A Sketch-based Interface for Real-time Control of Crowd Simulations that incorporate Dynamic Knowledge

Luis Rene Montana Gonzalez, Steve Maddock
Department of Computer Science, University of Sheffield, Sheffield, UK

Abstract

Controlling crowd simulations typically involves tweaking complex parameter sets to attempt to reach a desired outcome, which can be unintuitive for non-technical users. This paper presents an approach to control pedestrian simulations in real time via sketching. Users are able to create entrances/exits, barriers to block paths, flow lines to guide pedestrians, waypoint areas, and storyboards to specify the journeys of crowd subgroups. Additionally, a timeline interface can be used to control when simulation events occur. The sketching approach is supported by a tiled navigation mesh (navmesh), based on the open source tool RECAST, to support pedestrian navigation. The navmesh is updated in real time based on the user's sketches and the simulation updates accordingly. A comparison between our navmesh approach and the more often used grid-based navigation approach is given, showing that the navmesh approach scales better for large environments. The paper also presents possible solutions to address the question of when pedestrians should react to real-time changes to the environment, whether or not these changes are in their field of vision. The effectiveness of the system is demonstrated with a set of scenarios and a practical application which make use of a 3D model of an area of a UK city centre created using data from OPENSTREETMAP.

First presented at the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications VISIGRAPP 2019, extended and revised for JVRB

Keywords: Sketch-based interface, dynamic knowledge, crowd simulation, multiagent system, navigation mesh.

1 Introduction

Crowds are present in quotidian activities such as walking to work, taking a train, shopping and attending a football match, and are simulated for numerous computer applications such as in entertainment, urban planning, safety and military training. Such crowd simulations are commonly modelled using an agent-based approach. However, controlling such simulations can be challenging, since many parameters must be tuned by the user. With commercial systems such as SIMWALK[1], MASSMOTION[2], MASSIVE[3], LEGION[4] and EXODUS [GOL96][5], the user needs to be familiarised with the system and must know which parameters to change to obtain a desired outcome. Editing the environment can also be an issue. For instance, to create an obstacle such as a barrier in MASSMOTION, the simulation must be terminated and started again after adding the obstacle.

This paper presents a real-time, sketch-based control approach, where non-expert users are able to interact with the simulation by creating entrances/exits to define the spawning and goal positions for pedestrians, barriers to block paths, flow lines to guide pedestrians, waypoint areas, and storyboards to control the journey of groups or the entire crowd. In addition, users can simulate events through the day us-

[1] `http://www.simwalk.com`
[2] `http://www.oasys-software.com/products/engineering/massmotion.html`
[3] `http://www.massivesoftware.com/`
[4] `http://www.legion.com/legion-software`
[5] `http://fseg.gre.ac.uk/exodus/index.html`

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

ing a timeline interface. We use a microscopic modelling approach based on agents and a tiled navigation mesh (navmesh [Sno00]) as the navigation approach to guide the agents through the environment.

Modifying the environment in real time using sketching raises the question of when an agent should become aware of the changes. Should it be when the agent sees the change, or when the agent hears about the change through some communication medium, e.g. an announcement on the radio, or after a certain amount of time has elapsed and the change has become common knowledge? This question appears to be unanswered in previous research work. Our paper addresses this issue by giving agents different types of knowledge of the environment for a particular scenario and discussing the performance implications.

Sketching approaches to control crowd simulations have been presented before [JXW+08, OO09, PvdBC+11, HOC14, GM17]. However, this paper presents six novel contributions. First, sketching is used to update a navmesh in real time, rather than using a grid-based approach. This includes the ability to draw barriers, unlike previous work where a list of points was used to add an obstacle to a navmesh [Kal05], which is less intuitive for the user. Second, flow lines can be sketched and the cost of traversing each flow line can be individually changed. Third, areas can be sketched onto an environment, similar to [HOC14], but with explicit control being given over the percentage of agents visiting each (waypoint) area. Fourth, storyboards can be created to define the journeys of subcrowds. Fifth, a timeline interface can be used to control events during a simulation of a 24-hour period. Last, we identify the issue of agents becoming aware of and reacting to dynamic environments and propose solutions.

The remainder of the paper is organised as follows. Section 2 gives an overview of related work and offers a classification of the existing graphical approaches to control pedestrian simulations. Section 3 describes the implementation of the system. Section 4 presents a range of scenarios and a practical application. Section 5 gives a detailed comparison between a grid-based approach and our navmesh-based approach. Section 6 discusses when the agents should become aware of the environment modifications. Finally, Section 7 concludes the paper.

2 Related Work

Agent-based modelling is the most common approach to simulate virtual crowds. Each agent calculates its own movement based on a set of rules or behaviours. Reynolds's seminal work [Rey87] demonstrated how to control a bird flock with three simple rules, and his later work [Rey99, Rey00] implemented new steering behaviours such as seek and pursuit. Other work has shown how the movement of agents could be determined by 'social forces' produced by agent-agent interaction and agent-environment interaction [Hel91, HM95]. More recent work has shown how psychological aspects can be taken into account when modelling agents [SBPO05, PAB07, YCP+08, UT01, UT02, RCL+11]. Sociological factors have also been modelled [MT97, PHDL07, YT07].

Two levels of control can be observed in pedestrian simulations. Local motion defines the short range motion of the agents considering their immediate surroundings. This level relates to the previously mentioned work using rules, behaviours and other factors. Global navigation guides the agents through environments where pedestrians could get stuck in local minima when using local rules. Several techniques have been proposed for global path planning. These methods include flow fields and navigation meshes. Reynolds [Rey99] was the first to propose the idea of flow fields. The environment is mapped to a two-dimensional grid where each cell contains a force vector. This is a grid-based approach. In our paper, we use a navigation mesh. Our sketch-based interface for graphical control of the simulation is implemented on top of this.

2.1 Navigation Mesh (Navmesh)

[Sno00] introduced the term navigation mesh, a decomposition of a 3D environment into a mesh of convex polygons to represent walkable areas. He called this a navmesh, Several techniques have been proposed to generate navmeshes from the geometry of the environment. [vTCG11] employed a medial axis to create a navmesh for a multilayered environment, and then extended this work to support dynamic updates [vTCG12]. [KBT04] represented a 2D environment using a constrained Delaunay triangulation. In subsequent work, [Kal05] added obstacles in real time. [HYD08] created a 2D navmesh by dividing the environment into a grid. Square regions are seeded and grown in every direction until obstacles are found to

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

form the polygons. This work was later extended to work with 3D environments [HY09]. [OP11] represented the environment as a single polygon that may have holes in it. Their later work used GPUs to automatically generate a navmesh from a 3D environment [OP13]. [AG13] proposed an approach to create a navmesh based on images. [BKF16] created a navmesh based on the curvature of the original mesh. There is also an open source software to create a navmesh: RECAST is used in video games to generate navmeshes for a given environment. Section 3.1 will give more details about the use of RECAST in this paper.

