# How to run

This prototype was developed on the VSCode text editor for Windows 10 x64, using the MinGW x64 C++ compiler “g++”. The overall setup within Main features window creation followed by a purpose-built simulator implementation, built using only standard C++ libraries and these graphics libraries to display windows and simulation environments: OpenGL (3.3), GLSL, SDL2, GLM, and GLEW.

Testing was performed on a Ryzen 7 5800H CPU using an NVIDIA Geforce RTX 3060 graphics card, which was found to be sufficient to simulate in real-time for 400 particles at almost 50 epochs per second (1 second in the chosen default fine time step). Any lost simulation time when simulating 400 particles was gained back during the reduction in number throughout the simulated experiments before termination was reached.

To run this prototype the following files must be present alongside the executable within the same directory: SDL2.dll (SDL2), glew32.dll (GLEW), libgcc\_s\_dw2-1.dll (Windows), libwinpthread-1.dll (Windows), libgcc\_s\_seh-1 (Windows), and libstdc++-6 (MinGW). Also, absence of an NVIDIA GPU means visualisation within the window’s OpenGL context is not guaranteed to display the entire environment. The prototype can be executed using default settings which adds analysis data to “output.txt”, a file that is generated if not present and otherwise appended to, or can be executed with an alternative output filename as an argument via console.

To compile this prototype, the following installations must be present within the local MinGW program as well as the “g++” command being assigned as a system path variable: glew-2.1.0 (GLEW), glm-0.9.9.7 (GLM), and SDL2-2.0.16 (SDL2), all found freely online. Executing the “compile.bat” batch file automatically links all required files to produce the prototype. The required code files should include the following: “contact.h”, “debug.cpp”, “debug.h”, “generators.h”, “input.h”, “main.cpp”, “main.h”, “obstacles.cpp”, “obstacles.h”, “particles.cpp”, “particles.h”, “path.h”, “simulation.cpp”, “simulation.h”, “sph.cpp”, “sph.h”, “window.cpp”, and “window.h”.

# Design

The following simulator is later implemented into a prototype to be used in proceeding experiments. The original simulator, UMANS, was developed as a platform to compare a variety of different crowd simulation techniques. This simulator features some main components of the UMANS while incorporating the SPH crowding behaviour of the later paper.

Each movement module produces an acceleration, , , and respectively which are used in equations 1, 2 and 3 to calculate each particle’s next acceleration, velocity and position. The coarse time step, , is used for updating some values less frequently within the simulation loop; the latest calculated value is used for multiple subsequent simulation epochs. Time steps are measured in seconds and are simulated in real-time, provided the software and hardware can process each step under 1 second.

Particles are each assigned a randomised disc radius and relative mass, which represents their influence on neighbouring particles within the Contact and Hydrodynamics modules, respectively. Density relates to the area-density calculated within the Hydrodynamics module using relative mass.

Obstacle boxes generate static particles inside as uniform columns and rows of static touching particles, using static values for disc radius and mass to represent a consistent boundary for pedestrian particles to navigate around via the Hydrodynamics module. Obstacle box walls are used to repel touching pedestrian particles away from the obstacle via the Contact module.

Tables 1, 2 and 3 show default values for each simulation property of particles, obstacles and obstacle boxes respectively, except for temporary or non-scalar values, which involve spawning regions or potentially non-trivial values like goal position that heavily depend on the scenario. Most default values will be assigned during experimentation based on the original literature’s most stable or real-world-representative discovered values, to be altered during the experiments. All parameters for the simulation excluding scenario-specific parameters are summarised in table 4.

## Components

### SPH

The Hydrodynamics component is used to compute the density of particles then provide hydrodynamic forces to enhance the movement fluidity. Overall the force is found by calculating the particle’s density about all other particles representing people and obstacles, then calculating the pressure and viscosity force about the surrounding particles within the smoothing length, which is the radial influence of particles.

Equation 4 for acceleration is a particle-discretisation of a continuous field derived from the Navier Stokes partial differential equations of incompressible, viscous fluid motion and represents the overall output of this component, where is the density, is the pressure force, and is the viscosity force. Density is a particle property, pressure enacts a force pushing away from a neighbouring particle, and viscosity enacts a force pushing against the particle’s current direction of motion.

