Large-Scale Agent-Based Pedestrian and Crowd Simulation in Real-Time

HENRIK BOSTRÖM
and LUKAS WENSBY

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2012

Large-Scale Agent-Based Pedestrian and Crowd Simulation in Real-Time

H E N R I K B O S T R Ö M
a n d L U K A S W E N S B Y

DD143X, Bachelor's Thesis in Computer Science (15 ECTS credits)
Degree Progr. in Computer Science and Engineering 300 credits
Royal Institute of Technology year 2012
Supervisor at CSC was Michael Minock
Examiner was Mårten Björkman

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2012/bostrom_henrik_OCH_wensby_lukas_K12014.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

May 20th, 2012

Large-Scale Agent-based Pedestrian and Crowd Simulation in Real-Time

Written as a part of the
Degree Project in Computer Science, First Level
at the Royal Institute of Technology
Stockholm, Sweden
Supervisor: Michael Minock

Henrik Boström
Kungliga Tekniska Högskolan
hbo@kth.se

Lukas Wensby
Kungliga Tekniska Högskolan
wensby@kth.se

Abstract—This paper deals with the implementation details and results of simulating a city populated by a large number of pedestrians. The goal of the simulation was to, as realistically as possible, simulate large numbers of people going about their daily lives, interacting with each other and the city environment, in real-time. We also simulated dense crowds and realistic collision avoidance techniques, and tried to replicate some observations of previous studies. Simulators based on the mechanics of human interaction can easily become inconveniently complex and/or resource expensive. As this has been the main risk during the project we've been careful in the implementation to keep coupling as low as possible and to construct interfaces that allow for scaling and adding of new behaviour in "isolation", without having to modify prior code. The concern for performance was just as real – the simulation, after all, was to support thousands of interacting pedestrians walking about in real-time.

In the end, the resulting simulation turned out to be a good and efficient representation of inner-city pedestrians, and was mostly fine in handling the issues of denser crowds. This may potentially be extended for use in city and public transport planning, producing large amounts of data for data mining or as a basis for further development into city life dynamics and the artificial intelligence of individuals in a populated environment.

Index Terms—Agent-based, Crowd simulation, Large-scale, Real-time, Pedestrian simulation

1 Introduction

1.1 Introduction

The goal of our project was to simulate large amounts of individuals in a big system, all of which interact with one another and the environment. As it scales up, situations quickly become more complex and the simulation becomes more process- and resource costly. You might want to model just a few individuals, or you could attempt to model hundreds, thousands, tens of thousands or even hundreds of thousands of individuals, but such large simulations would be unlikely to run in real-time on an average personal computer. We imagined that our simulator would handle at least several hundreds or even thousands of pedestrians, running in real-time. We find it more rewarding and thrilling to be able to see the system interact with itself in real-time, and to have the possibility of intervening as well, as one would naturally want to do. This puts some constraints on the scale and implementation aspects, and furthermore it does need to be able to run on the average personal computer (since we've currently misplaced our supercomputer).

Pedestrian and crowd simulation is, amongst others, a subject of artificial intelligence – what are the pedestrians' goals and how do they interact with each other and behave when various, perhaps unexpected, situations arise? The pedestrians could act based on personal needs and goals, or based on randomness and statistics, or a combination of the two. If the behaviour of the pedestrians – on individual level, as large groups or a population – successfully models the real world, a simulator like this could potentially be used for city and public transportation planning and to model the results of large scale events such as major sport arena events. If the pedestrians have the proper behavioural responses to stressful situations then maybe the simulation could be modified to predict some of the immediate consequences of various crisis situations such as buildings on fire, natural disasters or terrorist attacks. This is pure speculation and beyond the scope of our simulation, but we've aimed for the implementation to be as scalable as possible, enabling further development. Another potential application is to use the final results of our simulator (without modifying the pedestrians' behaviour) in order to collect large amounts of statistical data and to analyze patterns spawned by the pedestrians (again, assuming that the simulation successfully models the real world). This data could then be mined in search for interesting anomalies or to be used for machine learning.

1.2 Document Overview

In *2 Background* we discuss the use and applications of crowd simulation, list some observations from previous studies that may be used as requirements or goals, and begin to talk about some of the approaches to implementing crowd simulators.

In the next section, *3 Individual Agents Approach*, we talk about the agent-based approach in the context of our simulation and why we've chosen this camp for our simulator. Then, in section *4 Implementation*, we go into details about our approach and its final implementation, attempting to give an as extensive description of the essentials as possible without giving an overwhelming amount of detail.

Following this comes *5 Results*, where in we present the outcome of our implementation, its performance – i.e. the number of pedestrians it managed to support – and of course, how close to reality our pedestrians behaved in various situations. This is followed by a discussion in *6 Discussion* about the results, which is followed by the final conclusions in *7 Conclusion*.

1.3 Statement of Collaboration

Background research and discussions about the project's goals and approach has been done equally by Boström and Wensby. As far as the implementation goes, Boström has done more of the back-end graphics coding, being formerly familiar with the library used, and geometry/math stuff. The rest of the implementation is a mix of both partners, and code written by one part is much likely edited by the other later. The same goes with the report, which has been collaboratively created using Google Documents (enabling both authors to edit the document simultaneously).

2 Background

2.1 Background and Applications

There has been an increasing number of studies in recent years looking into the subject of simulating crowds of people realistically and/or efficiently. Some for the purpose of city planning, architectural aspects or simulating disasters such as fire outbreaks in buildings where escape panic, crowd turbulence and stampede accidents are a big concern. There has also been a large number of popular games released that include some sense of artificial intelligence for individuals or crowds of individuals with relevance to our subject.

According to Thalmann and Musse [10], the dominant domain for crowd simulation is that of safety science and architecture. In architectural design of buildings, measuring and accounting for how safe a layout is in situations like fire and a panic outbreak is obviously too late when the building has already been built.

In a completely different area, crowd simulation is often used for the 3D computer graphics in films. A good example of this is the massive number of independently acting warriors and monsters seen in the Lord of the Rings trilogy [6], and we are sure that you could create an extensive list of titles of this if you wanted.

In video game series such as Sim City or RollerCoaster Tycoon you had to build roads or paths for the pedestrians to walk on so that they could reach various destinations. Their behaviour was very simple, just walking along designed paths. In the Grand Theft Auto series, the pedestrians was not just part of the scenery but could also be interacted with. In a game where the player could (and would) drive on the sidewalks and shoot people for no apparent reason, these pedestrians were programmed to try to avoid oncoming cars and interrupt what they were doing and run in panic if the player started shooting. Although fairly primitive responses, such behaviours were enough to make the city feel like it was inhabited by "living" people. These games did not focus on crowd forming or "crowd behaviour" at all. In the Dead Rising series however, almost all the people of intensely crowded areas (such as shopping malls) had turned into zombies, and the sheer number of individual zombies were impressive to look at. However, because they were "zombies", their behaviour was not complicated at all – they were just mindlessly trying to move in the direction of the player if he was close enough, and otherwise shuffle around slowly and randomly – not a very realistic crowd simulation. The Sims series, on the other hand, focuses more on individual "Sims" with wants and needs, motivated by these wants and needs to do things, and although highly controlled by the player, if a Sim for example got hungry, it "wanted" to – and needed to – eat.

