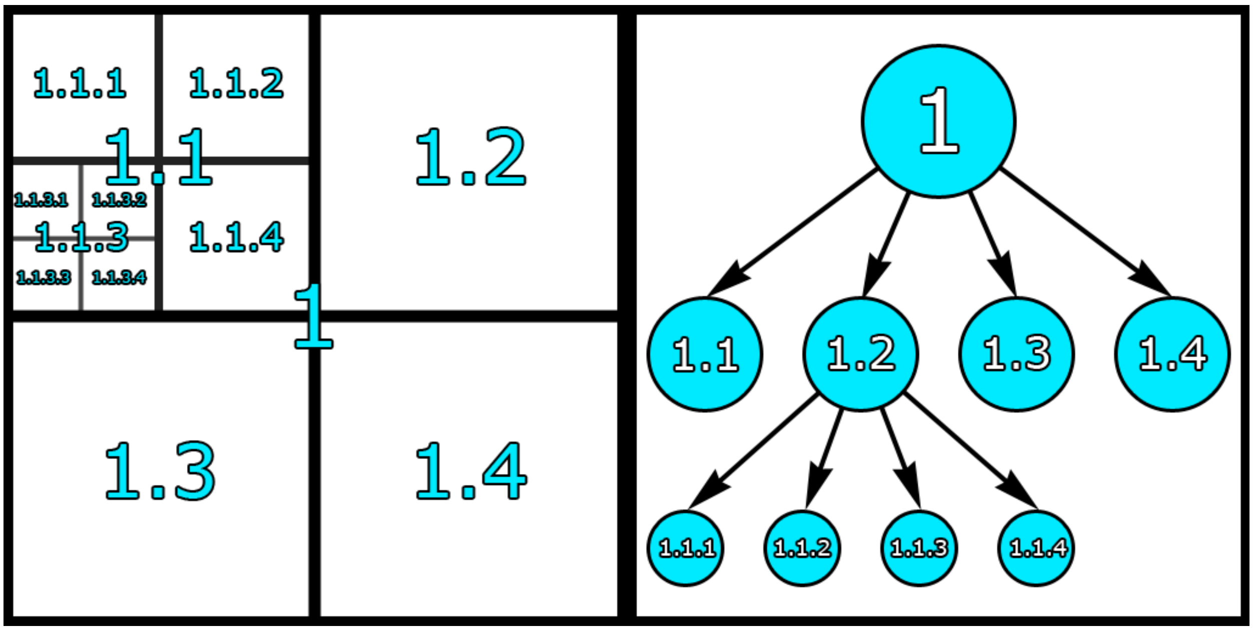


Fig. 11. Left: Quad-tree spatial overview, showing how areas are subdivided into equal size. Right: Quad-tree node structure.

The quad-tree implemented within the crowd simulation is implemented using a parent-child recursive node structure. Starting from the root node, each node can have 0 or 4 child nodes; a node without any child nodes is referred to as a "leaf node". When adding an object, the object will pass down the tree until it finds the node that it fits entirely into, if an object lies on the border of two nodes' then it is placed in the parent of those two nodes. When a node holds too many objects (defined as a parameter upon creating the tree) if the node doesn't already have children it will create child nodes and divide its objects amongst them up until a maximum tree depth has been reached.

The quad-tree is optimized for a build-query-clear usage pattern, the other option is a tree that can dynamically change growing and shrinking to match its contents however given all agents are likely to move each frame rebuilding each frame is faster.

5.7. Integrating a Personality Model

The following section discusses the various steps that were taken, and algorithms implemented in order to integrate a personality model into the crowd agents. The main area of focus was individual agent personality to order to provide active dynamic's within the crowd, as well as more intelligent path planning and velocity selection.

5.7.1. Personality using OCEAN

After the successful work of Durupınar et al. of integrating the OCEAN personality into the HiDAC crowd simulation software it became apparent that this was the first major step to take.

The five OCEAN personality variables; openness, conscientiousness, extroversion, agreeableness and neuroticism were mapped to internal variables within the simulation. Using this method it is possible to describe an agent to be added into the simulation using five variables ranging from 0.0f to 1.0f and the agents personality be created accordingly. Table 1 shows how the OCEAN traits map to agent variables.

When using OCEAN to describe an agent in the simulator a negative value is in the range [0, 0.5] and likewise a positive value is in the range [0.5, 1], with a value of 0.5 representing a perfectly centred personality. It is important to note that values are not Boolean, a value of 0.9 would affect the personality trait more than a value of 0.6.

The process of converting the OCEAN variables to proprietary internal variables is necessary because terms such as "neurotic" are meaningless to the algorithms involved. Instead the variables used were carefully chosen and mapped so that the personality model could tie in closely with the path planning, locomotion and local obstacle avoidance algorithms already being employed for each agent.

Within the simulation the variables used are;

- **Look Distance –** Affects how large the search area is when an agent decides which obstacles it should be avoiding this frame.
- **Shyness –** Shy agents will avoid going near other agents.
- **Threat** – Threat increases the distance other agents will attempt to stay away from this agent
- **Threat tolerance** – The opposite of threat, the higher this value is the more an agent will ignore another agents "threat" radius.
- **Stubbornness** – Affects how willing the agent is to deviate from its desired velocity.
- **Avoid congestion** – Agents with a high willingness to avoid congestion will actively avoid areas flagged as "highly congested".
- **Speed** – The agent's maximum speed scales according to this value.
- **Target change** – A higher value will increase the chances of the agent making sudden changes in destination.

		Trait-descriptive adjective	Variable modifications
O	+	Curious, Alert, Informed, Perceptive	LD+++, SH---, TT++, ST--, AC-
	-	Simple, Narrow, Ignorant	LD---, SP -, ST++, TT--, SH---
C	+	Persistent, Orderly, Predictable, Dependable, Prompt	ST-, TC--, SP+,
	-	Messy, Careless, Rude, Changeable	SP++, LD---, TH+, ST+, TC++++
E	+	Social, active, assertive, dominant, energetic	TT++, AC--, SH---, SP++
	-	Distance, unsocial, lethargic, vigorless, shy	TT--, AC++, SH++, SP---
A	+	Cooperative, tolerant, patient, kind	TT++, SH--, ST--, SP-
	-	Bossy, negative, contrary, stubborn, harsh	TH++, ST++++
N	+	Oversensitive, fearful, dependent, submissive, unconfident	TT--, AC++, SH++, ST--
	-	Calm, independent, confident	AC++, TT++, ST--

LD = Look Distance
SH = Shyness
TH = Threat : TT = Threat tolerance
ST = Stubbornness
AC = Desire to avoid congested areas
TC = Chance agent changes target
	• Number of + relates to how much that variable increases
	• Number of – relates to how much that variable decreases

Table 1. Table showing how the OCEAN personality model is mapped to simulation-agent specific variables

5.7.2. Dynamic velocity

Most crowd simulations make the assumption that agents should always travel at their maximum possible speed, this however tends to give the simulation a "rushed" and un-realistic feel. Logic has been incorporated into this crowd simulation software to vary the speeds an agent will travel in an intelligent way.