The SPH-approximation of these three terms is represented by equations 5 and 6. These equations are applied to density, pressure and viscosity to make equations 7, 8 and 9. is a smoothing kernel which usually decreases as the distance between two particles is increased, while and are it’s gradient and Laplacian. Smoothing kernels of equations 10, 11 and 12 are used for the approximation of the three terms, completing the calculation of this component’s acceleration.

Some assumptions are added to the fundamental model to ensure more acceptable crowd behaviours. Rest density is dynamically calculated for particles before calculating the pressure using a moving average of previous densities, to imitate the tendency of crowds to gradually accept higher densities using equation 13. Negative pressure is ignored to avoid drawing people to dense crowds and obstacles, done by altering the original pressure force term into equation 14. Any simulation-wide particles outside the radial influence of the smoothing length for the current particle are completely ignored when summating neighbouring particle properties for this component, to simplify overall interactions. Obstacle particles influence other particles but can only compute their density then pressure, whilst alternatively using equation 15 to compute density instead.

### Contact Forces

The Contact component is used to compute forces between touching particles, for enforcing boundaries of particles and walls. To compute the nearest point between particles and walls the “nearestPointFromLine” function is borrowed from UMANS.

The overall contact force is computed as a sum of the forces between each particle and wall using equations 16, 17 and 18, where and and are the resulting particle-particle and particle-wall contact forces respectively. is a vector to from the nearest boundary point of the wall calculated using equations 19 and 20, where is the closest point on to , and are the outer edges of , and is an intermediate value denoting the relative distance between and of the point along which is closest to .

Additional operations are performed in order of priority, to prevent deeply-penetrating pedestrians from becoming trapped inside the box or multiple walls forcing the pedestrian away from a corner doubly: Directly-adjacent contacts are prioritised to ensure the closest wall provides the contact force for the pedestrian particle; a single wall is chosen for corner contacts to ensure only one contact force is applied to the pedestrian particle; for pedestrian particles with their centre inside a wall the force equation is altered, to reverse the contact’s direction and apply the correct force, using equation 21 where the wall with the lowest contact distance is chosen, so that the contact of lowest penetration will force the particle out through the closest wall.

### Navigation Policy

The Pathing component is the final force to compute in the total acceleration equation. For simplicity the social forces collision avoidance isn’t implemented leaving only the goal force, the component responsible for veering the particles towards their goal locations.

The overall goal force is simply computed as a relaxed redirection towards the goal at a preferred speed using equation 22. All particles that reach the goal are deleted via a Sink, when the condition given in equation 23 holds true after the latest movement procedure. Finally particles may be introduced at the start of a coarse time step before the forces are calculated, by creating a new particle positioned randomly within some defined area called a Source

### Visualisation

Particles and obstacles are visualised for mid-development testing and observation. Particles are drawn as circles with a radius matching their contact range and obstacle boxes are drawn as grey blocks. Each particle is coloured dynamically based on SPH density using equations 24 and 25, which allows it to shift through blue, into green, into red, then finally into pink as the crowd becomes highly concentrated.

### Additional Components

The final components used in experimentation don’t contain complex calculations or affect the simulation so they are listed briefly here.

The input component sets all parameters and initialises the simulation before it runs.

The experimentation component periodically extracts various statistical information from the simulation useful for analysis and experimentation and is capable of varying a parameter mid-run. Statistics include updated data over a time frame including average and maximum values, and simulation data like run length and remaining contents, for model validation and experimentation.

# Implementation

The following section describes the process of converting the previously designed model into a running prototype. Components from design are converted into classes which are combined to produce a working codebase capable of performing experimentation.

The description of this study’s programmed output as a prototype owes to the premise that no further iterations were performed to improve the working program. Despite this, all functionality from the design is implemented and the program works independently apart from some DLLs used by external libraries.

## Class Structure

The original design has been represented using classes which are described during this section. The overall simulation class structure comprises of the simulation, the entities, the movement procedures, the entity generators, and the experimentation section governing inputs and outputs. These classes are listed within table 5.

Mapping of classes to entities is avoided, to ensure maximal utilisation of classes by overall purpose of the given modules rather than performing individual-element operations. For example density is calculated for all particles together, not for each particle separately, because each particle’s operation still involves every particle’s position and velocity making the calculation simpler when combined into a single function. The class structure is summarised within figure 1.