2.2 Introduction to Crowd Simulation

Existing work in these areas of simulation is said to be classified either into agent-based methods, which focus more on individual behaviour, or crowd simulations, that aim to exhibit emergent phenomena of the groups [9]. One can either focus on achieving these phenomena from the top down, by programming the flow of the crowd as a whole, or one can focus more on the individuals, and from the bottom up try to recreate the phenomena, adjusting the behaviours of the individuals accordingly. These two camps are not distinct and can be combined.

Where the density of individuals is high or where there are "bottlenecks", individuals can't simply walk in the direction of their goals because of other obstructing individuals and objects. In these cases certain phenomena arise that are of high interest to some simulations.

> "Human crowds display a rich variety of self-organized behaviors that support an efficient motion under everyday conditions. One of the best-known examples is the spontaneous formation of unidirectional lanes in bidirectional pedestrian flows. At high densities, however, smooth pedestrian flows can break down, giving rise to other collective patterns of motion such as stop-and-go waves and crowd turbulence. The latter may cause serious trampling accidents during mass events. Finding a realistic description of collective human motion with its large degree of complexity is therefore an important issue." [7]

There are a number of interesting observations that have been made regarding pedestrians' behaviour. We present a condensed, summarized list of a few observations based on Helbing, et al.'s [2] research.

- Pedestrians will most likely choose the fastest – and straightest – path towards their goal, even if the direct route is crowded. They refrain from taking detours or moving opposite to the desired direction.
- Pedestrians prefer to walk at individual walking speeds (personal preference).
- Pedestrians keep certain distance from other pedestrians and from obstacles. The distance decreases as the crowd density increases or if they are in a hurry or around "attractive" places. Resting pedestrians tend to be uniformly distributed.
- Individuals who know each other form groups that may act as single entities. Loscos et al. notes that, typically, only around half of pedestrians walk alone, the rest walk in groups of varying sizes [5].

Also, pedestrians obviously don't walk indefinitely. Usually they start at one building and end at another, and on their way there they may (or may not) do various actions such as window shopping, stop to talk to another pedestrian, queue for a bus [4] or may use various other vehicles (personal cars, taxis, etc) for transportation.

2.3 A Brief Overview of Approaches

Reviewing the literature reveals that there are commonly three broad types of approaches one can take in simulating crowds of people [4]. On a macroscopic scale, crowds can – one might find surprisingly – be modelled as flows of fluids pertaining to physical laws of fluid dynamics, and attempts have been successfully made with this approach [3].

In the Cellular Automata (CA) approach the system is split into cells of discrete states, where future cell states are determined by rules based on the states of surrounding cells [4]. For example, a very simple cell could have just two states: "a pedestrian is or is not occupying this cell". A well known example of CA, although not related to pedestrian simulation, is Conway's Game of Life, which demonstrates that complex life-like behaviour can arise from a very simple set of rules.

Lastly, the most common approach, and perhaps the first one that would come to mind, is to model the pedestrians as particles – individual entities or "agents" – that interact with each other based on social and physical laws [4].

Because this paper focuses on the last of these approaches we will not go into detail about the other ones, but it should be noted that these other approaches exist.

3 Individual Agents Approach

3.1 The Individual Agents Approach and Why We Chose it

Our approach is to model the pedestrians as "individual agents" – entities that act on behalf of themselves and interact with other pedestrians based on social and behavioural rules. The pedestrians also pertain to physical restrictions such as to hinder them from walking through solid objects (buildings) if they were to fail to avoid them.

We chose to model the pedestrians as individuals because we are interested in the behaviour of individuals and not just, for example, the "flow" of them. We wanted to model a fairly large city area, so we could have gone for the fluid dynamics approach, but at the same time we wanted to be able to zoom in on a single street and follow individual pedestrians and see what they specifically were up to and to examine their individual behaviours. Furthermore, we are very interested in the creation of artificial intelligence and in how to create behaviour such that observed and studied phenomena spontaneously will arise. The Cellular Automata approach did not seem as appealing, as applicable, as intuitive or as flexible as programming individual entities with "behavioural modules". A fun thing about artificially intelligent entities is that you can put them in new situations and see how they react.

More specifically, our approach is that of a "social force model".

3.2 The Social Force Model and An Alternative

The social force model approach is a Newtonian mechanics inspired approach (as is the fluid dynamics one) in that it describes the pedestrians' motions as the sum of attractive and repulsive forces reflecting external influences and internal motivations [7]. In reviewing Helbing and Molnár's work, Leggett [4] describe three "essential forces" of theirs: acceleration forces as a pedestrian attempt to reach optimal speed towards its goal, repulsive forces from other pedestrians, obstacles or edges, and attraction forces between certain other pedestrians, such as friends, or "attractive" objects or locations such as window displays [4]. Legget goes on to say that Helbing has produced a social force model which has successfully demonstrated some observed phenomena such as lane formation, and applied the social force model to the simulation of building escape panic ("with impressive results").

In Mussaïd et al.'s paper however, it says that there are problems with the Newtonian-inspired approaches, that it's "becoming increasingly difficult to capture the complete range of crowd behaviours in one single model" and claims that it is problematic to model interactions of multiple individuals as a number of binary interactions, i.e. when the interactions of a group of individuals are resolved through resolving interactions between each of the pairs in isolation [7]. An alternative, as proposed by Mussaïd et al., is that for every individual, we examine its field of view and determine the distance to impact of various different walking angles, taking other individuals' velocity into account. As such, this model tries to choose an appropriate immediate path (optimal walking angle at every moment) through the environment instead of having forces determine the path.

Our model is mainly that of a social force model, but inspired by Mussaïd et al., the pedestrians also try to avoid other pedestrians and obstacles by adjusting the angle of their path based on what objects are in the immediate view and the velocity of these objects, and as such, our model is not limited to binary interactions. However, we never leave the roots of the

social force model and any behaviour affecting a pedestrian does so through adding of "forces", even if these forces were determined by a more sophisticated process than binary interactions.

3.3 The Scope of Our Simulation

The initial goal of our simulation was to be able to handle several hundred or even thousands of individuals in a city environment. The interactions between individuals and other individuals or crowds of individuals was planned to be as realistic as possible, and to ensure this we used the list of observations described in *2.2 Introduction to Crowd Simulation* as the main goals for the simulation. We also wanted to look critically at our results every step of the way in order to spot unrealistic anomalies (subjectively), i.e. the behaviours of the pedestrians should look "natural". The simulation would not contain any cars or traffic besides pedestrians due to time constraint, but it would certainly be an appropriate thing to have.