This step takes place after the local area navigation has chosen the best velocity for the agent and happens in two stages:

- If the chosen velocity will result in a collision almost immediately or involves a large deviation from the desired path, for example in a very crowded situation where movement is severely restricted, the agent will instead choose a velocity of 0, thus waiting for space to clear before attempting to proceed.
- A check is then made to see how many agents or obstacles are directly in-front of the agent and the velocity is down-scaled accordingly.

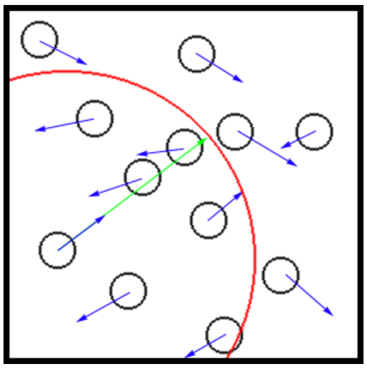


Fig. 12. Diagram shows how the focused agent would slow down because there are two other agents in front of it.

The result of these velocity changes mimic's that of real people, where stopping and waiting for other people to pass is the preferred action to take over turning around and walking the opposite way, or in a crowded situation where the person would have to walk at a slower pace.

5.7.3. Highly congested area avoidance

It was discovered through research and observations of crowds that people with certain personality types would seek to avoid congested areas, taking a wider path around, whereas other's would be happy to walk straight through the congestion. Taking this into consideration an attempt was made to recreate this effect within the simulation.

Previously it was stated that the decision was made to utilize a Quad-tree over a different sub-dividing structure because it would allow for this exact type of query to take place. Using the Quad-trees structure, a node was considered congested if the number of agents in the tree from that node onwards through its children was more than the maximum number of objects the node should hold, multiplied by the number of successive children it has.

When a node becomes congested a temporary entity is placed in the world, which should deter agents outside the entity from entering that area, instead they should pave around the outside, this works much in the same was as how a composite agent would place a sub-entity in the world to control the movement of others.

6. Experiments, Results and Analysis

All experiments and results were measured on the same computer, the system specifications were:

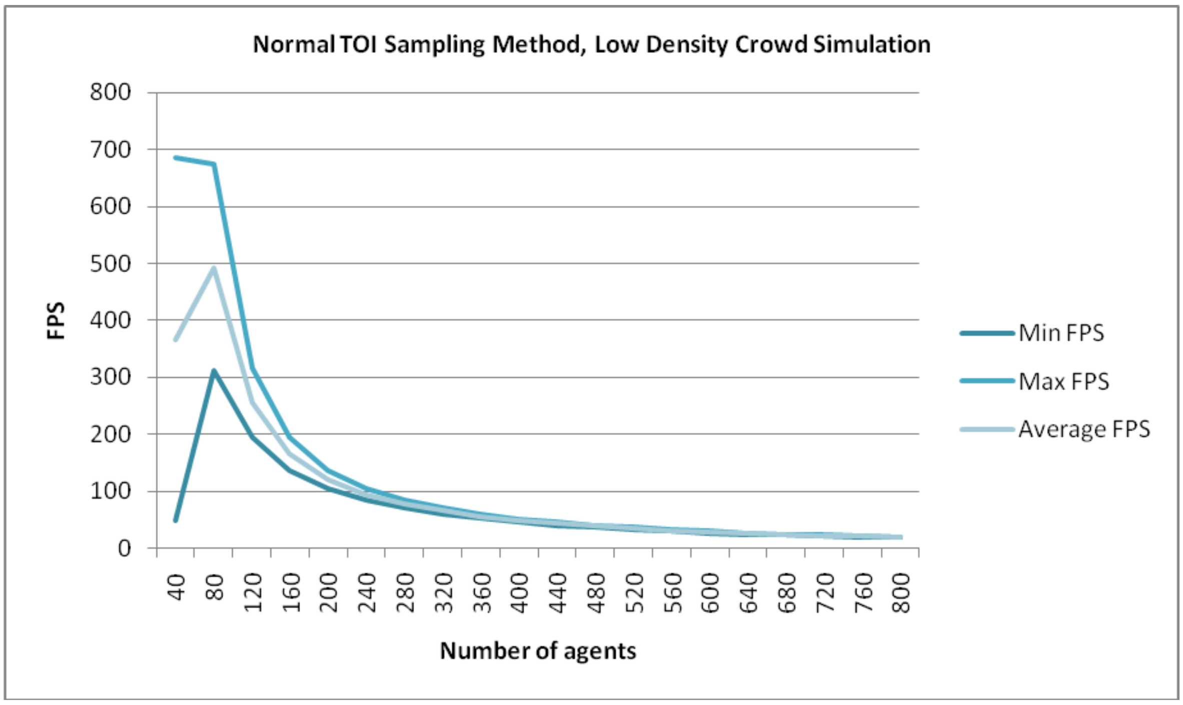
- **Processor:** Intel Dual-Core E5200, over clocked to 3.1GHz
- **RAM:** 4 GB
- **Motherboard:** ASUS P5KPL-AM
- **Graphics:** XFX NVIDIA GeForce 9600 GTO

6.1. Run-time performance

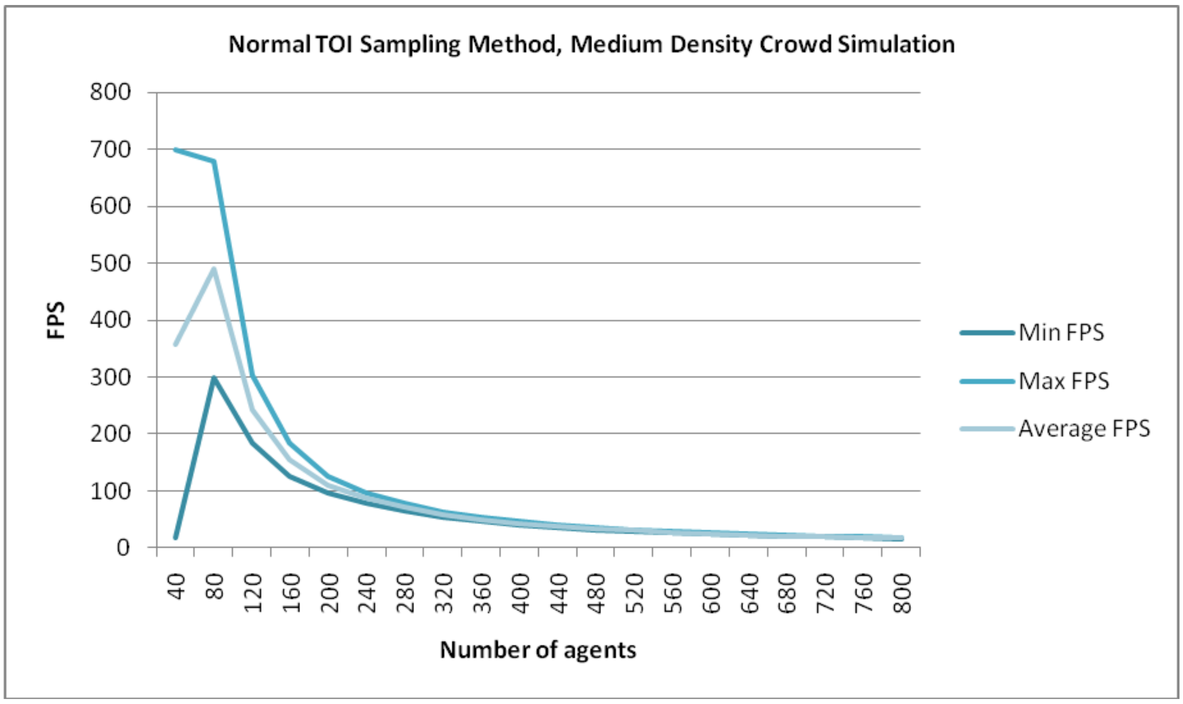
The first test performed was to measure the run-time performance of the crowd simulation software. While the software was designed with a "medium density" crowd simulation in mind, in order to stress test performance in situations outside of the software's comfort zone the test was carried out over 3 different situations; a low density crowd, a medium density crowd, and a high density crowd. Each situation involved increasing numbers of agents being created and moving around within an open area free from static obstacles. Each agent within the simulations had a randomly generated personality model. It is expected that as the number of agent's increases the fps drop will be close to linear.