With the Simulation class at the interface between the simulation and the Main class, code is distributed within the simulation components whenever possible. This allows the Simulation class to perform flexible operations between the data structures whilst avoiding assigning all functionality to one class. Some hierarchical modelling for components like particles or movement was avoided to promote rapid development, but should be possible in the future having since produced a working prototype of adequate validity.

### Particles

The Particles class, of “particles.h” and “particles.cpp”, stores all data specific to pedestrian particles and is responsible for moving and displaying them throughout the simulation. Before the simulation is started a square region is decided for placing the particles uniformly throughout, following that shown the pseudocode function involving a custom positioning algorithm given by figure 2. This function generates a region of uniform columns and rows of particles, granted a correct configuration of parameters is given for the spawn distance between particles and the number of pedestrian particles to produce overall as to avoid overspilling or diagonal columns of particles forming.

Upon starting the simulation, particles are assigned their acceleration terms, used by the function shown in figure 3 to produce movement in uniform intervals. This procedure describes forward-Euler integration, where movement is approximated by calculating the initial acceleration to a section of time and using it to represent the change in motion for the entire time step. This creates some inaccuracy, mainly when using excessively large values or time steps, but is adequate to simulate in this instance due to the restrictions placed on the acceleration, velocity, and the small time step which avoids erratic, non-fluid movement. Table 6 describes the remaining functions of the simulation excluding graphical processing.

### Obstacles

The Obstacles class, used in “obstacles.h” and “obstacles.cpp”, stores all information and functionality of the simulation obstacles including walls, boxes, and particles generated within boxes. Initially, box-shaped regions are described within the environment, then particles are generated within the boxes using the function presented in figure 4. Given a box’s position, defined as the most immediate corner to be passed most frequently of all sides of the box during simulation, and the box’s size extending across from this point, the generated particles are packed tightly from the immediate corner to the other sides of the box with each touching their vertical and horizontal neighbours. The total number of particles is calculated in the same way, by counting the number of particles placeable in all the boxes instead of assigning them. If each dimension of the box’s size is valued to be near a multiple of the obstacle particle diameter value (two disc lengths across), particles should fill the box to all outer edges leading to the most stable behaviour within the SPH module.

The remaining function ensures both the boxes and their particles are displayed, as grey boxes and circles coloured like their pedestrian counterparts respectively. The data is set using for boxes and for particles as the simulation progresses.

### Simulation

The Simulation class, placed inside “simulation.h” and “simulation.cpp”, unites all other class components relevant to the simulation whilst allowing input and debugging to take place before and during the simulation. Miscellaneous functions include which evokes the displaying functionality from the Particle and Obstacle classes, and which builds the environment using the Input class and directs the output of Debug statistics to a given filename.

The simulation awaits each update, and following the expiration of a period will update the positions of all dynamic particles within the environment using the code in figure 5. This update function firstly updates the current time, processes the coarse time step, updates the acceleration terms for each pedestrian particle using their respective classes, moves the particles, then calculates the next coarse update time. The Debug class may also update the statistics, progress a parameter variation experiment, and terminate the simulation upon a special condition. Finally the update function specifies the fine time step, which is identical to the coarse time step except the Path acceleration term update and Debug sections are removed, since they perform at a lower rate to more interactive components like the Hydrodynamics and Contact acceleration terms.

### Hydrodynamics

The Hydrodynamics class, built inside “sph.h” and “sph.cpp”, implements the SPH section of design using the Particles and Obstacles classes to produce the SPH acceleration term for each pedestrian particle. Alongside the main functions which calculate acceleration, a function is used for initialising this class’s lists, constants, and precomputed values for the kernel functions. An and function is also used to ensure particle index consistency when called alongside the respective and functions of the Particles class.

Each simulation time step, the first function shown in figure 6 is used to calculate each pedestrian and obstacle particle’s values for density and pressure. To start, each particle’s density is calculated by summating the individual particle interactions involving that particle. Then, the particle pressure is calculated by updating the average density and the dynamic rest density. The density and pressure data is then used by the function shown in figure 7, called directly afterwards to calculate the acceleration term for pedestrian particles. In this function, the acceleration term is produced by summating each of the pressure and viscosity terms.