4 Implementation

We implemented our simulation in the Java programming language because of our familiarity with it and its cross-platform capabilities. Due to the scale of our implementation – and for performance reasons – we decided to use the OpenGL API for graphics rendering, which we access using an external open source and free-to-use library called The Lightweight Java Game Library (LWJGL). This library is basically a way for us to easily create a window, handle input events and use the OpenGL functionality.

This section is split up into helpful subsections regarding different aspects of implementation.

4.1 Importing Maps from OpenStreetMap.org

OpenStreetMap [8] is an online world map that is entirely collaboratively created, editable and maintained by the public – it is much like what Wikipedia is for articles. Maps are more or less available all over the world and you can zoom down to street level anywhere. Because of the process of which these maps are created and edited, the number of details often varies over different areas. The maps of inner cities, like Stockholm, are thankfully very detailed.

We wanted our simulation to model the real world and we wanted to use real world areas, namely locations in Stockholm that we have visited. From the OpenStreetMap.org website you can select any area of the world map that you wish to export and download the section as a *.osm* file. This meant that we had to implement our own *.osm* file format loader, but also that our simulator can be used – or at least extended to be used – to model almost any area of the world.

This section will give a very brief overview of the *.osm* file format and superficially how we go about loading it, more details are available on the wiki section of OpenStreetMap's web site [8].

4.1.1 OSM File Structure Overview

The *.osm* file format is structured like an XML file with tags (`<tagtype ...>`) that may have zero or more attributes (`name="value"`) and child tags. What tags are allowed and what attributes they may possess are specified by the OpenStreetMap wiki. There is a sea of different attributes that may be used to describe all the possible map features, and too many for us to interpret them all. Our limited map loader only cares about classifying things in a broad sense such as buildings, roads, pedestrian paths, etc. Because of the simple XML-like structure of the file format, however, it would not be too difficult to interpret more map features.

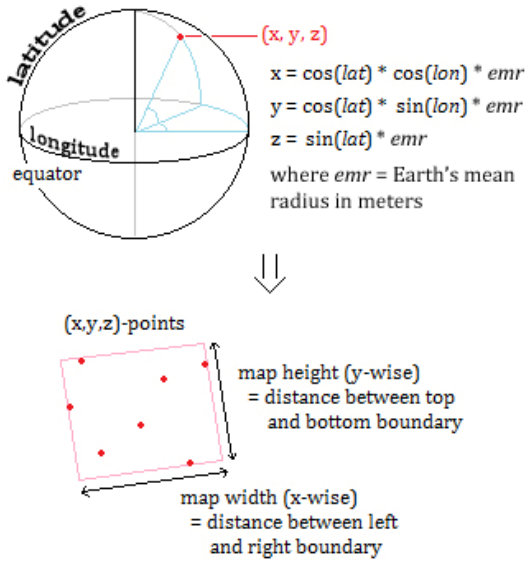
The basic tags for the basic "data primitives" of which everything of the map is defined are Nodes, Ways and Relations. How to interpret a data primitive more precisely is specified by their attributes.

- A Node represents a point on the map, i.e. a GPS coordinate.
- A Way is either a line strip or a closed polygon, and is defined by a number of Nodes.
- A Relation is any "relation" you might want to declare between one or more Nodes and/or Ways; the relation could have a functional purpose or be purely descriptive.

A building for example may be defined as a polygon (a Way) with attributes that specify that the Way is some type of building.

Interpreting the file is a matter of writing parsers and interpreters dispatching different tags, etc, to more specialized interpreters that create output to the rest of the program.

4.1.2 Converting GPS coordinates to plane coordinates, in meters



GPS coordinates are defined in terms of angles (longitude and latitude) of the globe, and are quite cumbersome for simulation purposes. The Earth is slightly uneven and not a perfect sphere, however, there is a need to convert the GPS coordinates to plane (x,y)-coordinates so that distance between coordinates can be easily measured in meters. Since we are looking at a relatively tiny piece of the world we can indeed assume that the map is a

flat plane, and also assume that the Earth is a perfect sphere, because we only really care about the scale of the map and the distances within it, i.e. the coordinates relative to each other – not their absolute position of the world (which would require a 3D model).

The GPS-coordinates are converted in two steps, first they are converted from longitude and latitude angles to approximate (x,y,z)-coordinates, then, by measuring the linear distance between the points, we get (on this scale) reliable (x,y)-coordinates. The *.osm* also contains the GPS-coordinates of the boundary's top-left and bottom-right position.

4.1.3 Map features and lack thereof

Our simulation imports buildings, roads, pedestrian paths, parks, etc. The *.osm* file format – or at least its present manifestation – does pose significant limitations on the available data. One of the most obvious absences is the lack of information about the width of roads – roads are just defined as line strips, and it says nothing about their thickness. Because of this we give our own predefined widths to the different kinds of roads; one for motorized roads and one for pedestrian paths (although in *.osm* there are a lot more road types than that). There is also no information about the location of crosswalks, so either you have to add them yourself or do without crosswalks – we went with the latter. It also lacks information about sidewalks, so we simply assume that the sides of motorized roads are sidewalks. Entrances are not explicitly defined either, however, house and street numbers are, and so based on the positions of these numbers, entrances can be created more or less where they're supposed to be. Restaurants and shops for example do exist in the data, so there is certainly the potential to use these and have the pedestrians interact with them based on personal needs or preferences. However, this is not something that we have implemented.

4.1.4 Pathfinding

It is reasonable to assume that the pedestrians know more or less how to get to where they are going in the city, and so the pedestrians mainly use pathfinding to determine their paths in the city. The nodes and roads contained within the *.osm* file can more or less directly be translated into nodes usable for pathfinding. The pathfinding algorithm we're using is A* because it's fairly efficient and reliable, the cost of a node being the combination of distance travelled from the start node and the heuristically approximated distance left to the goal.

Some buildings, particularly small houses at the outskirt of the city areas, have entrances that are not directly connected to roads. For example, there are several houses whose entrance is placed on the other side of the house compared to the road. This lead to some problems because of the path determined by the pathfinder is not directing the pedestrian to go around the house but through it. Because of the small number of cases like this we simply decided to remove entrances that do not have a road directly outside it, rather than to create more robust pathfinding, adding additional path finding nodes around the houses or more intelligent pedestrian navigation techniques.

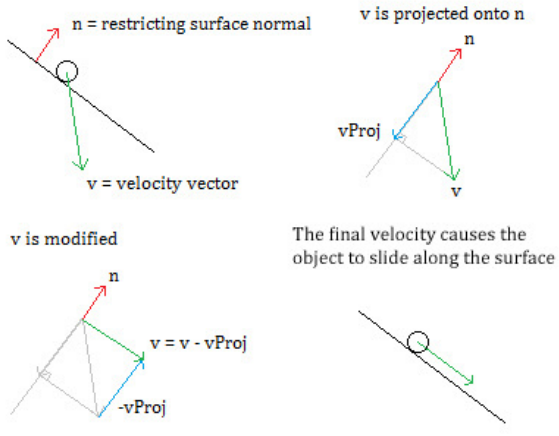
4.2 The World Structure

For performance reasons objects need to be registered to the world of the simulation in an efficient manner. Pedestrians, for example, need to be aware of nearby pedestrians and buildings in order to make behavioural decisions, and solid objects should not be penetrate-able. All of the world's objects searching through a list of all other existing objects is not an option. The data structure of which we register objects' existence and position to we refer to as the world structure, or simply "the world". Objects capable of being registered to the world are descendants of the WorldObject class, and have (amongst other things) rectangular bounding boxes.