2.2 Graphical Control

Graphical tools for the control of crowd simulations make it possible for a user to interact with a simulation in an intuitive way, eliminating the time-consuming task of parameter tuning. We identify five categories to describe the different graphical control approaches: Navigation Graph, Map, Patch, Direct Interaction and Sketching. A visual illustration of the different approaches is given in Figure 1.

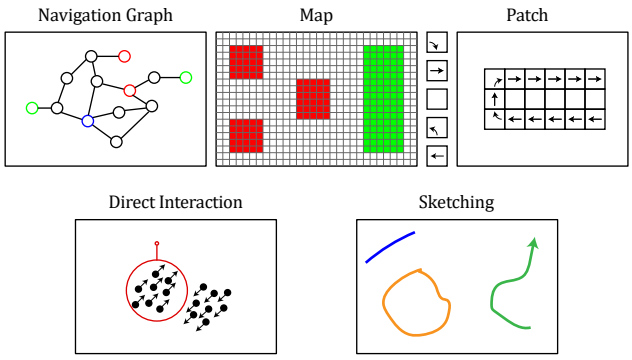


Figure 1: Graphical control approaches for crowd simulations.

In the first category, *Navigation Graph*, a graphical interface is used to manipulate graphs to control crowd movement. [YMDHC+05] created a graph from a predefined environment and allowed the user to assign nodes as goals for the pedestrians.

The second category, the *Map* approach, attaches extra information to the environment by drawing maps on top of it. Agents use this information to influence their behaviour. [SGC04] added 'situations' to the environment using a painting interface. [MR05] set environmental attributes such as height using maps. Similarly, [MMHR16] defined areas such as obstacles and exits with a painting tool. [JCC+15] specified the crowd density and direction by drawing maps on top of the environment.

In the third category, *Patch*, large and complex environments are created by connecting small predefined patches or blocks. [Che04] presented *Flow Tiles*, which are small blocks with predefined forces. These tiles are connected to move agents around the environment. [YMPT09] developed *Crowd Patches* with flows and animation attached. This work was extended by [JPCC14] where the patches could be deformed or combined to fit the environment. Another approach called 'motion patches' [LCL06] includes motion data to animate characters within the patch. This concept was used by [KHHL12] to model a simulation with characters interacting with each other.

In the fourth category, *Direct Interaction*, a crowd is directly controlled to change its behaviour – the agents themselves are directly targeted. [UCT04] create and modify agents and their behaviour using brush tools. [KLLT08] created a graph from an existing animation of characters, which is then user-deformable to create a new animation. A similar method was presented by [KSKL14]. Here, the existing animation can be manipulated in space and time. Furthermore, [KHKL09] permits the user to change the position and direction of a group of characters with spatial and temporal constraints. Different input approaches have also been experimented with [HSK12, SHW+18].

The final category includes work that interacts with the simulation by *Sketching*. This could be by modifying the environment or by controlling the path or actions of a group. [JXW+08] controlled pedestrian movement by drawing arrows in the environment. These sketches update the underlying vector field responsible for guiding the agents. [OO09] allowed the user to specify the spawning location and trajectory of pedestrians by sketching lines. [PvdBC+11] created a navigation field to direct the crowd based on flow lines drawn by the user.

[HOC14] presented a sketch-based approach to populate environments initially based on an image. These environments cannot be used in automatic navigation mesh generation tools. Thus, the user first defines the boundaries of the navigation mesh and the borders of the obstacles (e.g. buildings) in an offline process using sketching. The mesh is triangulated to obtain a navigation graph. Then, users are able to dynamically use sketching to add waypoints, select pedestrians, create a path, and define behaviour areas where

agents perform a certain action. This work is the only approach that uses sketching on top of a navmesh. However, unlike our paper, the navmesh is not updated in real time based on user input. In our work, the navmesh itself is changed in response to sketches that change the environment, such as sketching barriers and flow lines. [GM17] developed an interface where users are able to specify the agent spawning position and draw obstacles and flow lines to control the simulation in real time. The underlying grid-based navigation is updated according to the user's actions.

The creation of crowd formations is a popular application of sketching interfaces. [TYK+09] controlled formations taking into account agent adjacency. [GD11] define a formation with user-sketched lines. This work was subsequently extended to give more options to specify the formation and to allow the sketching of trajectories [GD13]. Similarly, [APKM15] create and move formations with sketches but offer the extra feature of subgroup control. [XJY+08] presented a flock simulation constrained by a user-defined shape. The user can define fixed positions over time and the path followed by the flock. In [XWY12], the user specifies the source and target formations. [XWY+15] created a similar crowd formation transformation model but using different criteria to assign the final positions. Subgroups are formed to maintain the cohesion of the group. [ZZC+14] suggested a method to create formations based on geometry which does not require collision avoidance algorithms. A similar approach was presented by [ZLD15]. The user is able to specify the density of the formation and the final formation by importing an image or by sketching the shape. [HAMB+14] proposed a flocking algorithm to represent user-defined shapes within a robot swarm. A drawing interface is provided where static shapes or animations can be made by the user.

The approach implemented in this work falls into the Sketching category, using a navmesh as the environment discretisation method and controlling the agent behaviour by modifying the environment. Table 1 summarises the relevant research.

Category	Control	Discretisation	Work
NavGraph	Env	Graph	[YMDHC+05]
Maps/Direct	A/Env	not stated	[SGC04]
Maps	Env	Grid	[MR05, MMHR16]
Maps	Env	Graph	[JCC+15]
Patches	Env	Grid	[Che04, LCL06]
Patches	Env	Graph	[YMPT09, JPCC14]
Patches	Env	not stated	[KHHL12]
Direct	Agent	not stated	[UCT04, KLLT08, KSKL14, KHKL09]
Direct	Agent	Grid	[HSK12]
Direct	A/Env	not stated	[SHW+18]
Sketching	Env	Vector field	[JXW+08]
Sketching	Env	Grid	[PvdBC+11, GM17]
Sketching	Env	Navmesh	[HOC14, GM19]
Sketching	Agent	not stated	[OO09, TYK+09, GD11, GD13, APKM15, XJY+08, XWY12, XWY+15, ZZC+14, ZLD15, HAMB+14]

Table 1: Summary of the graphical control approaches for crowd simulations. The Control column indicates whether the agent behaviour is controlled by changing agent (A) parameters and/or by modifying the environment (Env). The Discretisation column indicates how the environment is represented: Grid, Navmesh or Graph (where Graph includes techniques that use a graph structure based on circles or polygons).

3 The System

This paper presents an intuitive and simple way for non-technical users to interact in real time with crowd simulations. Figure 2 gives an overview of the entire system. In practice, we need to make decisions about particular pieces of software to create the system. Figure 3 gives an overview of these. There are two main modules: a visualisation module created by making use of UNREAL[6] and RECAST[7], and a simulation module based on the FLAMEGPU framework [RR08][8], which handles the agent and behaviour specifications. Additional open tools are used to create the 3D model of the environment.

The main modules communicate with each other through a CPU-based shared memory segment. The data required by each module is shown in Figure 3. The agent data used in the FLAMEGPU framework must be available to the visualisation module running on the CPU, which, in turn, must send sketched updates to the environment back to FLAMEGPU to influence the simulation running on the GPU.