The fps was measured using an automated algorithm which for each TOI sampling method and density type combination would spawn agents in group of 40, waiting 20 seconds before spawning the next group, during these 20 seconds it would record the minimum and maximum fps achieved and then work out an average. The test continued until 800 agents were spawned. Each test was run 20 times and all values averaged to ensure there were no unexpected results caused by outside interference. In total the benchmark ran for around 16 hours.

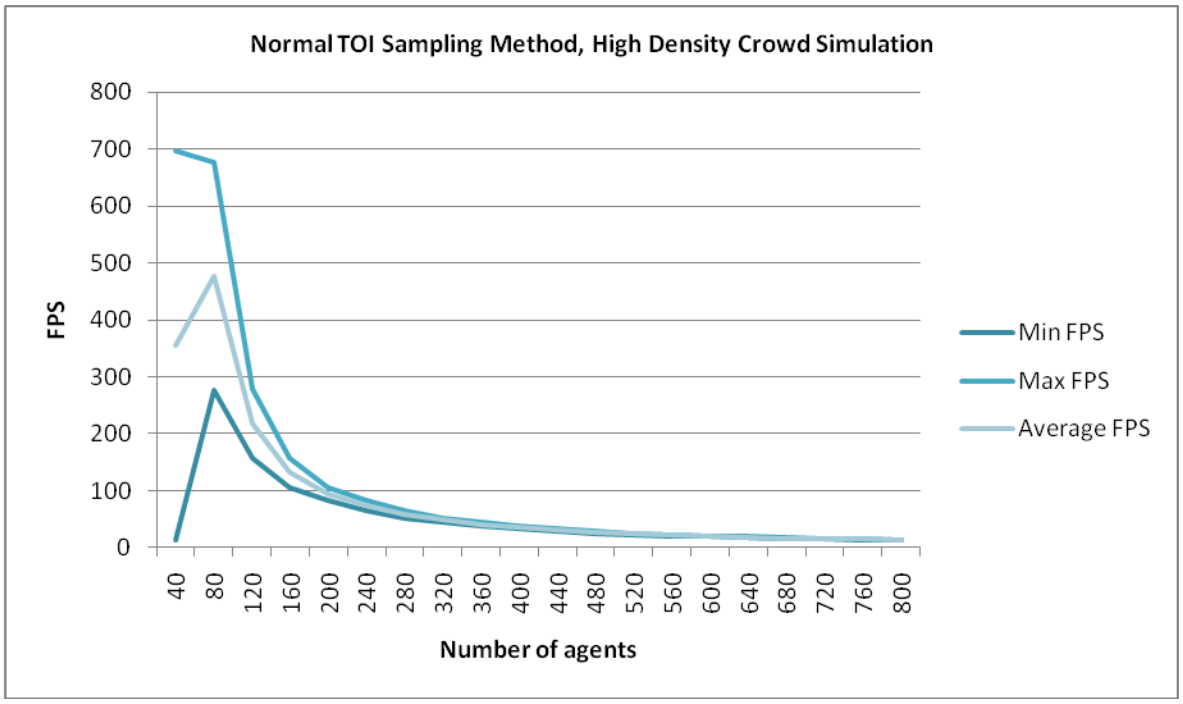
In order to test the simulation run-time rather than the render time, visualisations were turned off for the tests.



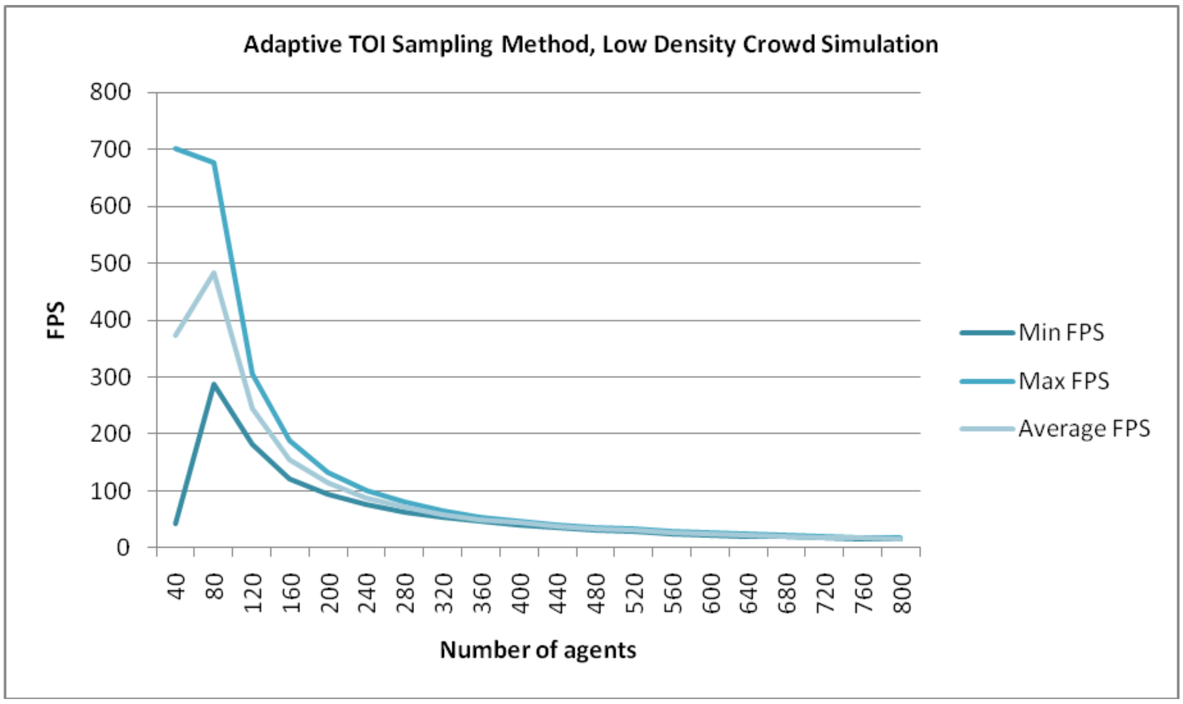
Graph 1. A graph showing the minimum, maximum and average FPS values as the number of agent's increases in a low density crowd simulation scenario using the normal brute force TOI sampling method.