Our world structure is implemented as a grid of cells of fixed sizes, where each cell keeps track of what objects are located in that specific cell. In this way, objects can efficiently find nearby objects by looking at what objects are located in the same or nearby cells. When an object moves it does a "refresh" call to the world to check if the cell registrations needs updating.

4.3 Solid Objects – ISolids

Behaviourally speaking, pedestrians should obviously try to avoid colliding with buildings, other pedestrians and other objects. However, since these goals are in no way guaranteed to be fulfilled, the simulated world has "physical" restrictions. Objects that should be restricting and impossible to penetrate from the outside implements an interface called ISolid – these objects are referred to as solid objects. For simplicity's sake the collision handling only bothers with circles colliding with solid objects; pedestrians are collision-wise represented as circles. The solid objects however can be any shape – buildings, for example, are closed polygons (convex or concave). Pedestrians also implement the ISolid interface so that pedestrians can't penetrate each other, which serves its purpose in dense crowds and bottleneck situations.




```
vProjLen = ( n dot v ) / |n| = { n is normalized, |n| = 1 } = n dot v

if (vProjLen < 0)
  velocity will cause penetration, continue
else
  velocity will not cause penetration, do not modify v

vProj = vProjLen * n
```

The main component of an ISolid is the *getRestrictingSurfaceNormals* method. Given a circle (center position and radius) the method returns a collection of zero or more surface normals that should be used to restrict the circle's velocity. If a circle is touching the side of a wall, the wall's *getRestrictingSurfaceNormals* method would return the normal of that wall's surface, i.e. a normalized vector perpendicular to the surface. Given this vector, adjusting the velocity of the circle is a matter of trivial vector mathematics (see the figure on the previous page). As soon as the circle stops touching the wall *getRestrictingSurfaceNormals* would return an empty collection and have no effects.

With the velocity of the pedestrians restricted like this they are guaranteed not to penetrate any other solid objects, given that tunnelling is not an issue (which in our case it isn't). When there are multiple restricting surface normals active the process described above is simply repeated for each normal, the velocity being modified in each step. However, sometimes the result of conforming to one of the normals causes the velocity to "disobey" a previously handled normal. If this happens the object should become "stuck" in that its velocity should be zero. This is achieved by going through each normal once more to make sure that none of the normals "wants to" modify the final velocity vector – if one does, the velocity is set to zero.

4.4 Pedestrians

Pedestrians are the very heart of the simulation. The Pedestrian object is a WorldObject and an ISolid and collision-wise it is described as a circle. Visually, it is represented by a circular sprite and the pedestrian itself is fairly lightweight. It's described as a circle that has a position, radius (0.25 m) and velocity, and it has an update method and a draw method. But the pedestrian also has a list of active Behaviour objects which entirely determine what the pedestrian will do for every moment – these are described in more detail in the next section. Remember that in *2.2 Introduction to Crowd Simulation*, one of the observations was "Pedestrians prefer to walk at individual walking speeds (personal preference)." Because of this, when a Pedestrian is created it is randomly given a preferred walking speed (between 0.6 and 2.35 m/s) that can be read by its behaviours.

Tying back to the social force model, the velocity of the pedestrian is never directly set by a behaviour. Instead, the velocity is determined by adding a number of "forces" together. This requires further explanation; this is not exactly like a force in the Newtonian sense and it is not an acceleration over time – a force is simply a vector used during the current update. In each update the velocity (also a vector, its components in m/s) is reset and any force added during the update call acts during that update call alone. The velocity vector is added to by the force vectors (`velocity += force`), in this sense our forces are more like impulses. If there is only a single force acting upon the pedestrian it directly translates into the final velocity, but if a second force joins in the pedestrian's velocity becomes the sum of the two forces, which may counteract, further repel, or change the direction of the pedestrian. Since the velocity is reset after each update there is not necessarily any acceleration or deceleration – this is up to the behaviours exerting the forces.

An update call handles behaviours and updates the velocity accordingly by summing up the current forces, but it also restricts the velocity. Firstly, a pedestrian should not be able to move faster than a reasonable running speed (we limit the speed to 7.5 m/s). Secondly, the pedestrian should not be able to walk through any solid objects, so nearby ISolids have their *getRestrictingSurfaceNormals* methods surveyed and the velocity is adjusted accordingly. This means that we continuously have to go through every nearby object. The world structure is appropriately implemented so that getting nearby objects is fairly effective, however, for further optimization, each pedestrian keeps an internal list of nearby objects and updates it at an appropriate interval. This list can then be shared by all behaviours that depend on nearby objects, without having to re-request nearby objects from the world structure. We should point out though that this is only marginally faster than re-requesting nearby objects from the world structure – it is, after all, an effective process. If other world structures were implemented, this might make more of a difference.

Here is a pseudocode outline of the Pedestrian's update method:

```
if (fair time since previous nearbyObjects update)
   update nearbyObjects

for each Behavior b in behaviors
   b.behave() // adds forces, etc
velocity = sum of forces
list of forces is cleared

restrictingNormals = empty list
for each ISolid s in same subsection(s) as this
   if (bounding boxes overlap)
      restrictingNormals +=
         s.getRestrictingSurfaceNormals(...)

restrict velocity according to restrictingNormals
restrict velocity to max speed

update position based on velocity and time step
notify world of updated position
```

4.5 Behaviours

Without behaviours, Pedestrians do nothing but stand still. Because behaviours are separate components, different behaviours can be implemented in isolation, without having to modify prior code, and can simply be added or removed from a pedestrian depending on what the pedestrian should be doing. Some behaviours we want to be permanent, like the avoidance behaviour – pedestrian not wanting to be too close to other pedestrians – and other are temporary, like "walk to said point on map". Anything you can see a pedestrian do is the result of one or more behaviours. Behaviour is the name of the behaviour base class.

A Behaviour instance is tied to a Pedestrian instance and must be added to said pedestrian in order to be active. A behaviour affects a pedestrian using its *behave* method and is called by the pedestrian in its update method. It typically asserts a force on the pedestrian, and it may remove itself from the pedestrian when it has fulfilled its purpose. For performance reasons,

behaviours may use timers in order to act at a lower update frequency than the rest of the program; it's simply not necessary for every behaviour to do a major update 30 times a second if its *behave* method does a significant amount of work. In those cases the behaviours may, for example, assert the same force for each *behave* call in-between the comparatively expensive re-evaluation of what the force should be. These details have been discarded from the pseudocode of the behaviours presented in this report.