3.1 Visualisation

The objective of this module is to provide a visual representation of the simulation and to create an interface where users can use sketching to manipulate the virtual crowd. Additionally, this module is responsible

[6] https://www.unrealengine.com/en-US/what-is-unreal-engine-4

[7] http://masagroup.github.io/recastdetour/

[8] http://www.flamegpu.com

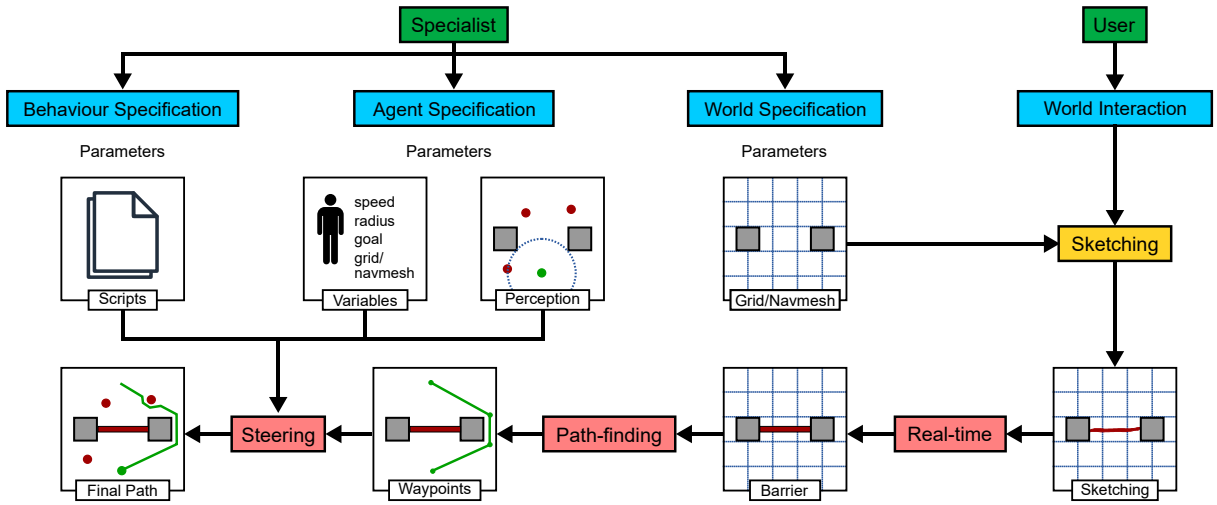


Figure 2: Overview of the crowd control system. In general, a domain specialist defines the model of the agents, their behaviour and the underlying data structure to represent the environment, in our case a navmesh. The user interacts with the environment by sketching on top of the navmesh to modify it in real-time. The updated navmesh is used to find the shortest path from the agent position to the goal. Agents follow this global path and use local forces (obtained from the agent and behaviour specification) to compute the final path avoiding inter-agent collisions. Based on Figure 1 in [KSB+12]

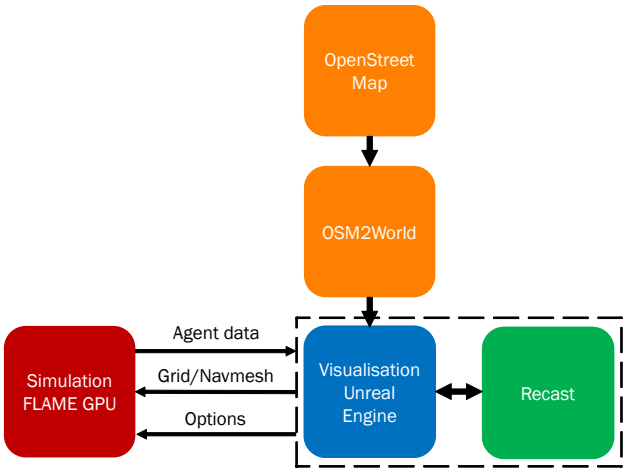


Figure 3: System overview.

for the navmesh creation and finding the paths to be followed by the agents.

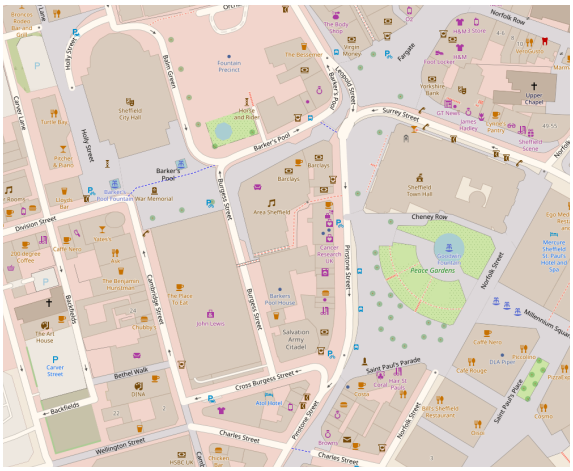
The area selected for the simulation is part of the city centre of Sheffield, UK. Figure 4 shows the area in OpenStreetMap[9] together with the final 3D model of the environment. The tool OSM2World[10] was used to convert the OpenStreetMap data into a 3D model before importing it into the game engine. Some modifications were made to the model prior to the import: imperfections on the ground were removed, tree models were substituted with a new 3D model, and a few materials were replaced.

3.2 Navmesh

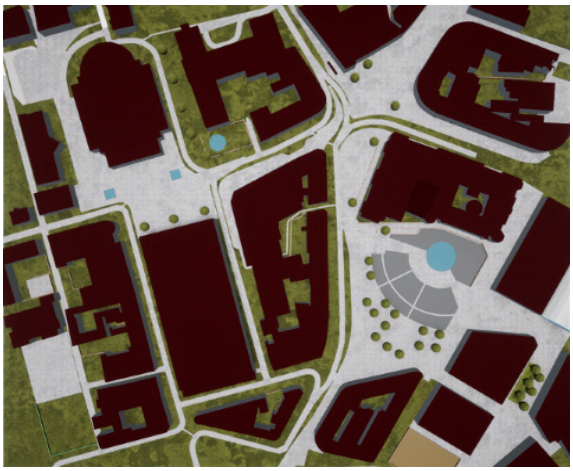
A novel contribution of this paper is the use of a navmesh to support the sketch-based solution. Previous work [GM17, JXW+08, PvdBC+11] used a grid-based approach. A navmesh approach is a more scalable solution (see Section 5 for a detailed comparison based on the scenarios we use in Section 4).