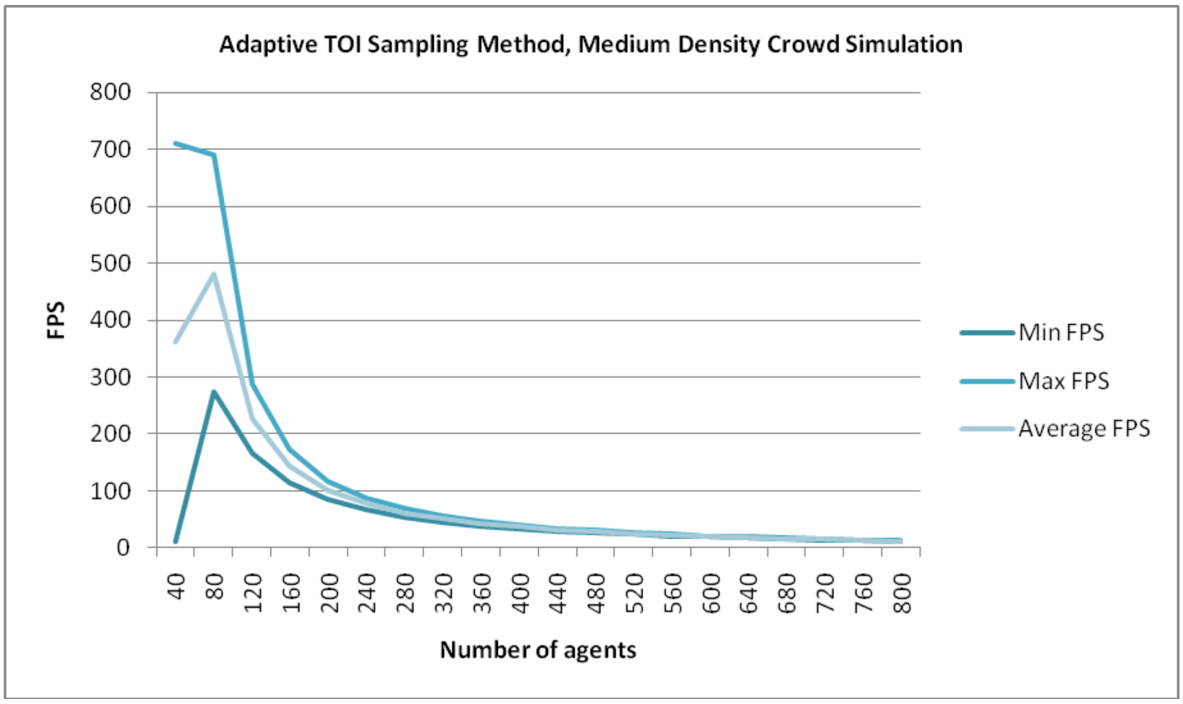
Graph 2. A graph showing the minimum, maximum and average FPS values as the number of agent's increases in a medium density crowd simulation scenario using the normal brute force TOI sampling method.



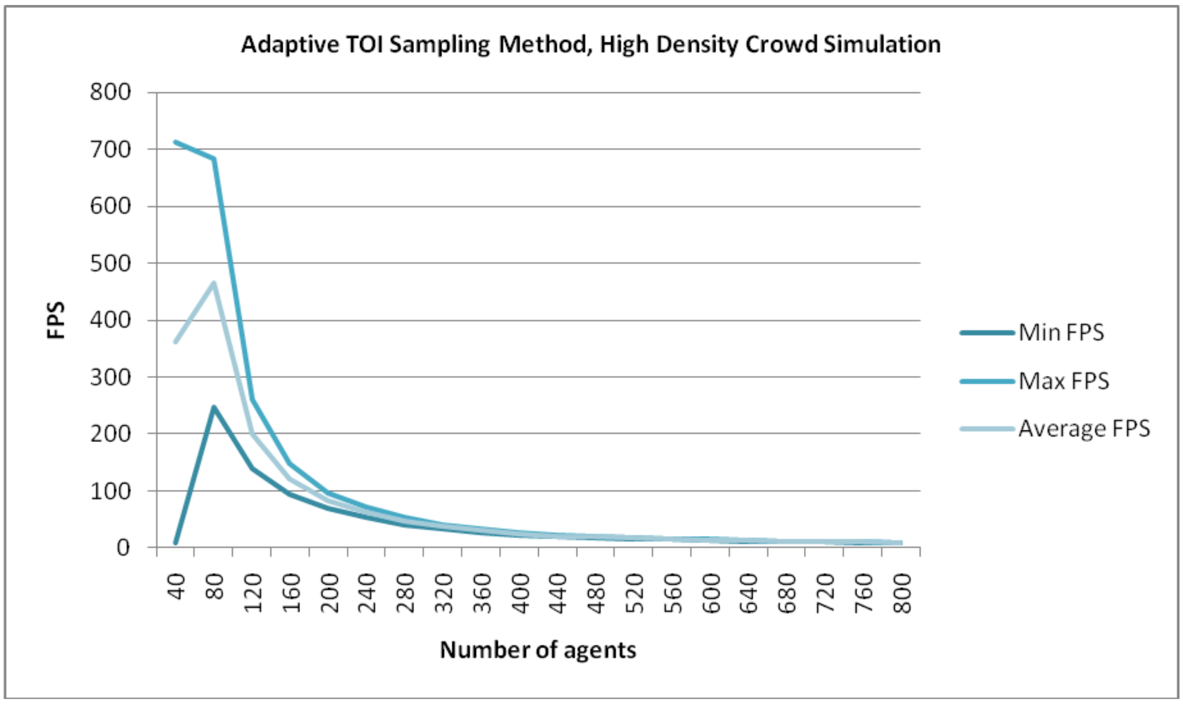
Graph 3. A graph showing the minimum, maximum and average FPS values as the number of agent's increases in a high density crowd simulation scenario using the normal brute force TOI sampling method.



Graph 4. A graph showing the minimum, maximum and average FPS values as the number of agent's increases in a low density crowd simulation scenario using the adaptive TOI sampling method.



Graph 5. A graph showing the minimum, maximum and average FPS values as the number of agent's increases in a medium density crowd simulation scenario using the adaptive TOI sampling method.



Graph 6. A graph showing the minimum, maximum and average FPS values as the number of agent's increases in a high density crowd simulation scenario using the adaptive TOI sampling method.