### Contact

The Contact class, programmed inside “contact.h”, represents the contact forces component of the designed simulation. The function assigns the input values, and calculates each box’s centre point which is used to detect particle-box collisions, before the simulation starts.

The function is used periodically to calculate the contact acceleration term for pedestrian particles, described in detail by figure 8. Initially, the contact accelerations concerning other pedestrian particles are summated, while overall contact acceleration “pressure” is calculated for the Debug class throughout the function, which represents the contact-based physical exertion on each pedestrian particle. Each box wall has their respective distances to each particle calculated for finding the contact penetration between the box and the particle, using the function from the design contact section. Cases where the particle being completely outside all walls of the box are ignored, which may cause incorrect behaviour if the particle is completely enveloped by the box, but this is unlikely to happen under otherwise stable simulation conditions. Any case in which the particle is adjacent to and outside of a wall is evaluated as the closest wall to the particle, otherwise whether the particle is outside a contacted wall is noted for later. Cases where the particle is outside the box touching a corner are then evaluated using one of the two touching wall corners the particle is in contact with, having checked that there were no outside adjacent wall contacts previously. Finally, cases involving particles inside the box are evaluated by choosing the lowest-penetrating wall contact as the repelling obstacle, to force the particle back outside the box through the closest wall.

### Pathing

The Pathing class, within “path.h”, implements a simple but effective goal acceleration term to each pedestrian particle, which pushes them in the direction of their goal. It comprises of two functions: , which assigns the input values to the simulation’s instance of the Pathing class, and , which calculates the goal accelerations as described in the design navigation policy section.

### Generators

The generators, of “generator.h”, include a Source class for generating new pedestrian particles inside the environment and a Sink class for removing them.

The Source class is instantiated using , and updates per coarse time step by decrementing a , down from a specified quantity to zero, to generate a particle within a square spawning region specified using a and , after which the counter is reset. This spawned particle may spawn partially penetrating outside the box, from the range to on each axis.

The Sink class also has a function to assign the input parameters and is used following a fine time step to de-spawn all particles with their centres inside a given circular area around the sink.

### Input

The Input class, which “input.h” implements, represents the parameterisation nexus of the overall program where all parameters are assigned to the simulation upon code compilation. There are two functions inside: , which assigns related graphics scaling and time step values for the simulation; and , which assigns each simulation class instance their respective parameters and shared structures. Any Debug experiments are also assigned and the random seed value for the simulation is planted. All functionality of this class is called before the simulation is run.

### Debug

The Debug class, implemented inside “debug.h” and “debug.cpp”, ensures all relevant statistical data is extracted using it’s functions called across the other classes. That data alongside overall simulation properties is processed, displayed, and tracked within an output file for future analysis using . gives the final simulation time, which is useful during manually-interrupted runs not terminated on a coarse time step. The function opens an output file given as input and initialises the tests, averages, and run parameters.

Some processing functions like help obtain further data for printing. is used to check for the terminating conditions described in the design section. The function is responsible for linearly altering a variable from an initial value to a final value within a given period of the simulation time, which is called periodically during each coarse time step when conducting a parameter variation experiment in the simulation.

Tracking via is done by compiling statistics together across a value, measured in times the function is accessed before average readings are computed, then displaying the final average values or otherwise on the final access before the next period. For example, the function is called every time but only the maximum result logged during the current number of accesses is displayed. For averages, individual average values for each step are summed over the then divided to get the overall average for that period.

### Main

The Main code, “main.h”, “main.cpp”, “window.h”, and “window.cpp”, is responsible for initialising a window and graphical context. The window is given primitive functionality to , and it’s display. When updating, the window sleeps to enforce a maximum frame rate of 144 frames-per-second. Without pre-processing this leads to the simulation never reaching this frame rate, but the drop in frames appears insubstantial compared to a simulation step’s processing time. The main code uses to retrieve window events, the functional ones being to resize and close the window, and runs a loop for the simulation to update over time. After the simulation ends all classes call their function to free their dynamically-allocated structures through the class hierarchy, starting in Main.

# Experimentation

Within this section, the experimental components from the implementation are described in detail, then the model is validated using a base case and some exploratory experiments are defined to be used for producing the results found in the following section.