In our work, the underlying navmesh used to determine the movement of the agents is created with Recast, which is an open source tool used in games to automatically create a navmesh from a 3D environment. Recast divides the environment into square tiles on top of which a navmesh is generated which allows the individual update of the tiles. At the beginning of the simulation, the navmesh of all tiles is computed to generate the polygons (see Figure 5a) – Recast's mesh generation process produces some long, thin triangles, but this does not affect our sketch work. Later updates are only made in affected tiles, which facilitates real time modification of the mesh. The user has the option to hide or show the navmesh – in most of the figures in this paper the navmesh is visible to make it clear how it is updated based on user inputs. The mesh used to represent the environment in Figure 5 is

[9] https://www.openstreetmap.org/
[10] http://osm2world.org/

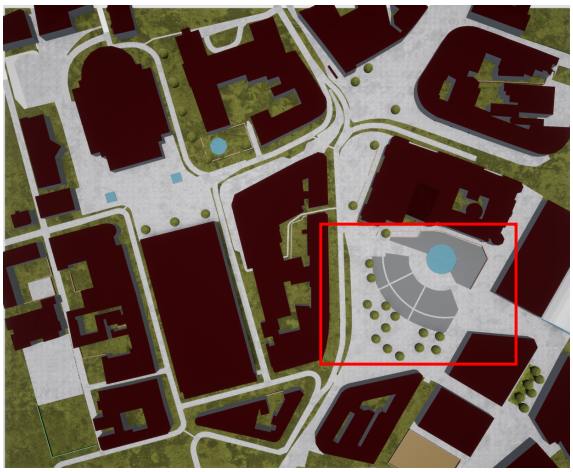


(a)

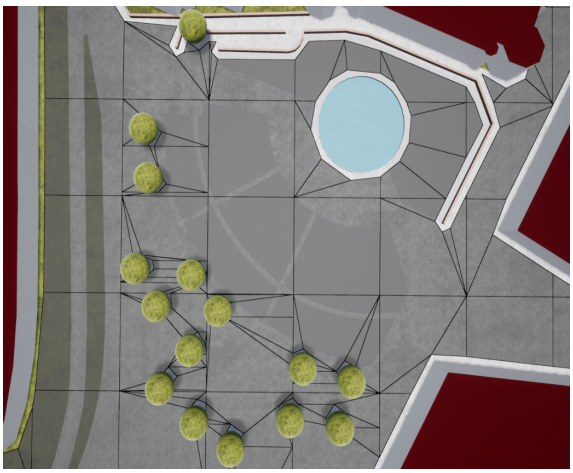


(b)

Figure 4: (a) Map of the selected area in *OpenStreetMap*. (b) Part of the final 3D environment in Unreal Engine.



(a)



(b)

Figure 5: (a) The tiled navmesh created with Recast and displayed in Unreal Engine. The underlying square tile pattern is shown, as well as the polygons created to connect different parts of the environment such as buildings and trees. (b) The environment where the simulation runs. The red rectangle highlights the area shown in (a).

formed by a rectangular grid of 21x19 (399) tiles. Of these, 333 tiles contain walkable areas. The resulting navmesh consists of 726 polygons. In addition, each tile is represented by a grid of voxels whose size is set to 0.15 – more details about these are used are given later. The tile and voxel size can be controlled by RECAST parameters.

The RECAST software was modified to implement sketching on top of the navmesh and to update it according to user actions (Section 3.4). The underlying tile data structure means changes are limited to affected tiles. After every navmesh change, the shortest path from every polygon to the target is recalculated. This information is sent to the simulation module through the shared memory segment.

3.3 Simulation

The agent-based simulation uses the social forces model [HM95] to determine the movement of the agents. Whilst this is a relatively simple model, it is sufficient to support the combined sketching work and navmesh use. A more complex approach could be used if required. The agent motion is the result of the weighted sum of three forces: (i) The pedestrian avoidance force for inter-agent collision avoidance. This is computed taking into account the position and velocity of nearby agents; (ii) The collision force used to prevent agents colliding with the envi-

ronment. The polygon edges without neighbours exert a repulsive force on agents depending on the distance; (iii) The goal force to guide agents to their destination. This is obtained using the graph (navmesh) sent by the visualisation module.

The graph is formed by nodes (polygons) and the edges shared by adjacent polygons. Each node contains a list of neighbouring polygons and the connecting edges. This graph is used to calculate the shortest path from every polygon to all the exits and areas. The $A*$ algorithm is used to compute the shortest path – this uses a heuristic value to guide the search for better performance. A route is computed for each exit. To create the route, every polygon stores the adjacent polygon leading to the corresponding exit. In this manner, agent movement can be calculated knowing the current polygon and the assigned exit of the agents. One approach to generate the agent movement is by following the middle point of the edges connecting the polygons of the shortest route. However, this would produce unrealistic paths. This problem can be solved by smoothing the resulting path using *The Simple Stupid Funnel Algorithm* [DB06]. This technique finds the corners of the path staying inside the polygons found by the A* route.

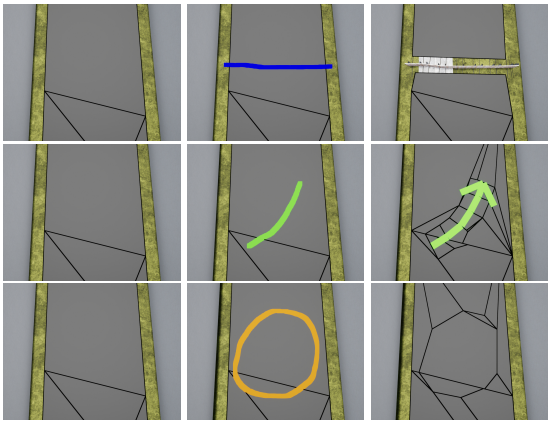
3.4 Sketching

The interface, implemented in UNREAL, allows the user to perform a series of actions by sketching or clicking in the environment. These actions include definition of agent spawn and goal locations, sketching obstacles to alter the crowd movement, creation of flow lines to guide the motion of the agents, drawing areas to create waypoints, and definition of journeys via storyboards. The entrances and exits are created by selecting a polygon edge with no neighbours. These locations define the spawning position of the agents and also serve as goals.

The first step to update the navmesh is to capture the user sketch and sample the line into equidistant points. Each sequence of points can represent an obstacle, a flow line or an area edge. Then, the line is mapped to the navmesh by marking the area covered by the sketch. These regions are given an id to differentiate among obstacles, flow lines and areas. The tiles affected by the user sketch are identified and the navmesh of these tiles is rebuilt with the new information.

3.4.1 Barriers

The barrier obstacles are created by marking the affected navmesh area as null. A null area cannot be crossed and is not used in navigation computation. The process is made efficient by using the tiles that the relevant navmesh area overlaps. Each overlapped tile is divided into an integer grid of voxels. Every voxel in the grid is tested to determine if it lies within the sketched obstacle region, whereupon it is marked as empty. Using this information, the contours of the updated walkable areas inside the affected tiles are calculated and these are used to re-triangulate this area to obtain the new polygons of the navmesh. The first row of Figure 6 shows the process of producing a barrier by sketching a line—navmesh polygons are generated on both sides of the barrier.



(a) (b) (c)

Figure 6: (a) Original navmesh. (b) User-sketched lines. (c) Updated navmesh and the elements created: barrier, flowline and area.

3.4.2 Flow Lines

A sketched flow line is given an id to identify it. The sketched flow line is divided into a set of polygons which are traversable only in the direction of the sketch (second row in Figure 6). The cost of traversing a flow line can be changed by the user – a higher value means that a flow line is more likely to attract agents from the surrounding area.