Given our definition of a behaviour, a behaviour can be pretty much anything and the details are specific to the specific types of Behaviour descendants. Note that spawning pedestrians with various behaviours can be "probabilistically scheduled" through the use of so called Activities. For example, one could have the Activity "going to work" defined as "during the time 7:00-9:00 there should be an average of 10 pedestrians per entrance spawning with the behaviour 'going to work'." We describe activities more later.

4.5.1 BasicAvoidanceBehaviour

This behaviour is the very simplest form of avoidance, the pedestrian simply attempts to keep distance from other objects (including other pedestrians, of course). It achieves one of the observations defined in *2.2 Introduction to Crowd Simulation*:

> Pedestrians keep certain distance from other pedestrians and from obstacles. The distance decreases as the crowd density increases or if they are in a hurry or around "attractive" places. Resting pedestrians tend to be uniformly distributed.

The force asserted is fairly weak at a distance, but grows (exponentially to a point) the closer the pedestrian is to an object, but the force never becomes overwhelmingly strong. This means that "resting" pedestrians will move away from each other and become uniformly distributed, and the same goes for a crowd of pedestrians. Due to forces cancelling each other out, and also the final force asserted by the behaviour being divided by the number of individual forces (i.e. number of nearby objects/pedestrians), a crowd of pedestrians is "slow" at spreading out, so despite the avoidance behaviour crowds can easily form, and we achieve the property that as the density increases the distance between the individual decreases.

In isolation, two pedestrians with BasicAvoidanceBehaviour are reasonably good at avoiding each other, but if a pedestrian is blindly walking straight towards another pedestrian or into a crowd, there will be collisions – especially in the case of walking into a crowd. This might sound like a problem, however, it's exactly what we want. Imagine you are trying to walk through a crowd of people; it is (mostly) up to you to navigate in order to avoid the other people. The individuals in the crowd are not "afraid" of you, they will only move a little bit. Also, if the BasicAvoidanceBehaviour were too strong it would be impossible to walk into another human being. So, the BasicAvoidanceBehaviour is the default behaviour and is mostly concerned with keeping distance from other pedestrians in absence of other influences, and is not responsible for keeping a walking pedestrian from avoiding objects in its path, that's up to the walking behaviour (WalkToPointBehaviour).

The forces of BasicAvoidanceBehaviour have been tweaked and re-tweaked based on subjective experience with the simulation – that is the justification for the formulas in the pseudocode. The update of forces is executed four times per second.

```
finalForce = previous finalForce / 2
for each WorldObject obj in agent's nearbyObjects
   if (obj is a agent in same group as this one)
      continue; //do not avoid other group members
   dist = distance between pedestrian and obj
   if (dist > 2)
      continue; //ignore objects 2 m away
   if (dist < 0.001)
      dist = 0.001 //minimum distance

   force = normalized distance vector from obj to
                                          pedestrian

   // Determine force intensity
   f = 1 + dist, ff = f * f
   forceIntensity = 0.25 / ff + 0.75 / (ff * ff)
   // Tweak
   if (number of nearbyObjects > 10)
      forceIntensity = forceIntensity * 2
      if (number of nearbyObjects > 20)
         forceIntensity = forceIntensity * 3

   force = force * forceIntensity
   finalForce = finalForce + force

finalForce = finalForce / (number of individual
                           forces / 3 rounded up)
pedestrian.addForce(finalForce);
```

4.5.2 WalkToPointBehaviour

According to Mussaïd et al., "past studies have shown that vision is the main source of information used by pedestrians to control their motion" [7].

This is the behaviour used for telling a pedestrian to walk to a given point. It is assumed that the point is reachable through a more or less, straight path, as this behaviour does no pathfinding. However, based on the "vision" of a pedestrian, the angle of the path is adjusted to proactively avoid colliding with objects (i.e. without relying on the avoidance behaviour, which is generally too weak for a walker anyway). The walking angle directly from the pedestrian to its goal point is not adjusted unless a collision is determined to be inbound some time during the next three seconds, and smaller adjustments are preferred.

With this behaviour, a pedestrian on one side of a crowd can be told to walk to the other side of the crowd, going through the crowds, "strafing" left and right, avoiding other individuals. Cutting through a crowd like this is one of the more extreme examples. More commonly WalkToPointBehaviour simply makes sure that walking pedestrians don't foolishly walk in to each other or other solid objects. WalkToPointBehaviour is updated four times per second. If a pedestrian gets "stuck" and can't move, it backs off in the opposite of the desired direction for just a quarter of a second. This causes some movement that

prevents “clumping” of pedestrians in bottleneck situations and helps them to get unstuck.

The technique for determining if there is a collision inbound in a given walking direction is fairly complicated as it does not only account for the current distances between the objects, but their velocities as well. Any object may be moving away or into a given path, so we have to appreciate the *time until impact* of a number of alternative direct paths (walking angles), accounting for the fact that all the nearby objects might be moving in any direction. In order to achieve our goal we must briefly talk about continuous collision detection, which is a complicated subject in and of itself!

The reason for not settling with only the distances to the other objects (it being easier) is that the velocity of the other pedestrians is very important. If two pedestrians, for example, are walking one in front of the other, the distance between them could be very small indeed, but that certainly doesn’t imply that one of the pedestrians needs to turn in order to avoid collision, because they could both be walking in the same direction (in a line). Accounting only for the distances would only work properly if all the other objects were immobile, which they are not.

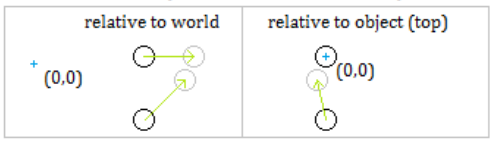
Basically, WalkToPointBehaviour should pick a walking angle and have the pedestrian walk in that direction. The default angle is always the one straight from the pedestrian to the goal point. However, if the path is – or rather *will be* – blocked, an alternative angle has to be chosen. We create a reasonable number of candidate angles (five) that make up a “cone” of angles representing the pedestrian’s field of view, kind of. Determining the time until impact of all the candidate angles (where time until impact = infinity if there will be no collisions), the final walking angle is simply one of the angles that have the greatest time until impact, preferring angles with the smallest deviation from the default angle. See figure on bottom of page.

Complicated though it might be, the process of choosing the walking angle is pretty straightforward assuming we can determine the “time until impacts”.

So, given a walking angle and a walking speed (the pedestrian’s preferred walking speed), how do we determine the time until impact, assuming all of the velocities will remain constant for the near future? We will use some techniques of *continuous collision detection*. Due to space constraints we will be very brief about this, and would like to refer to Squirrel Eiserloh’s presentation on the subject [1].