## Parameters Used

The Input component allows scenario-specific parameters to be altered, including the placement of pedestrians and obstacles as well as the overall map size. The Debug component allows tracking of useful metrics and linear variation of a parameter, for testing hypotheses and validating the model.

For each scenario a table will state what parameters have been altered from default and base case values, and also any deeper alterations. The remaining initialisation parameters to be completed from the implementation model and relating to the scenario include those mentioned within table 7.

There will also be a table for showing tracked metrics and their significance to the current experiment. Experimental and tracked parameters are given in table 8.

## Validation

To ensure correctness and accuracy of the implementation, as well as observe simulation outputs, the Debug module tracks the metrics shown in table 9 over time for analysis.

Of these metrics, and should describe pedestrian throughput, , , and should describe pedestrian strain or crush, should describe stagnation of movement within the simulation, and should describe the level of enforcement of the contact force rules within the simulation, respectively. These metrics will be graphically presented as interesting results emerge from the experiments.

Visualisation may also elude to problems in stability; jittering will appear as uneven densities and particles moving back and forth frequently, while particles which are tightly packed or spread thin will give knowledge of any mismatching behaviours to what was expected or encouraged through changes in parameters. Realistically particles should never pack tighter than their radii, as clearly demonstrated in the base case, unless the resulting simulation is portraying crowd congestion with a subsequent slowdown in evacuation rate.

To ensure consistency and higher quality of results, 10 simulation runs are analysed for each experiment with each run using unique pseudo-random seeds. Termination is defined by the following rules:

* If no source is spawning and no more particles are present, terminate due to completion.
* If the maximum particle velocity is less than 1mm, terminate due to deadlock.

Due to the highly dynamic nature of the crowd simulation metrics throughout the run, and the constraint of time to experiment, sensitivity analyses will be omitted for parameter variation experiments focusing on the flow of the simulation through time while making brief cross-experiment comparisons.

## Base Case

The doorway scenario is a suitable test for validating movement as it contains all major components of pedestrian movement within crowds: navigation through crowds, navigation past obstacles, bottlenecked crowd behaviour, and a controlled, evenly-distributed population of the crowd scattered within a simple, unobstructed area.

Pedestrians are arranged within a 20m x 20m square room spread in a grid with 1m distance between each particle centre. A doorway 80cm across and 1m deep is to the right spanning the whole right side of the room, with the pedestrian goal spanning 1m further out from the opposite side of the doorway. This makes room for 2 columns of obstacle particles within the 2 boxes beside the doorway, giving ample opportunity to test the effectiveness of obstacle-based repulsion forces and their effect on preventing pedestrian-obstacle wall boundary penetration. This scenario is summarised within table 10. The chosen simulation and experimentation parameters are given in table 11, with chosen values taken from their originating literature where possible.

To validate the implementation, the standard metrics are investigated using the base case to evaluate simulation stability and general behaviour. The results of executing 10 base cases are displayed using time-plot graphs in the appendix, where the tracked variables are described in table 12. These graphs prove an adequate stability of the simulation which produces relatively consistent results between runs, proved by the similarity of data between simulation runs. Simulation limits are found to be usually sufficient, like the hard-limit on speed being avoided on average or the wall penetration mostly keeping pedestrian particle centres from entering the walls of the doorway. Frame rates stay around 50 steps per second for the 400 particles of the base case, which narrowly allows the simulation to run at real time using the hardware and software defined in the “How to run” section.

As well as tracking metrics the simulation was visually compared to the UMANS implementation, which was accompanied by video material[[1]](#footnote-1) to demonstrate it’s features and improvements. By viewing each simulator with matching parameters, from the contact forces and SPH + contact forces sections, each one is able to produce equal behavioural effects including perturbation at the doorway due to excess contact forces, circular crowding around the doorway, and orderly, controlled evacuation following SPH involvement. Using density visualisation alike to the original, the new prototype appears to effectively demonstrate the improvement in parametric control and density consistency found in the original simulator, shown in figure 9.

## Experiments

Here are the parameter variations used to represent crowd health changes so that the experiment results parallel the emergent behaviour of crowd health disruption, involving variation of the maximum rest density, SPH gas constant, preferred speed, and goal force strength. These four experiments are presented in tables 13, 14, 15 and 16, respectively. The results of these experiments are presented as time plot graphs in the appendix.