The addition of flow lines converts the navmesh in the flow line area into a directed graph which means that adjacent polygons are not necessarily connected for navigation purposes. Therefore, agents inside flow lines must follow the complete flow line until the end of it is reached. The routes to areas and exits are re-

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

calculated when the cost of traversing the flow lines is changed.

Sketching flow lines that cross tile boundaries complicates the process of creating a new navmesh. These situations occur when the same flow line area is mapped to two abutting tiles as shown in Figure 7. Numerical conversion issues can result in a misalignment between adjacent flow line polygons. When mapping the sketched flow line to the tile, a conversion from floating point (line coordinates) to integer (tile coordinates) is performed and this conversion may produce a misalignment. This problem generates small gaps where pedestrians can 'escape' from the flow line. In Figure 7, pedestrians would be able to leave the flow line early. The problem was solved by modifying the adjacency conditions of polygons from different tiles. Regular polygons from one tile are connected to the flow line polygons of the adjacent tile, but not vice-versa.

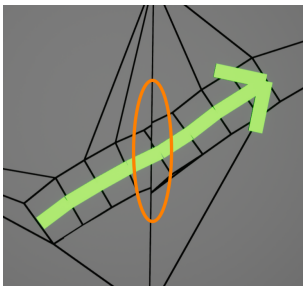


Figure 7: Polygons misaligned when crossing tile boundaries.

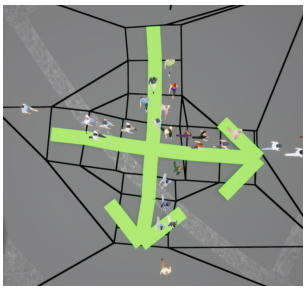
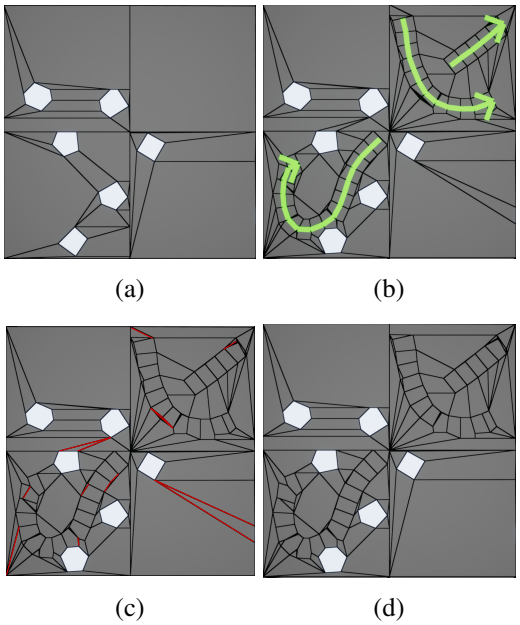


Figure 8: Pedestrians following overlapped flow lines.

It is possible to overlap flow lines and pedestrians are free to move between flow lines at a crossing point. In Figure 8, pedestrians follow their specific flow line without having problems at the crossing point. The weight of the last sketched flow line will be assigned to the intersection polygon. Pedestrians may be forced out of a flow line when it is crowded, however, they will try to re-enter unless a shortest path is found from their new navmesh polygon to their destination.

Sketching flow lines produces some long, thin and sometimes unnecessary polygons that could be merged into a single polygon. We thus improved the algorithm used by RECAST to merge polygons so as to reduce the total number of polygons. Figure 9 shows the result of the new algorithm in four tiles of the navmesh after sketching three flow lines. The original tiles and the flow lines created by the user are shown in Figure 9a and Figure 9b, respectively. Figure 9c shows the 148 polygons created with RECAST'S original algorithm. With the improved algorithm, this number is reduced to 130, as illustrated in Figure 9d. Typically, an initial navmesh does not have many tiles with complex areas or polygons. In these cases, the polygon reduction is not considerable but increases as the environment gets more complex with barriers and flow lines.



(a) (b) (c) (d)

Figure 9: (a) Four tiles of the navmesh. (b) The same four tiles with three flow lines sketched by the user. (c) *Recast's* polygon merge algorithm creates 148 polygons. Edges that are eliminated by our improved algorithm are highlighted in red. (d) The 130 polygons resulting from the improved algorithm.

3.4.3 Areas and Behaviours

The user can create areas in the environment that serve as waypoints. The shape of the sketched polygon is mapped to the navmesh and the new area is re-triangulated to eliminate concave polygons, as shown in the third row of Figure 6. The shortest path from each navmesh polygon to the area is then computed. In addition, the interface offers the ability to set the percentage of agents that will visit each area.

Currently, within an area, a simple wandering behaviour is implemented. When an agent reaches an area, it moves in random directions within the area for a predefined amount of time, before continuing on its journey. Whilst the idea of areas and behaviours has been previously suggested [HOC14], the main difference in our approach is the ability to set the percentage

of agents visiting and to assign the areas to a storyboard (Section 3.4.4). In future work other behaviours could be assigned to areas.

3.4.4 Storyboards

A storyboard defines the exact journey of agents throughout the simulation. To create a storyboard, the user must select an entrance polygon, areas (optional) and an exit. Figure 10 gives an example. The selected polygons and areas are highlighted and the indication arrows show the order of the storyboard, connecting entrances to intermediate points and on to targets. Individual pedestrians are guided by the storyboard but calculate their own specific path using the navmesh.

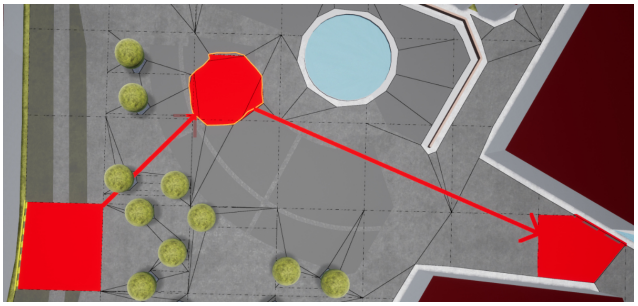


Figure 10: Storyboard created by the user. The selected polygons and areas are highlighted in green. The route begins at the left entrance, continues to the area in the middle and ends at the right exit.

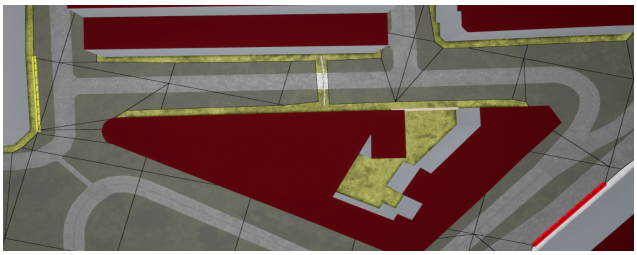
Currently, up to ten storyboards can be created per entrance. The user is able to define the percentage of pedestrians (spawned at the selected entrance) that follow the desired storyboard. A menu displays the existing storyboards and offers editing facilities. If no storyboard has been created, agents spawn at every entrance depending on a user-defined emission rate. Similarly, their exit is selected from all the exits based on percentages specified by the user. By creating storyboards, the user defines exact agent journeys.