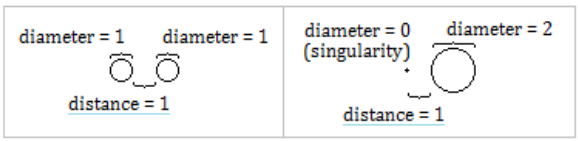
Basically, collision detection can be considered “discrete” or “continuous”. In discrete collision detection, bodies are moved at discrete time steps, say at a certain frequency, and once two bodies overlap, the collision is handled. With continuous collision detection you instead determine beforehand when the next time of impact is going to be based on position, shape and velocity. You advance in time so much and handle the collisions one at a time, without ever getting any overlaps. Do note that we do not use continuous collision detection for any other part of the simulation, or even truly for this, we only use the *techniques* of continuous collision detection in order to determine the precious time of impact between moving objects. In order to grasp how to get the time of impact there are two fundamental concepts one must first realize:

1. Relative coordinate systems. Position and velocity are relative.

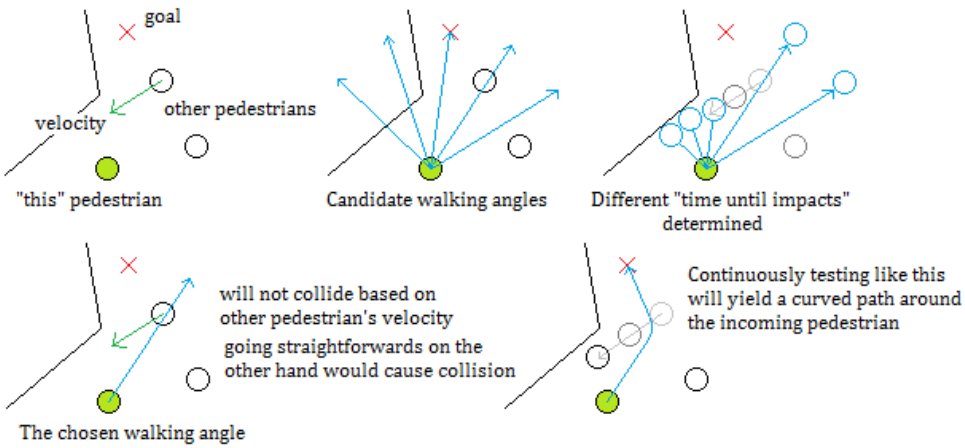


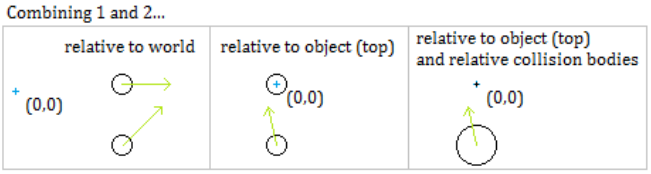
Position and velocity are relative to the current frame of reference. You can switch from the world’s frame of reference to the top circle’s frame of reference (or any other frame of reference) and get a different, but equally valid, picture of the same scene.

2. Relative collision bodies



Not only can we view the position and velocity of objects as relative to one another. Surprisingly, this can be done with shapes as well, maintaining the distance between the objects. It is easy to see how this works for simple objects like circles (see above), but the same strategy can be applied to any shapes – it’s just that the process of determining the relative shape becomes more difficult. For general shapes like polygons one would use what is called Minkowski differences, but we will not go into details about that here (see [1] instead).





No matter the situation or what shapes are involved, we end up with a singularity at the origin (0,0) and a relative shape (generally this would be some kind of polygon), and its relative velocity. By ray-casting from (0,0) and in the opposite direction of the relative velocity vector, either a hit-point is determined on the relative shape, or the ray misses. If the ray misses, the two objects will not collide and time until impact can be considered infinite. If the ray hits, we have the distance to impact (between origin and hit-point) and can hence easily determine the time until impact. Ray-casting will not be explained due to space constraints.

Thus, the time until impact can be determined for each candidate angle and the final angle is determined as described earlier. If the time to impact is greater than three seconds, we consider it infinite, because WalkToPointBehaviour is only concerned with immediate collisions.

6.5.3 FollowPathBehaviour

This behaviour is also used to make pedestrians walk to a point on the map, but not necessarily to points that are more or less straightforward from the pedestrian. FollowPathBehaviour uses the pathfinder to plan the path beforehand, and then has the pedestrian follow it. We remind the reader of an observation made in *2.2 Introduction to Crowd Simulation*:

> Pedestrians will most likely choose the fastest – and straightest – path towards their goal, even if the direct route is crowded. They refrain from taking detours or moving opposite to the desired direction.

Because the pedestrians should take the most direct route towards their goal (even if it's crowded) it is sufficient to find the shortest path through the city and follow it. The pathfinder returns a set of nodes that should be visited, one after the other. Each of these subgoals (nodes) are walked to one at a time using the previously described WalkToPointBehaviour. We will refer to a subgoal as a waypoint, and when we talk about moving a waypoint we're not talking about moving the actual node that spawned the waypoint, just the position to be visited.

Beyond simply queuing up WalkToPointBehaviours, FollowPathBehaviour keeps track of the road currently being used in the pathfinding (if any), and which the next road is going to be (if any). Based on this, waypoints are re-positioned onto the sidewalks (i.e. side of roads if motorized) so that pedestrians are not encouraged to walk in the middle of motorized roads. Waypoints are moved just before it's their turn to be walked to. Before a waypoint is moved it is ensured (through ray-casting from the pedestrian's current position) that the new position is not obstructed by a building – otherwise some waypoints would be used that could be unreachable.

The path to follow should hence not lead the pedestrian into imaginary traffic (our simulation does not actually have cars), however, at this point there is nothing that actively prevents the pedestrian from walking onto a motorized road. Because the pedestrian needs to cross certain roads, all motorized roads should not be avoided, instead, FollowPathBehaviour makes sure that the pedestrian avoids just the road currently being used, if motorized, through repulsive forces that are added if the pedestrian touches the road. Similarly, if the road is a pedestrian road, the pedestrian should prefer to walk on it, and so attractive forces are added if the pedestrian leaves the road.

4.5.4 GroupingBehaviour

The last observation made in *2.2 Introduction to Crowd Simulation* was as follows:

> Individuals who know each other form groups that may act as single entities. Loscos et al. (2003) notes that, typically, only around half of pedestrians walk alone, the rest walk in groups of varying sizes.

Because only half of pedestrians walk alone, we couldn't ignore the grouping phenomena. Thus, we've made it possible for pedestrians to form groups. People who walk together in a group obviously have the same walking speed, so whenever a pedestrian is a member of a group it uses the group's mutual preferred walking speed rather than its own. Pedestrians within the same group should not avoid each other, so if two pedestrians are in the same group, BasicAvoidanceBehaviour will ignore the other pedestrian. It is an underlying assumption that all members of a group have the same goals so that they're always walking in roughly the same direction. Given this, GroupingBehaviour makes it so that individuals that form a group 1) stay together and 2) walk side by side each other.

Given all the member's individual positions and velocities, GroupingBehaviour calculates an average position and velocity that approximately applies to the group as a whole. Using the group's direction, a perpendicular line can be drawn through the group, crossing the group's center position. Using this line, pedestrians that are behind it gets a force added so that they will "catch up" with the rest of the group, and forces are added along this line ("left" and "right" forces) to adjust the positions of the members so that they will walk side by side.