# Appendices

## Figures

1: Implementation Class Structure

Diagram

Description automatically generated

2: Particles Class Set Function

Text, letter

Description automatically generated

3: Particles Class Move Function

Text

Description automatically generated

4: Obstacles Class Set Function

Text

Description automatically generated

5: Simulation Class Update Function

Text

Description automatically generated with medium confidence

Text, letter

Description automatically generated

6: Hydrodynamics Class UpdateDensity Function

Text, letter

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

7: Hydrodynamics Class UpdateAcceleration Function

Text, letter

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

8: Contact Class Update Function

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Text, letter

Description automatically generated

A picture containing text

Description automatically generated

9: Pictures of the original UMANS SPH model, with the study prototype beside it, performing the same experiment.

A picture containing shape

Description automatically generatedBackground pattern

Description automatically generated with medium confidence

## Tables

### Model Design

1: Particles Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Symbol | Default Value | Description |
| Simulation |  |  | Position |
|  |  | Velocity |
|  |  | Acceleration |
|  |  | Disc Radius |
|  |  | Mass |
| SPH |  |  | Density |
|  |  | Pressure |
|  |  | Average Density |
|  |  | Dynamic Rest Density |
|  |  | Acceleration Term |
| Contact Forces |  |  | Acceleration Term |
| Navigation Policy |  |  | Goal Position |
|  |  | Acceleration Term |

2: Obstacles Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Symbol | Default Value | Description |
| Simulation |  |  | Position |
|  |  | Disc Radius |
|  |  | Mass |
| SPH |  |  | Density |
|  |  | Pressure |
|  |  | Average Density |
|  |  | Dynamic Rest Density |

3: Obstacle Box Data Structure

|  |  |  |
| --- | --- | --- |
| Symbol | Default Value | Description |
|  |  | Position |
|  |  | Size |

4: Model Parameters

|  |  |  |
| --- | --- | --- |
| Component | Symbol | Description |
| Simulation |  | Fine Time Step |
|  | Coarse Time Step |
|  | Maximum Particle Speed |
|  | Maximum Particle Acceleration |
| SPH |  | External Forces |
|  | Smoothing Length |
|  | Ideal Gas Constant |
|  | Viscosity |
|  | Minimum Rest Density |
|  | Maximum Rest Density |
|  | Dynamic Rest Density Time Window |
| Contact Forces |  | Neighbour Particle Contact Force Strength |
|  | Neighbour Wall Contact Force Strength |
| Navigation Policy |  | Goal Force Strength |
|  | Preferred Particle Journey Speed |
|  | Particle Speed Relaxation Time |
| The remaining components use ticks or epochs-until-next-update in place of simulation time. Their parameters aren’t shown here since they don’t directly affect the running of the simulation. | | |

### Simulator Design

5: Simulator Classes

|  |  |  |
| --- | --- | --- |
| Class | Corresponding Component | Description |
| Main |  | Initialise and update the main loop |
| Window |  | Display the simulation |
| Simulation | Simulation | Initialise and update the simulation |
| Hydrodynamics | SPH | Update particle densities and movement |
| Contact | Contact Forces | Update particles upon collision |
| Pathing | Navigation Policy | Update particles’ desired directions |
| Source | Source | Add new particles to simulation |
| Sink | Sink | Remove particles from simulation when at destination |
| Input | Input | Initialise simulation and component parameters |
| Debug | Experimentation | Perform static and dynamic analysis, vary parameters |

6: Particles Class Remaining Functions

|  |  |
| --- | --- |
| Function Name | Description |
|  | Particles display themselves using the colour scheme mentioned in the design visualisation section, based on their current SPH density as done in the original UMANS implementation. This data is processed using which fills the buffer with all particle information ready for . |
|  | Particles can be introduced mid-run similar to the function while assigning random attributes to a particle at a given position. |
|  | Remove particles mid-run by index from the simulation using this function. |