3.4.5 Timeline

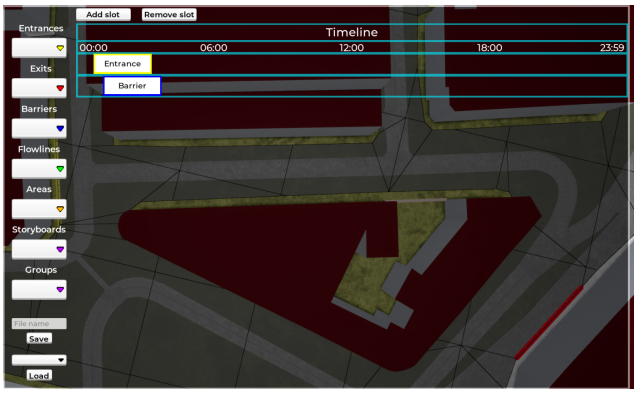
The simulation keeps track of the time allowing the user to simulate events during a day. Example events include: open/close entrances and exits; change the emission rate of an entrance; create barriers, flow lines, areas and storyboards. In addition, the speed of the simulation can be increased up to 24x to observe a day in an hour.

Using the timeline interface, the previously created elements can be dragged and dropped into the timeline to specify their occurrence time. The duration of the events is modified by re-sizing the elements. The simulation constantly checks for events and updates the environment accordingly to influence the pedestrian behaviour. The elements that are not added to the timeline are considered permanent as part of the environment.

Figure 11 shows a simple example of the timeline interface. In this scenario only two events are created. First, the emission rate of the yellow entrance is increased. Second, the barrier appears, blocking the path of the pedestrians and forcing them to walk around the building to reach the red exit.



(a)



(b)

Figure 11: (a) Simple scenario with one entrance (yellow), one barrier and one exit (red). (b) Timeline simulating two events. The emission rate of the entrance is increased, and the barrier appears shortly after that.

4 Validation scenarios

A set of scenarios were simulated to demonstrate the system's functionality. A video of the system in action is available at `https://tinyurl.com/y4qfkqb2`. The scenarios were run on an Intel Core i5 6500 with 16GB RAM and an NVIDIA GeForce GTX 1060 SC.

The performance of the system is shown in Table 2.

No. Agents	fps
1k	118-119
5k	86-87
10k	50-52
20k	24-26

Table 2: System performance with multiple crowd sizes.

The first scenario is the creation of a queue that simulates the entrance to a venue. Figure 12 shows a snake-like corridor created by sketching barriers.

The second scenario shows the use of flow lines (Figure 13) to control the crowd by forming lanes in areas such as road crossing points. Lane formation is a global behaviour that can emerge in agent-based simulations [TCP06]. Our system gives a user more control over where it occurs. Currently the user has no control over the width of a flow line. Another extension to consider for future work would be two-way flow lines.

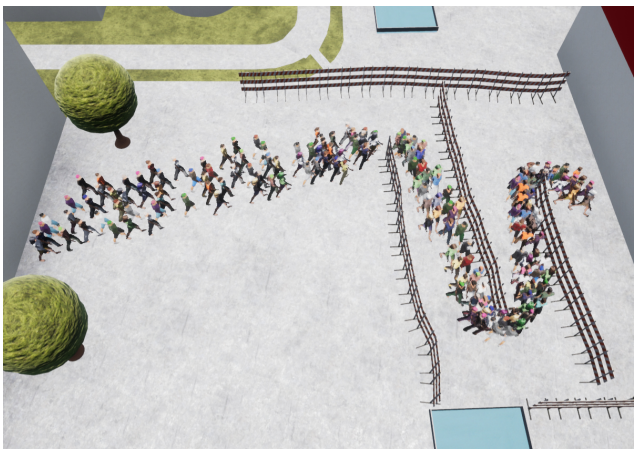


Figure 12: Pedestrians adjusting their path after the user created a corridor by sketching barriers.

A more illustrative example is shown in Figure 14 where two crowds move in opposite directions in a corridor. Multiple lanes are formed when both crowds meet, as seen in Figure 14a. The number of lanes can be reduced by sketching one flowline at each side of the corridor, towards the ends of the corridors (Figure 14b). These flowlines attract pedestrians moving in the same direction finally producing only two lanes.

Figure 15 shows the use of areas, which can be used as waypoints on a journey for a range of scenarios, e.g. where some pedestrians have to queue at a ticket machine before continuing in a train station or perhaps

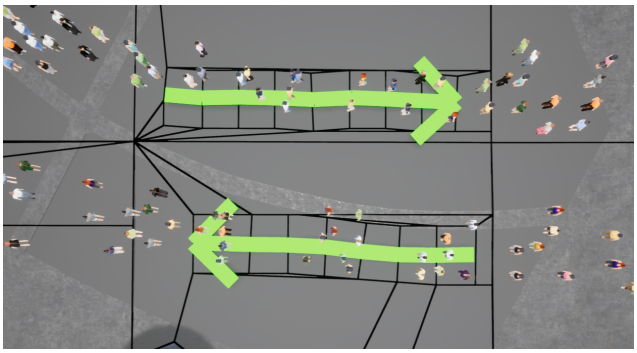
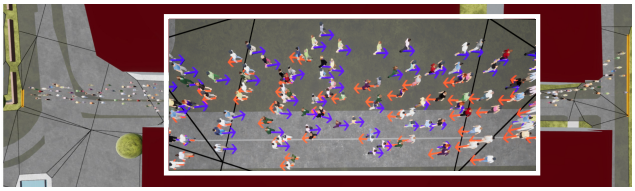
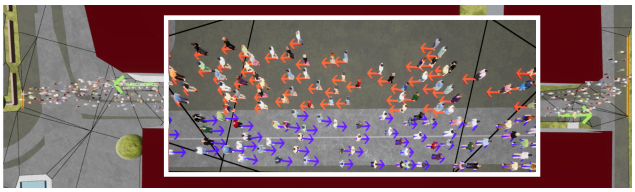


Figure 13: Pedestrians walking in the direction of the sketched flow lines.



(a)



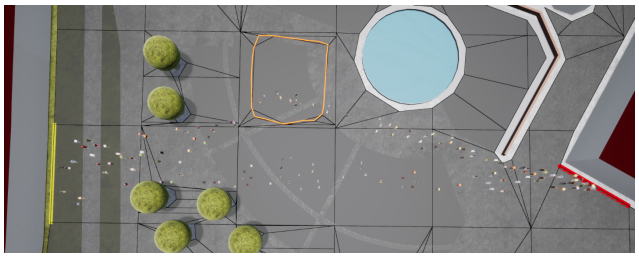
(b)

Figure 14: Pedestrians walking in opposite directions inside a corridor. (a) Multiple lanes are formed when the crowds meet. (b) Two flow lines are sketched, one at each end of the corridor, to control the number of lanes.