4.6 Activities

An Activity is our way to arrange for scheduling in the simulation and is a class that initiates behaviour of spawned pedestrians. Pedestrians always spawn at the entrances of buildings (that they "come out of"). An Activity "go to work", for example, could initiate the pedestrians' to have a FollowPathBehaviour that takes them to their place of work. The spawner of pedestrians is called the PedestrianSpawner, which has a list of active Activities. The PedestrianSpawner continuously fetches the probabilities that activities have for the likelihood that pedestrians should be spawned using the said activity, and as it fetches this probability, the current time is

passed as a parameter to the activity. This means that an activity could have different probabilities based on the time and day, so for example, there could be a greater chance of "going to work" in the morning than the rest of the day or on weekends. The probability we speak of is expressed as the average number of pedestrians that should come out of any one entrance per hour.

Just like Behaviours, Activities can be added in isolation without having to modify prior code, and what an Activity does or represents is entirely up to the specific implementation of IActivity. This makes it so that new scenarios can easily be simulated. For example, say there's an upcoming sport arena event that you'd like to simulate – a SportArenaActivity could be added that have a large number of pedestrians "scheduled" to spawn and go to this event.

Activities basically represents different goals that newly spawned pedestrians will have in the simulated world. A very basic activity, and the one mainly used in our simulation, is the MoveToEntranceActivity, which does exactly what the name implies. A pedestrian (or group of pedestrians) is spawned at the doorstep of a randomly selected entrance, it selects a second entrance at random from anywhere else on the map, and then initiates the pedestrian(s) to go there. Once a pedestrian initiated with this activity reaches its destination, it is indefinitely removed from the simulation (as he "enters" the building). Because "only around half of pedestrians walk alone, the rest walk in groups of varying sizes", the spawning group size of MoveToEntranceActivity is set so that 50% of spawned individuals are alone, 25% are in a group of two, and 25% are in a group of three.

This activity, in a sense, populates the world with a "living" population that seems to go about their daily lives, even though there is no underlying reason as to why they are going where they are going. One could imagine they are going to work, or visiting a friend, or going to a restaurant, etc. Because of the scale of our simulation, it's not really important why an individual is going where he or she is going, but the important thing is that every individual starts at one building and ends at another.

5 Results

In the end, the simulator was able to support a few thousands of pedestrians in an active city environment, running at 30 frames per seconds in real-time. In a 1280x1090 m^2 large area of Södermalm, Stockholm, we were able to support up to five thousand active individuals without frame-drops when zoomed-in to street level.

The spawning of pedestrians through random entrances and having them walk to other random entrances, although highly artificial, works well in making the city "come to life" with a population.

The behaviour of the pedestrians, and the avoiding of other individuals, worked particularly well for pedestrians not a member of any group, whether they were heading towards other individuals or cutting through large crowds. In common situations collision-avoidance appears "natural"; works good but not perfectly (collisions are still possible), and in cutting through the dense crowds there is some pushing aside going on.

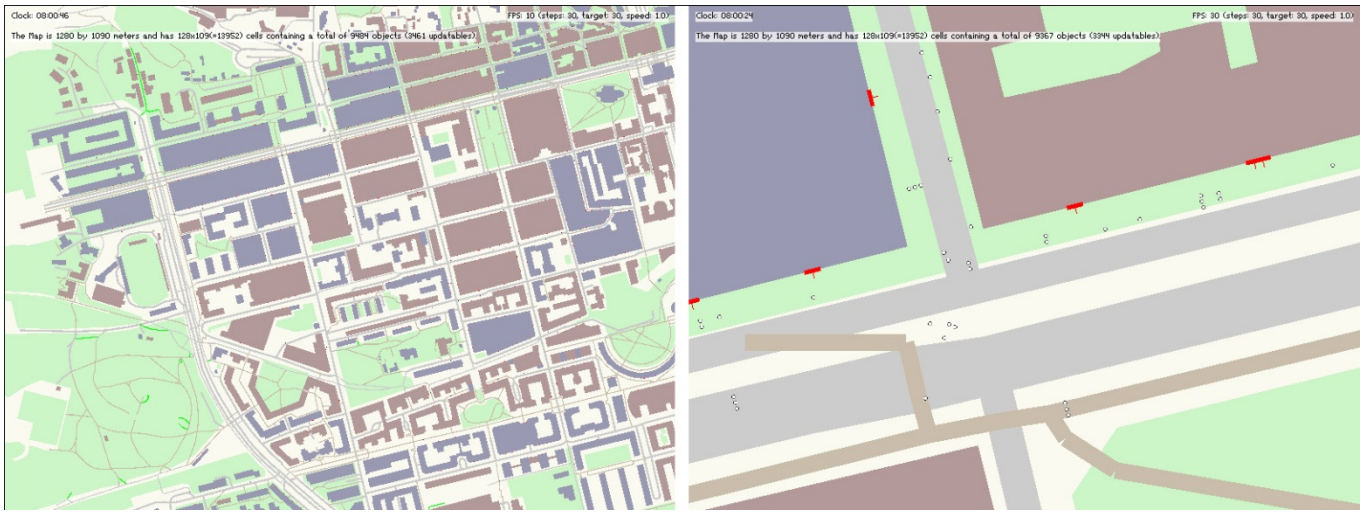
Small groups of individuals (acquaintances) are kept together and have shared goals, but each member still act as an individual entity. This results in groups of individuals sometimes being poor at avoiding other individuals or groups (especially in head-on cases), and sometimes they collide where it is unnecessary. However, each member still acting as an individual entity has the benefit that individuals can cut through a group of individuals, the members separating enough for the individual to cut through, and then they move together to walk side by side again.

We constructed a few scenes in which different scales and aspects of our simulation could be tested to get objective results in terms of number of pedestrians that could be handled with reasonable performance.

First off, we have the main scene that contains a map of Södermalm (loaded from a *.osm*) as previously described, here pedestrians are spawned by activities and walk around in the world.

The second scene is an empty scene that we fill up with a very large amount of pedestrians, all of which have the BasicAvoidanceBehaviour, stress-testing the simulation in regards to the sheer number of agents it can support.

The scene *"Empty Scene, Extreme density"* is much like the previous test scene, the difference is that in this scene every pedestrian is ordered to walk to the same point somewhere in



the middle of the scene, which tests how well our simulator handles massive densities of pedestrians before significant frame-drop. The number given for this scene is the amount that is supported as the crowd is as dense as physically possible, forming a ball of "hugging" pedestrians.

The simulation has been run with the different scenes on two household computers.