Graphs 1 – 6 show the minimum, maximum and average fps values achieved over each of the 6 different scenarios. It is interesting to note the rapid drop towards the start, which then levels out as the number of agent's increases, this creates a logarithmic curve, and while not what was expected, a logarithmic curve means that the number of agents being simulated could carry on increasing to large numbers before the simulation is no longer interactive.

The extremely low minimum fps values at 40 agents in each test is due to the software having to clean up the agents from the previous test, it was unexpected before the tests were ran that cleaning up the agents in this way would cause such a large performance drop and therefore indicates that the software would likely benefit from a pooling system to avoid unnecessary allocation and de-allocation.

The largest stress tests were the high density tests, higher density crowd simulations mean that each agent has to actively avoid more obstacles each frame and therefore the TOI algorithm run-time should increase, however, even in this test the fps at the largest number of agents tested (800 agents) averaged out at 18fps which is well within an interactive frame rate.

Recording the minimum and maximum fps values achieved at each agent count caused an interesting pattern to emerge. It can been seen from the graphs that there is a much larger gap between minimum and maximum fps when there is less agents on the screen, the gap is also increases slightly with the density of the simulation. This gap is likely due to agents bunching up causing small high density areas, in this situation each agent suddenly has to avoid many more obstacles and thus the run-time increases. This effect is shown in fig. 13.

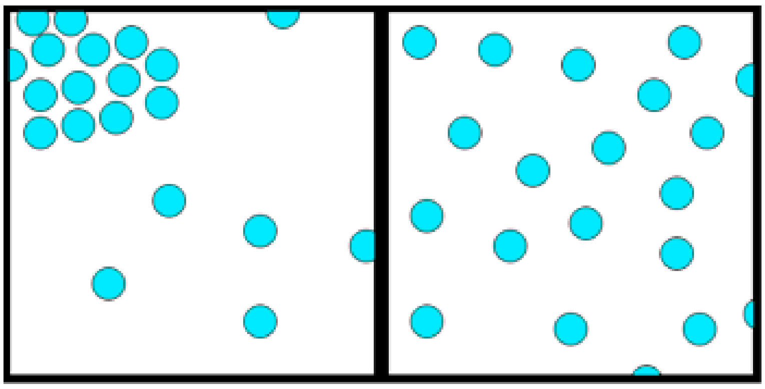
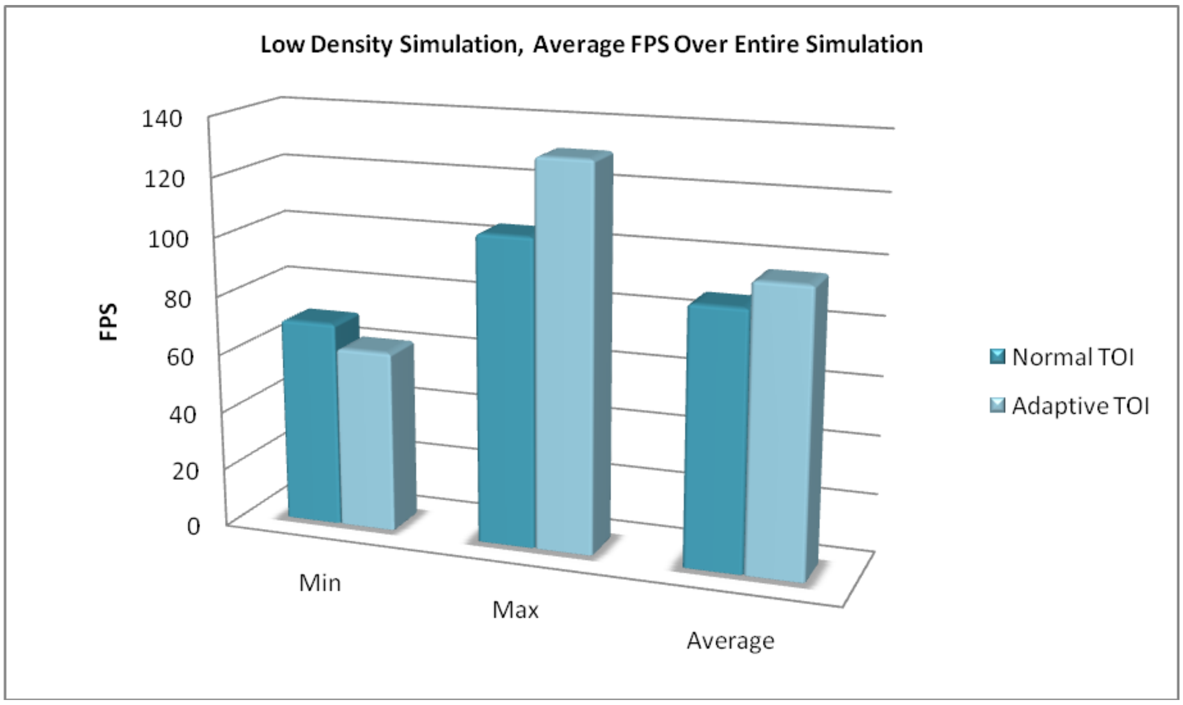
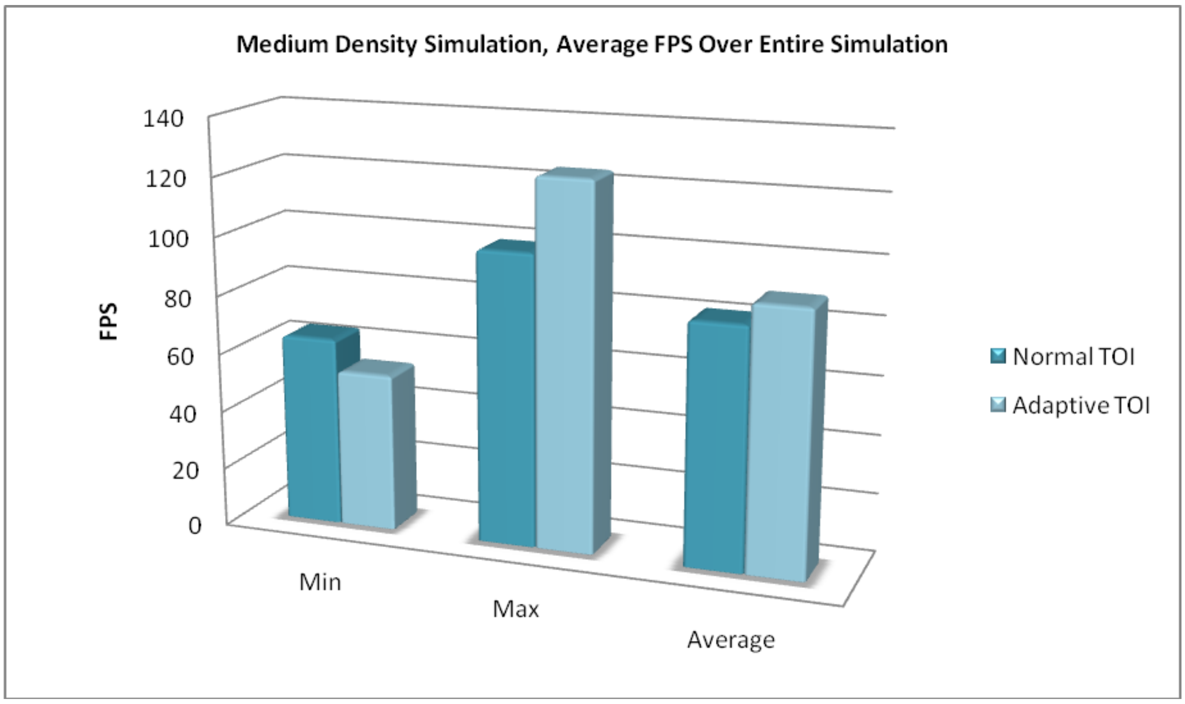


Fig. 13. Left: 20 agents that are huddled up together causing lower fps. Right: 20 agents spread out evenly resulting in much higher fps.