### Experimentation

7: Scenario Parameters

|  |  |  |
| --- | --- | --- |
| Component | Symbol | Description |
| Particles |  | Initial Number of Particles |
|  | Maximum Number of Particles |
|  | Spawn Area Position |
|  | Spawn Area Size |
|  | Spawn Distance Between Particles |
|  | Default Goal Position |
| Obstacles |  | Limit of Obstacle Particles to Fill Boxes (optional, maximum by default) |
|  | Total Number of Obstacle Boxes |
|  | Total Number of No-Particle Boxes |
|  | Box Positions |
|  | Box Sizes (negative values: reversed direction to align particles) |
| Generators |  | Source Position |
|  | Source Size |
|  | Source Tick Spawn Delay (negative signifies no source spawning) |
|  | Sink Position |
|  | Sink Radius |

8: Experimental Variables

|  |  |  |
| --- | --- | --- |
| Component | Symbol | Description |
| Tracked Value |  | Number of Pedestrian Particles Currently Within the Simulation |
|  | Average Pedestrian Particle Density |
|  | Average Pedestrian Particle Speed |
|  | Maximum Pedestrian Particle Speed |
|  | Maximum Obstacle Wall Boundary Penetration |
|  | Average Sum of Contact-Based Acceleration Affecting Pedestrians |
| Overall Value |  | Final Simulation Running Time |
| Parameter Variation |  | Variation Time Start |
|  | Variation Time Period |
|  | Variable |
|  | Initial Variable Value |
|  | Final Variable Value (after variation period) |

9: Tracked Variables

|  |  |  |
| --- | --- | --- |
| Symbol | Description | Desired Result |
|  | Total Pedestrian Particles | Consistent change with static parameters |
|  | Average Pedestrian Particle Density | Tends towards maximum rest density |
|  | Average Pedestrian Particle Velocity | Tapers then rises as crowd dissipates |
|  | Maximum Particle Speed | Never zero unless exit is blocked |
|  | Average Contact Acceleration Sum | Directly correlates to maximum rest density |
|  | Maximum Wall Penetration | Minimal in length |
|  | Final Simulation Time | Has a small deviation |

10: Base Case Model Parameter Values

|  |  |  |
| --- | --- | --- |
| Component | Symbol | Value |
| Particles |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Obstacles |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Generators |  |  |
|  |  |
|  |  |
|  |  |
|  |  |

11: Base Case Scenario Parameter Values

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Symbol | Default Value | Description |
| Simulation |  |  | Fine Time Step |
|  |  | Coarse Time Step |
|  |  | Maximum Particle Speed |
|  |  | Maximum Particle Acceleration |
| SPH |  |  | External Forces |
|  |  | Smoothing Length |
|  |  | Ideal Gas Constant |
|  |  | Viscosity |
|  |  | Minimum Rest Density |
|  |  | Maximum Rest Density |
|  |  | Dynamic Rest Density Time Window |
| Contact Forces |  |  | Neighbour Particle Contact Force Strength |
|  |  | Neighbour Wall Contact Force Strength |
| Navigation Policy |  |  | Goal Force Strength |
|  |  | Preferred Particle Journey Speed |
|  |  | Particle Speed Relaxation Time |
| Debug |  | 10 | Coarse time steps between each statistics update |

12: Tracked Variable Expected Behaviours

|  |  |
| --- | --- |
| Metric | Result |
|  | Average density remains close to the maximum rest density, until most pedestrians are evacuated |
|  | Average Speed remains below the preferred speed, and keeps above zero as pedestrians congest the doorway. Initially, pedestrians move freely towards the doorway, then slowly accustom to the denser crowd, finally moving freely again when exiting through the doorway. |
|  | Penetration distance usually keeps below the pedestrians' disc radii, meaning misdirected repulsions are minimised. |
|  | Average contact acceleration consistently rises as pedestrians evacuate, correlating inversely with pedestrian total count. |
|  | Maximum pedestrian speed rises above desired speed, until there are fewer pedestrians to push towards the doorway. Speed never reaches zero, signifying no deadlocks have occurred using the default settings incorporating SPH. |
|  | Pedestrian count gradually increases in rate of decrease due to absence of congestion. Average rate of pedestrian evacuation is around 3.4 per second, which appears to characterise this evacuation case as hurried, or slightly agitated. |