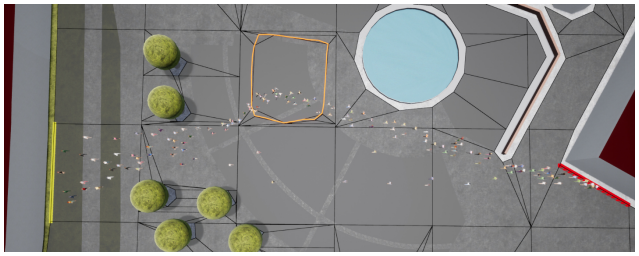
where pedestrians are stopping to watch some kind of street entertainment before continuing. In this example, the percentage of agents visiting the area, which is user controllable, is altered from 50% in Figure 15a to 90% in Figure 15b.

Storyboards offer complete control over the paths to be followed by the crowd. Figure 16 shows three storyboards created by the user – different shades of the same colour are used for storyboards that belong to the same entrance, and each entrance is assigned a different colour for its storyboards. The crowd is divided into three equal parts, each following a different storyboard. Two storyboards have an intermediate area and one directs pedestrians straight from the entrance to the exit. It can be difficult to see the different shades of colour used for storyboards emanating from the same entrance because the system allows up to 10

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037



(a)



(b)

Figure 15: Pedestrians walking from the right-hand side to the left-hand side via the same area (represented by the orange rectangle) with two different user-controllable area visit percentages: (a) 50%; (b) 90%.

storyboards from one entrance. This is something that still needs further work.

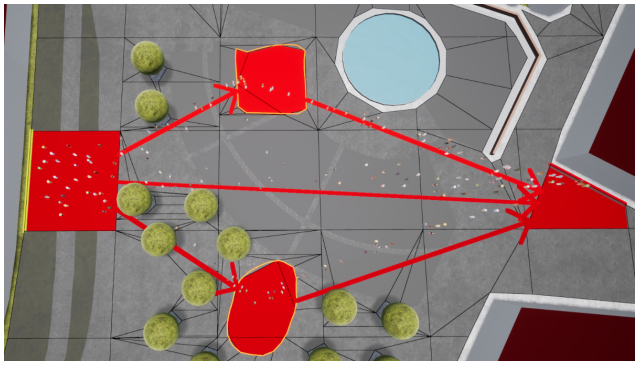


Figure 16: Pedestrians divided into three equal groups, each following a different storyboard.

Figure 17 shows a more complex scenario with 10 entrances/exits and 70 storyboards. In this example, a day in a city centre is simulated. Five entrances/exits are placed on the outskirts of the city centre and the other five entrances/exits are located in buildings. This simulates the flow of people going to work or visiting the area. Figure 18 shows the timeline for this complex environment. In the example, the emission rate of the named entrances is changed throughout the day to simulate two peak travel periods, as people travel to a city centre in the morning and then leave it in the evening. For example, the “Division” entrance has a high flow from 7 - 9 am, then a low flow until 4 pm. These people follow one of 5 storyboards to different buildings. For the building ”JLewis”, there is a low leaving rate from 10 am - 5 pm, followed by a high leaving rate from 5 - 7 pm. These people follow storyboards to exit routes (which are equivalent to the original entrance routes).

The use of storyboards combined with the timeline gives non-expert users control over the simulation to recreate real scenarios. One issue is that displaying all the storyboards simultaneously can cause visual clutter, as in Figure 17. To reduce this, users can hide/show storyboards individually or grouped by entrances. An area to investigate in future work would be the use of decision trees in conjunction with storyboards.

The last scenario is a practical application that could be used to marshal a crowd through a city. An example would be a police force controlling a football crowd moving from a railway station to a football ground. A video of the scenario is available at `https://tinyurl.com/y5cafnkj`. Figure 19a shows the starting position of the crowd to be marshalled (Entrance C) and its destination (Exit A). Additional pedestrians are also simulated to walk around the city. Figure 19b highlights the routes, entrances and exits used by the agents that are not part of the marshalled crowd. Without any guidance, the marshalled crowd would follow the path shown in Figure 19c, a route which passes through the middle of the potentially congested city centre area. The objective of the scenario is to dynamically control the path of the crowd (shown in Figure 19d) by sketching and deleting barriers to avoid congestion.

Particular frames of the simulated scenario are shown in Figure 20. At the beginning of the simulation (Figures 20 and 21) barriers are sketched to prevent the crowd from going to the area that will be congested. These obstacles also affect pedestrians who are not part of the marshalled crowd. Figure 22 shows that pedestrians moving from Entrance A to Exit B need to readjust their path and walk around the buildings on the left to reach their destination. This new route interferes with the path followed by the marshalled crowd, therefore it is blocked when the crowd approaches (Figure 23). The front part of the crowd arrives at Exit A in Figure 24. As the marshalled crowd walks past certain areas, barriers are deleted and pedestrians