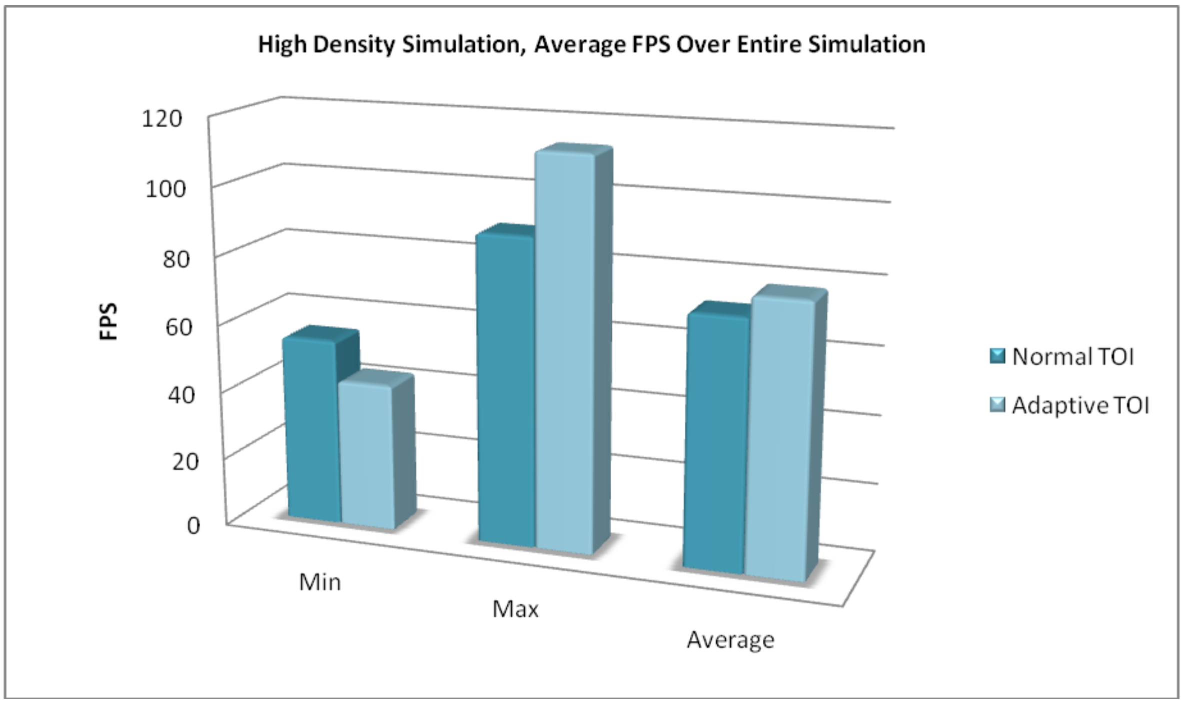
Graphs 1 – 6 do not show that the run-time of the simulation has decreased drastically with the integration of the adaptive TOI sampling algorithm, close analysis show the adaptive algorithm performing less well when the number of agents is low, however the fps drops off slightly later, giving better mid-range performance. In order to compare the two algorithms side by side, the FPS values gained over an entire simulation from 40 to 800 agents were averaged out; this value is important because it reflects how the software performs while handling an actual crowd scenario where the number of agents will vary. Graphs 7, 8 and 9 compare the performance of the two algorithms over low, medium and high density scenarios.



Graph 7. A graph showing how adaptive and normal TOI sampling techniques compare over the course of an entire simulation ranging from 0 to 800 agents in a low density simulation.



Graph 8. A graph showing how adaptive and normal TOI sampling techniques compare over the course of an entire simulation ranging from 0 to 800 agents in a medium density simulation.



Graph 9. A graph showing how adaptive and normal TOI sampling techniques compare over the course of an entire simulation ranging from 0 to 800 agents in a high density simulation.

The results from these tests show that while not the 10 fold performance increase that was expected there is a definite improvement in the maximum and average performance across each test. There is also a drop in minimum performance in each test. It is believed that while the algorithm requires a fraction of the velocity samples that the brute force algorithm requires, the algorithm itself required trigonometry functions to select rays and therefore takes slightly longer to calculate each candidate velocity. It could be assumed that an adaptive TOI sampling algorithm that uses a lighter selection technique would increase performance even more, for example corners and midpoints of boxes could be used to approximate the circles.

The results of this set of tests proves that the crowd simulation software can be described as real time, as even under high stress it performs at interactive rates. It is also shown here that the integration of an adaptive TOI sampling technique has further improved performance across an entire simulation.

The full results tables that these graphs were derived from can be found on the accompanying CD.

6.2. Extroverts Vs Introverts

The aim of this experiment was to test how two groups with personality's at polar opposites react. 100 agents were created around the edge of a circle, and then asked to move towards the centre of the circle. Half the agents created had "introvert" biased personalities [OCEAN: 0.5, 0.5, 0.0, 0.5, 0.5] and the other half were "extrovert" biased [OCEAN: 0.5, 0.5, 1.0, 0.5, 0.5].

The expected behaviour from this experiment is that the extrovert agents would move towards the centre, where as the introvert's wouldn't like the congestion and would stay around the outskirts of the group, even though their target is at the centre.

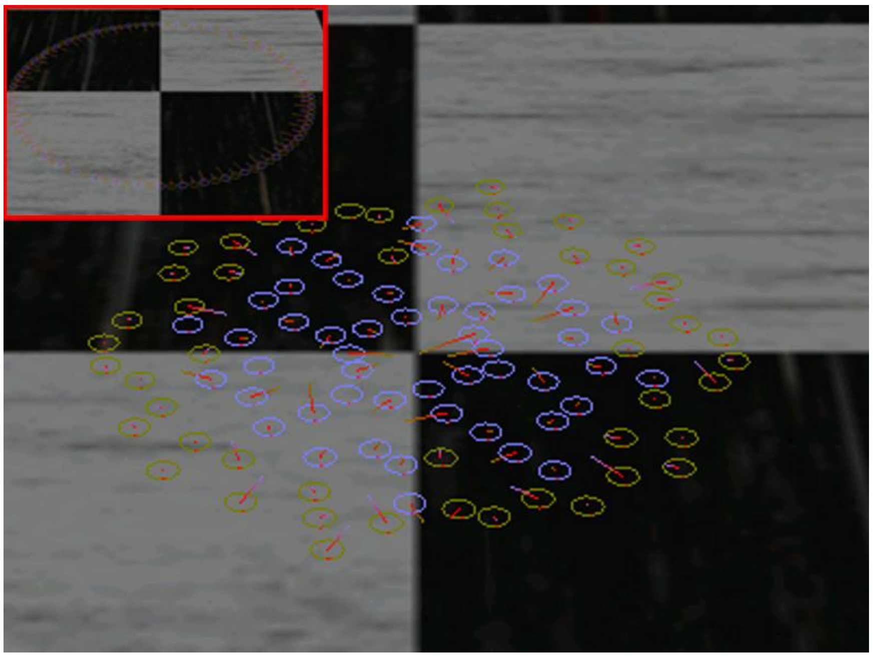


Fig. 14. Introverts stay towards the outside of the circle.