13: Experiment 1

|  |  |  |
| --- | --- | --- |
| Maximum Rest Density Experiment | | |
| This experiment will explore a rise in crowd panic represented by maximum rest density, while tracking the success of keeping the current average density similar to the varied value. The initial and final value represents a calm and panicked crowd respectively, and the variation start time is the previously-observed stabilisation time of the average pedestrian density for the default-settings base case scenario. The overall contact acceleration and average speed will also be observed to identify a suitable threshold before crowd congestion becomes hazardous. | | |
| Component | Symbol | Value / Desired Result |
| Parameter Variation |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Tracked Value |  | Rise with |
|  | Rise with |
|  | Rise with |

14: Experiment 2

|  |  |  |
| --- | --- | --- |
| SPH Gas Constant Experiment | | |
| This experiment will explore the impact of affecting the SPH parameters to generate crowd disorder. When 0 the gas constant effectively disables the SPH module, allowing contact forces to take control. The original paper from van Toll et. al. (2020) used heightened contact forces in the absence of SPH to achieve stable, albeit jittery and cramped, results. The contact forces will be kept weak here to observe the impact on simulation stability, and to find any suitability for representing crowd panic in this manner. | | |
| Component | Symbol | Value / Desired Result |
| Simulation |  |  |
| Parameter Variation |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Tracked Value |  | Lower as decreases |
|  | Maintain reasonable value |
|  | Maintain realistic gradient |
|  | Maintain realistic gradient |

15: Experiment 3

|  |  |  |
| --- | --- | --- |
| Preferred Speed Experiment | | |
| Factors of exit speed include preferred speed of the navigation policy. A higher value translates to a larger acceleration applied in the general direction of the pedestrian’s goal for pedestrians under their desired speed, which happens often. Thus increasing the acceleration strength by raising this value could lead to a faster exit time, granted the stability of the simulation is at risk. Whether density and contact-based interactions retain their quality and react correctly to changes in speed will determine this test’s success. The experiment will overall investigate the relation between preferred speed, simulation stability, and panicked behaviour within the crowd. | | |
| Component | Symbol | Value / Desired Result |
| Parameter Variation |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Tracked Value |  | Rise with |
|  | Maintain realistic gradient |
|  | Maintain reasonable value |
|  | Maintain realistic gradient |
|  | Maintain realistic gradient |

16: Experiment 4

|  |  |  |
| --- | --- | --- |
| Goal Force Strength Experiment | | |
| Goal force is originally used to target scenario-specific requirements for either crowd gatherings into open areas or directed crowd navigation past obstacles. This experiment will attempt to repurpose this parameter for representing the drive to escape a crowd; a characteristic of crowd panic. As opposed to altering preferred speed the goal force strength parameter will avoid increasing the maximum speed of the crowd, hopefully retaining the quality of feedback within a more controlled and stable test despite the presentation of panic being tamer. | | |
| Component | Symbol | Value / Desired Result |
| Parameter Variation |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Tracked Value |  | Rise with |
|  | Maintain reasonable value |
|  | Maintain realistic gradient |
|  | Maintain realistic gradient |

## Equations

Equations given throughout the design provide the majority of functionality within the simulation, which are summarised here:

Simulation

1. Acceleration Update

2. Velocity Update

3. Position Update

SPH

4. Acceleration Term

5. Smoothed Approximation

6. Approximation Derivatives

7. Density Approximation Term

8. Pressure Approximation Term

9. Viscosity Approximation Term

10. Density Smoothing Kernel “Poly6”

11. Pressure Smoothing Kernel “spiky”

12. Viscosity Smoothing Kernel “Müller”

13. Dynamic Particle Density

14. Non-Negative Pressure Term

15. Obstacle Particle Density Term

Contact Forces

16. Acceleration Term

17. Particle Contact Force

18. Obstacle Wall Contact Force

19. Nearest Point On Wall

20. Relative Wall Point Position

21. Inter-Obstacle Contact Force

Navigation Policy

22. Acceleration Term

23. Sink Removal Condition

Visualisation

24. Particle Colour

25. Colour Density

## Graphs

### Base Case

### Experiment 1

### Experiment 2

### Experiment 3

### Experiment 4

1. Video material of the extreme-density simulation from van Toll’s 2020 paper, involving SPH and Contact Forces: <https://www.youtube.com/watch?v=f-4JrPq-jn4> [↑](#footnote-ref-1)