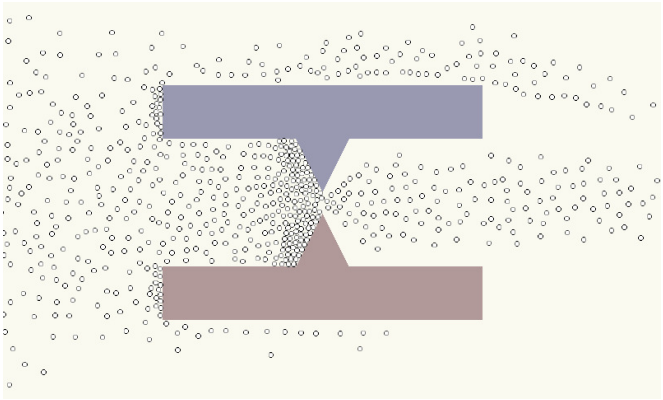
Computer	*Map Scene (Södermalm)*	*Empty Scene*	*Empty Scene Extreme density*
CPU: 2.8 GHz, 4 cores RAM: 4 GB	~5000 agents	~9000 agents	~1600 agents
CPU: 3.3 GHz, 4 cores RAM: 3.25 GB	~3000 agents	~7000 agents	~1600 agents

Note that the simulation is a single-threaded application.

There are still a few more scenarios that we tested in a near-empty scene, to examine some results unrelated to the sheer number of pedestrians.

In one scenario, pedestrians are ordered to form a fairly dense crowd, and we test how well an individual can cut through it. In another scenario two crowds are formed, and then each crowd is told to cut through the other. In both of these cases the pedestrians are fairly successful at avoiding each other, with not too many collisions or pushing about, and it looks realistic.

The perhaps most problematic testing scenario was the bottleneck test scene, in which crowds of agents are ordered to go through an hourglass-like construction (of which only a small number of individuals fit at the same time). This has been done with a single crowd walking in one direction, and also with one crowd at each end walking in opposite directions through the small bottleneck. The first case works fairly well – they get condense and start queuing as one would expect (see picture below) – but in the latter case, both groups meet in the middle and comes to a complete stand still. One observed phenomena that we did not manage to replicate was the spontaneous lane formation in big crowds of stuck pedestrians like in this case. Pedestrians from each end blocked each other, unable to move aside, and nobody got through.



Testing of WalkToPointBehaviour and BasicAvoidanceBehaviour. Note that this is not a test of the pathfinding, hence the stuck pedestrians on the structure's left.

6 Discussion

During development, we noticed a significant drop in the amount of pedestrians our simulator could manage, especially when pedestrians started to interact with nearby objects beyond collision detection and penetration prevention (from 50 thousand to the final five thousand). This is not surprising though, as when each entity goes from doing almost nothing to doing something, there is a large percentage shift in the difference of the amount of CPU each entity is going to require.

Half way into the development we also realised that, because the simulation runs in real-time, the different schedules based on the time of day (and even more so the ones based on different weekdays), is practically pointless. With different activities at different times of day you would have to run the simulation continuously for hours or even days without pause for the differences to occur. Instead, the activities we implemented were independent of the time of day.

It would have been interesting to have the pedestrians act on "wants" and "needs" such as if hungry, go to a restaurant, or pedestrians going shopping at various shops. While this was the plan throughout most of the project, it didn't seem relevant enough for the scale that we were going for, and we left it out as one of those things that "could be added in the future".

Also, as the map is directly imported from OpenStreetMaps, and the fact that pathfinding nodes are being created directly from the map's roads, problems occurred with open areas such as marketplaces or certain parks – pedestrians did not pass through these areas because the pathfinding avoided them since these areas lacked of pathfinding nodes. A more sophisticated node placer or even an altogether different pathfinding technique could possibly make these areas more accessible, and behaviours or activities could be added to make these areas attractive places to be.

When crowds became too dense with too many individuals, the simulator couldn't keep up and would freeze, and as reported in the results the number of pedestrians supported at extreme densities is significantly lower than that of a scene where the pedestrians are spread out. This is because each individual has to go through each of its nearby individuals. So the number of operations performed within a "nearby area" grows by the square of the number of "nearby individuals" (although this number is limited by the small area that is considered "nearby"). We tried to limit the number of objects considered nearby to avoid this problem, but it either didn't help a lot or caused other problems. If one truly wants the nearest x objects, a different world structure altogether might be needed. Perhaps some kind of network of objects, like a graph, rather than a grid of cells. This is probably not worth changing though unless you want to model massive amounts of individuals (thousands) that are all at extreme density. But then again, if that is your goal, then perhaps you should go with the fluid dynamics approach?

As for spontaneous lane formation not arising when pedestrians gets stuck, we believe that this is a feature that is still possible to solve with our approach, but that it would require additional or modified behaviour.

7 CONCLUSION

Fully fledged, real-time, pedestrian simulation with vast numbers of agents up to tens of thousands of individuals or more still, seems out of reach when using today's common household computers. This, however, would have been an overly optimistic goal to have. Our resulting thousands of pedestrians is still an order of magnitude greater than many of the resulting numbers found when reviewing the literature, and although other researchers' goals are not necessarily quantity, our implementation shows improvement in this area – plus, our real-time simulation runs at an impressive 30 frames per second.

Conclusively, the agent-based approach and the social force model combined with pedestrians acting on visual information is a good way to go about simulating pedestrians, both at large and small scales, and using "behavioural modules" is a good way to ease the development process by keeping different pieces of code separate.

REFERENCES

[1] Eiserloh, S., 2006. Physics for Games Programmers Tutorial – Motion and Collision – It's All Relative [PowerPoint Presentation] Available at: <http://www.eiserloh.net/gdc2006_Eiserloh_Squirrel_PhysicsTutorial.ppt> [Accessed 6 April 2012]

[2] Helbing, D., et al., 2001. *Self-organizing Pedestrian Movement*, Environment and Planning B: Planning and Design, Volume 28, (No. 3), pp.361-83.

[3] Hughes, R., 2003. *The Flow of Human Crowds*, Annual Review of Fluid Mechanics, Volume 35, pp.169-82.

[4] Leggett, R., 2004. *Real-Time Crowd Simulation: A Review*.

[5] Loscos, C., Marchal, D. And Meyer, A., *Intuitive Crowd Behaviour in Dense Urban Environments using Local Laws*, Proc.Theory and Practice of Computer Graphics 2003, IEEE ComputerSociety, 2003.

[6] *The Lord of the Rings Trilogy (Extended Edition Box Set)*. 2004 [DVD] Peter Jackson. United States: New Line Cinema. (Clip of referenced material available at: <http://www.youtube.com/watch?v=W5pNPJAhsBI>)

[7] Moussaïd, M., Helbing, D. and Theraulaz, G., 2011. *How simple rules determine pedestrian behaviour and crowd disasters*. Proceedings of the National Academy of Sciences (PNAS), Volume 108, (No. 17) [April 26, 2011], pp. 6884-8.

[8] OpenStreetMap Foundation, 2012. *OpenStreetMap Wiki* [open encyclopedia] Available at: <http://wiki.openstreetmap.org/> [Accessed 3 April 2012]

[9] Sud, A. et al., 2008. *Real-Time Path Planning in Dynamic Virtual Environments Using Multiagent Navigation Graphs*. IEEE Transactions on Visualization and Computer Graphics, Volume 14, (No. 3), pp.526-38.

[10] Thalmann, D. and Musse, S.R., 2007. *Crowd Simulation*. London: Springer