The result of the experiment was as expected, the blue agents are the extroverts and rush straight to the centre where they appear to "compete" with each other to be the agent that stands on the centre point, where as introvert agents opt to stay on the outskirts of the group, and prefer to not get mixed up in the barging around that is going on in the centre.

It is hypothesised that a major factor that aids this behaviour is the small speed increase that extrovert agents have over the introverts, this speed allows them to get ahead at the start, thus ensuring that no introvert agents become stuck in the centre of the circle, unable to get out.

6.3. Lane formation

Lane formation is a behaviour known to spontaneously emerge within human crowds under certain conditions. Most commonly where there is bi-directional traffic walking towards each other, the people within the crowd will begin to form “bands”, with each band being composed of pedestrians with a common preferential direction (Milgram & Toch, 1969).

The emergence of lane formation within the crowd simulation is important for two reasons, first of all lane formation facilitates the fast movement of agents within the crowd, helping to stop agents from becoming stuck in congested areas. The second reason is to prove that the crowd simulation is realistic.

Given no quantitative method to test for lane formation a qualitative method was used, different scenarios were created and ran and the simulation was examined for the emergence of lane formation. Figure 15 shows 4 scenarios and attempts to point out areas of lane formation.

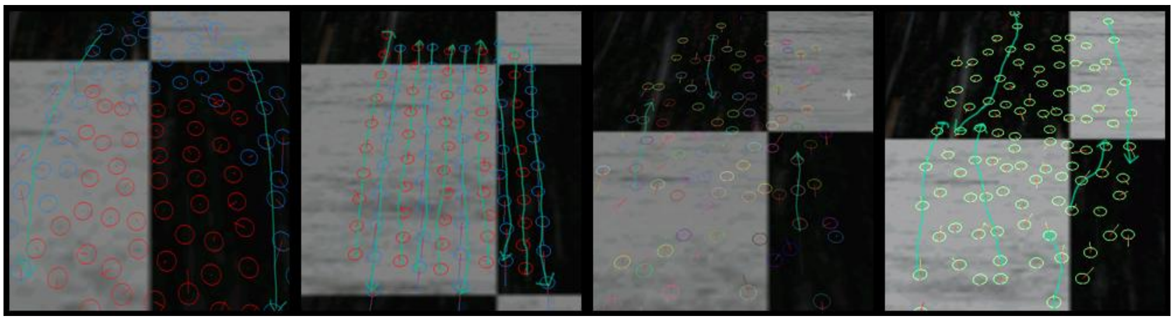


Fig. 15. From left to right: Test 1, two groups of agents with polar opposite personalities in close formation move towards and past each other, the red agents are neurotic and the blue agents are not. Test 2, two groups of agents with polar opposite personalities with large spaces between each agent move towards and past each other, the red agents are neurotic and the blue agents are not. Test 3, two groups of agents with randomly generated personalities in close formation move towards and past each other. Test 4, two groups of agents with identical personalities in semi-close formation move towards and past each other.

One of the deciding factors in choosing to use RVO’s for local agent obstacle avoidance was that natural phenomena such as lane formation are supposed to emerge naturally, however these tests appear to have shown the opposite. Other interesting observations were however made during these tests.

In test one two groups of polar opposite agents were sent at each other, the first group were classed as highly neurotic agents [OCEAN: 0.0, 1.0, 0.0, 0.5, 1.0] with the other group being made up of highly rational agents [OCEAN: 0.5, 0.0, 1.0, 0.0, 0.0]. Both groups were in close formation. When the agents arrived instead of forming lanes to pass they began to huddle up unable to move through, eventually the non-neurotic people moved around the outside. While not strict lane formation the observation was made that once one agent began to move that way, the other agents followed in the path that was now forming. A second observation made at this point was how the two groups of agents with differing personalities reacted; the neurotic people appeared panicky with quick changes in direction and a desire to stay away from other agents, the rational agents however were the ones to move around the outside, almost mimicking real-life behaviour where a rational person would think the situation through and act carefully.

Test 2 was similar to test 1 however there were large spaces in between the agents, this way they were able to easily move into each other as they passed. While this experiment provides the largest demonstration of lane formation it is however a contrived experiment that doesn't resemble a real-life situation.

For the third test two groups of agents with random personality models move towards and past each other. In this test lane formation was barely noticeable. The fourth and final test performed was similar to test three; however in this simulation all the agents had the same, centred personality [OCEAN: 0.5, 0.5, 0.5, 0.5, 0.5]. The interesting thing to note with the final two tests is that lane formation is much more prominent in the simulation where the agents have perfectly centred personalities. Through-out the development of the software as modifications were made to the RVO algorithm to accommodate a personality model it became apparent that the algorithms effectiveness to perform its original task lowered. This could explain why emergent behaviour such as lane formation is not being displayed as much as expected.

7. Conclusions

The work that was carried out as part of this project proved to be an interesting and challenging undertaking, demonstrating both success and failure over the course of the development. Considering the research conducted prior to development, at the time of publishing this work marks a new, pioneering attempt at integrating personality into crowds. While personality models have been built upon existing crowd simulations, and attempts have been made to integrate them at a high level, at the time this work was produced there was no evidence to suggest any academic work regarding the implementation of a personality model directly into the navigation and locomotion models of individual agents within a crowd simulation. Research into velocity obstacle algorithms also produced no academic evidence of an "adaptive time-of-impact" sampling method being used to accelerate a TOI based velocity obstacle algorithm. An extra challenge was presented by the lack of academic research into many of the areas covered within this paper, which meant novel solutions had to be created.

The implementation of the crowd simulation software is seen to have been successful, the artefact produced has been tested and proven to be able to simulate crowds of up to 800 agents in high density scenarios at interactive rates, whilst ensuring each agent is individual and behaves differently. The results and observations made on the software were however contrary to those which were expected.

The first investigation into the run-time performance of the software proved that the software was able to perform comfortably outside the areas that were originally focused on. The test also validated the use of a more advanced TOI sampling technique. Although the adaptive TOI sampling algorithm theoretically should have produced a much larger increase in performance, the increase in performance it did produce was noticeable.

The second part of testing, regarding the experiments that attempted to prove the successful integration of a personality model was much harder to conduct then the performance experiments. While the performance experiments were able to produce solid quantitative results, there was no such meaningful test that could be produced to prove personality; therefore all results were based on observation. The observations that were made did however go some distance towards proving that it is possible to map personality directly into navigation and locomotion, however for a realistic simulation personality integrated into the navigation and locomotion models can only make up a small part of the personality model as a whole, and other areas need to be looked at as well, such as interaction and path planning on a larger scale.

The main problem with the personality model implementation is that it pays no care to the varying amount of individuality and personality that each member of a crowd is able to show. In small, sparse scenarios each person is able to show individuality, however once the crowd becomes larger and more dense each person is less able to demonstrate individuality and instead emergent crowd dynamics take over. This effect was noted during the research stages of the project when the decision was made to use RVO's over an influence map based approach, however as the definition of at which point members of a crowd lose their individuality was unknown, completely discrediting influence maps could be considered a bad decision. With hindsight a better method would be to employ both methods, influence maps at a higher level path planning stage, moving into RVO's for the actual local area obstacle avoidance; as the density of the crowd increases, so too would the favour of using influence maps over RVO's, thus enabling seamless transition between low density scenarios where agents are able to show individual personalities, to high density scenarios where crowd dynamics take over completely.

The second problem that has been demonstrated is that while agents act individually, and demonstrate a degree of personality, they still don't behave intelligently. There are many cases where human reasoning would bypass the constraint of "I want to get to my target as fast as possible". A human would also exhibit behaviour such as avoiding someone who they "don't like the look of" or avoid an area completely and take a longer rout because the area has been known to be dangerous or congested. This intelligence must be shown at a higher level than the local area obstacle avoidance and locomotion models.

Overall it is believed this work can be seen as successful in its attempt to fulfil the initial aims and requirements, however it has emerged that the field is much larger then this paper attempted to tackle. Too many oversights and assumptions were made, and thus, while this work is successful, it is still far off becoming a full crowd simulation software that incorporates and communicates realistic individual personality and intelligence.

8. Further work

While this paper presents one method of integrating personality into a crowd simulation, it also opens up many avenues for further research. As noted in the experiments, analysis and conclusion sections, while the integration of personality has been successful as far as making agents appear individual and unique, the personality integration also needs to move into other areas of the simulation, namely that of individual agent reasoning and intelligent behaviour.

Another avenue of further research would look at ways of expressing personality, no personality model is complete without being able to visualise and communicate this to the viewer, therefore research into human interaction, regarding posture and body movement would be very interesting to pursue.

9. References

Abe, Y., & Matsuo, Y. (2001). Collision avoidance method for multiple autonomous mobile agents by implicit cooperation. *IEEE Int. Conf. on Robotics and Automation*, (pp. 1207-1212).

Beckley, D. A., Evens, M. W., & Raman, V. K. (1985). Multikey retrieval from K-d trees and QUAD-trees. *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data* (pp. 291-301). Austin, Texas, United States: SIGMOD '85 ACM Press, New York, NY.

Bentley, J. L., & Finkel, R. (1974). *Quad Trees: A Data Structure for Retrieval on Composite Keys.* Acta Informatica, Volume 4.

Bently, J. L. (1975). *Multidimensional Binary Search Trees Used for Associative Searching.* Stanford University.

Berg, J. v., Ming, L., & Manocha, D. (2008). Reciprocal Velocity Obstaclesfor real-time multi-agent navigation. *IEEE International Conference* (pp. 1928 - 1935). IEEE.

Buckland, M. (2005). *Programming Game AI by Example.* Wordware.

C, R. (2006). Gig fast crowds on ps3. *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames* (pp. 113 - 121). ACM Press.

Costa, P. T., & McCrae, R. R. (1996). Toward a new generation of personality theories: Theoretical contexts for the five factor model. *In J. Wiggens (Ed.), The Five Factor Model of Personality* .

Durupınar, F., Allbeck, J., Pelechano, N., & Badler, N. (2008). Creating Crowd Variation with the OCEAN Personality Model. *Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems.*

Fiorini, P., & Shiller, Z. (1988). Motion Planning in Dynamic Environments Using Velocity Obstacles. *The International Journal of Robotics Research* (17), 760 - 772.

Goldberg, L. R. (1990). "Description of Personality": The Big-Five Factor Structure. *Journal of Personality and Social Psychology , 59* (6), 1216 - 1229.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., et al. (2009). *ClearPath: Highly Parallel Collision Avoidance for.* SIGGRAPH Symposium on Computer Animation.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Trans.

Jacobs, J. (1961). *The Death and Life of Great American Cities.* Pelican.

Kluge, B., & Prassler, E. (2007). Reflective navigation: Individual behaviors and group behaviors. *IEEE Int. Conf. on Robotics and Automation*, (pp. 4172-4177).

Lamarch, F., & Donikian, S. (2004). Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23*, (pp. 509-518).

Milgram, S., & Toch, H. (1969). Collective behaviour: crowds and social movements. In G. Lindzey, & E. Aronson, *The handbook of social psychology* (pp. 507-610). Reading.

Pelechano, N., Allbeck, J. M., & Badler, N. I. (2007). *Controlling Individual Agents in High-Density Crowd.* Eurographics/ ACM SIGGRAPH Symposium on Computer Animation.

Pettre, J., Laumond, J. P., & Thalmann, D. (2005). A navigation graph for real-time crowd animation on multi-layered and uneven terrain. *First International Workshop on Crowd Simulation.*

Reynolds, C. W. (1999). Steering Behaviors For Autonomous Characters. *proc. Game Developers Conference 1999* (pp. 763-782). San Jose, California: Miller Freeman Game Group.

Shao, W., & Terzopoulos, D. (2005). *Autonomous Pedestrians.* Eurographics/ACM SIGGRAPH Symposium on Computer Animation.

Smith, J., & Chang, S. (1994). Quad-tree segmentation for texture-based image query. *Proceedings of the Second ACM international Conference on Multimedia* (pp. 279-286). San Francisco, California, United States: MULTIMEDIA '94. ACM Press, New York, NY.

Sud, A., Andersen, E., Curtis, S., Lin, M., & Monocha, D. (2007). Realtime path planning for virtual agents in dynamic environments. *IEEE VR.*

Thalmann, D., O'Sullivan, C., Ciechomski, P., & Dobbyn, S. (2006). *Populating Virtual Environments with Crowds.* Eurographics 2006 Tutorial Notes.

Treuille, A., Cooper, S., & Popovic, Z. (2006). Continuum Crowds. *ACM , 25* (3), 1160 - 1168.

Yeh, H., Curtis, S., Patil, S., van den Berg, J., Monocha, D., & Lin, M. (2008). *Composite Agents.* University of North Carolina at Chapel Hill. Eurographics / ACM SIGGRAPH Symposium on Computer Animation.

Yersin, B., Maïm, J., & Morini, F. (2008). *Real-time crowd motion planning, Scalable Avoidance and Group Behavior.*