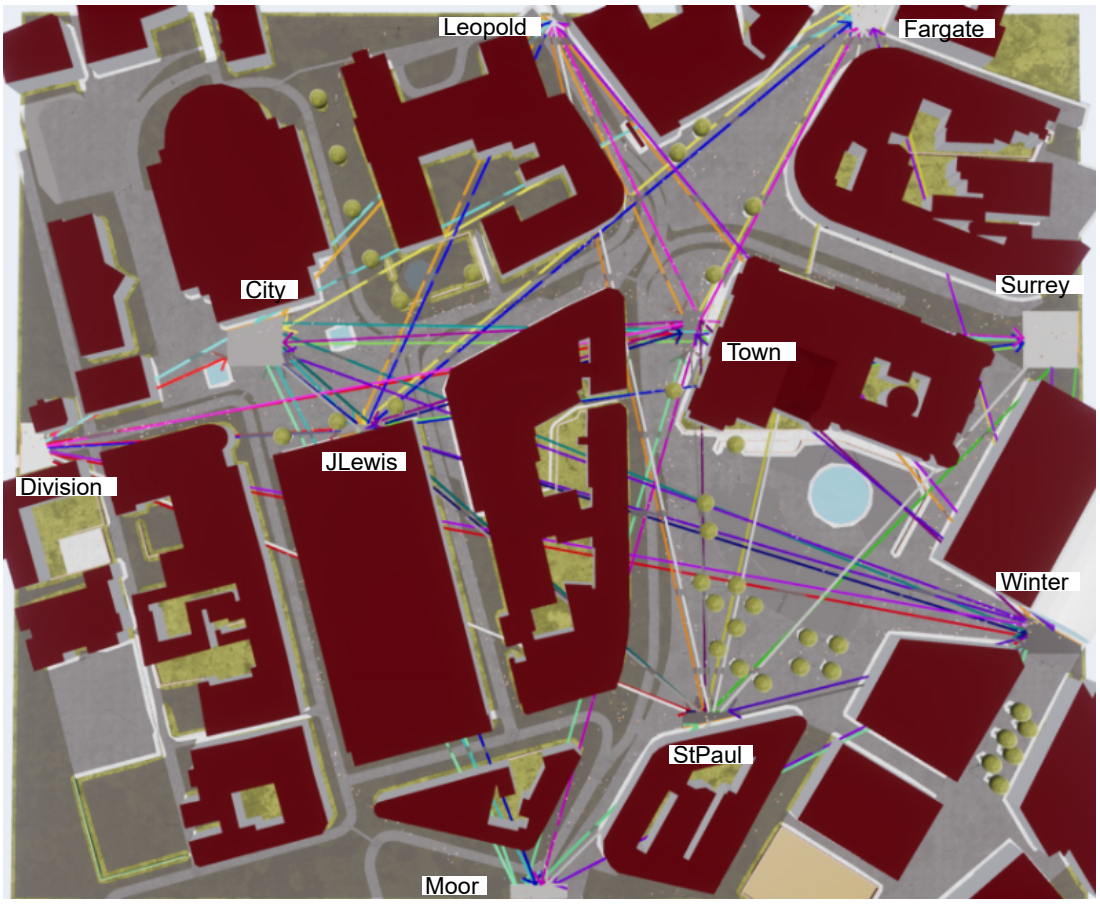


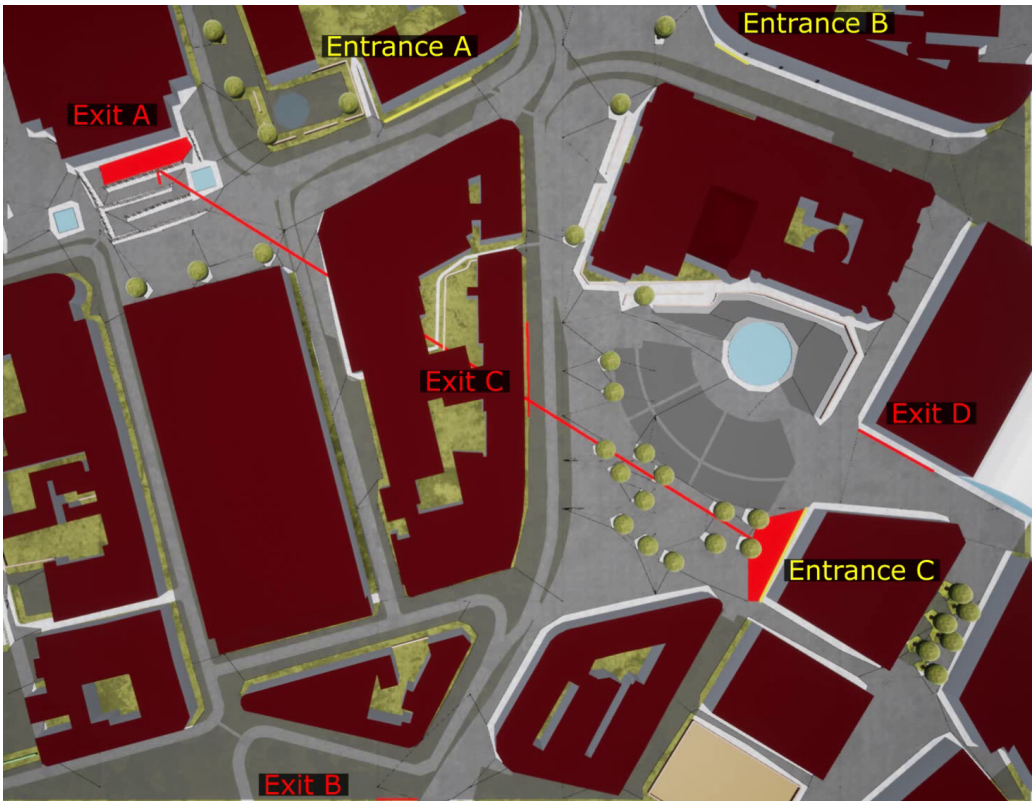
Figure 17: Complex scenario simulating a city centre with 10 entrances/exits and 70 storyboards.

Figure 18: Timeline interface. The elements created by the user can be selected with the drop-down lists on the left. These can be dragged into the timeline to determine the time of the event.

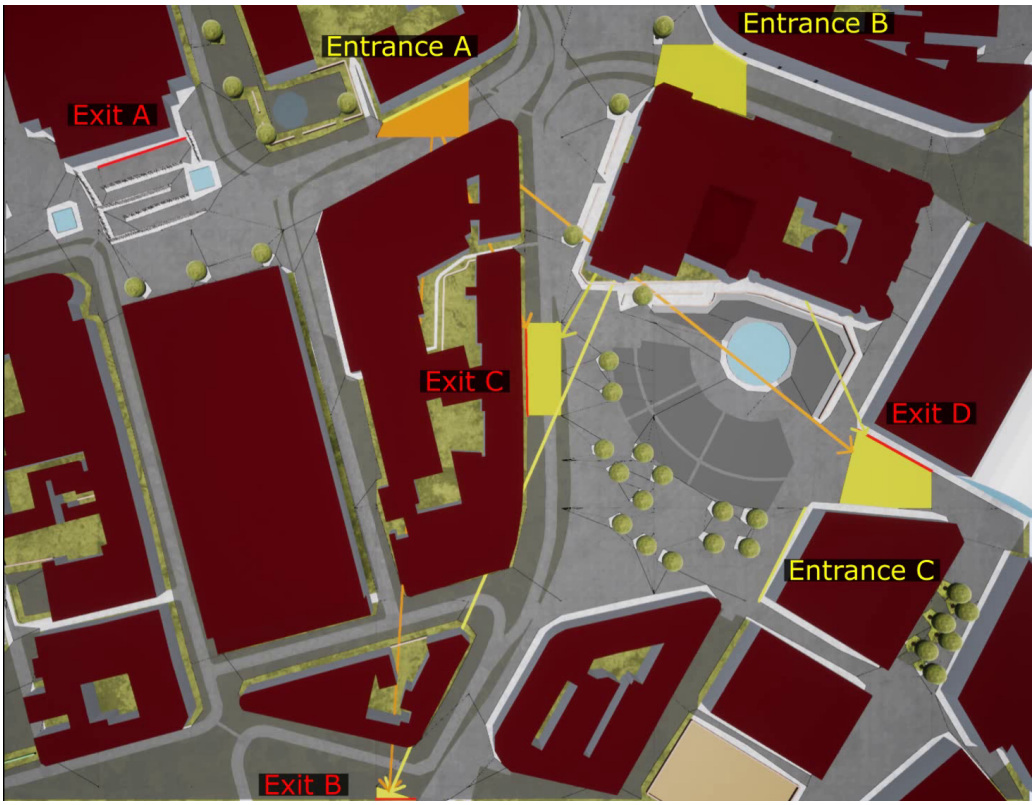
can retake their original path to their destination (Figure 25).

This scenario shows that it is possible to dynamically add and remove items such as barriers within the environment whilst a simulation is running. Multiple scenarios could easily be simulated in advance of an operation or tested in faster-than-real-time whilst a live situation is unfolding based on information feeds from on-the-ground operatives or from CCTV cameras. We believe this kind of functionality is unique to our system.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037



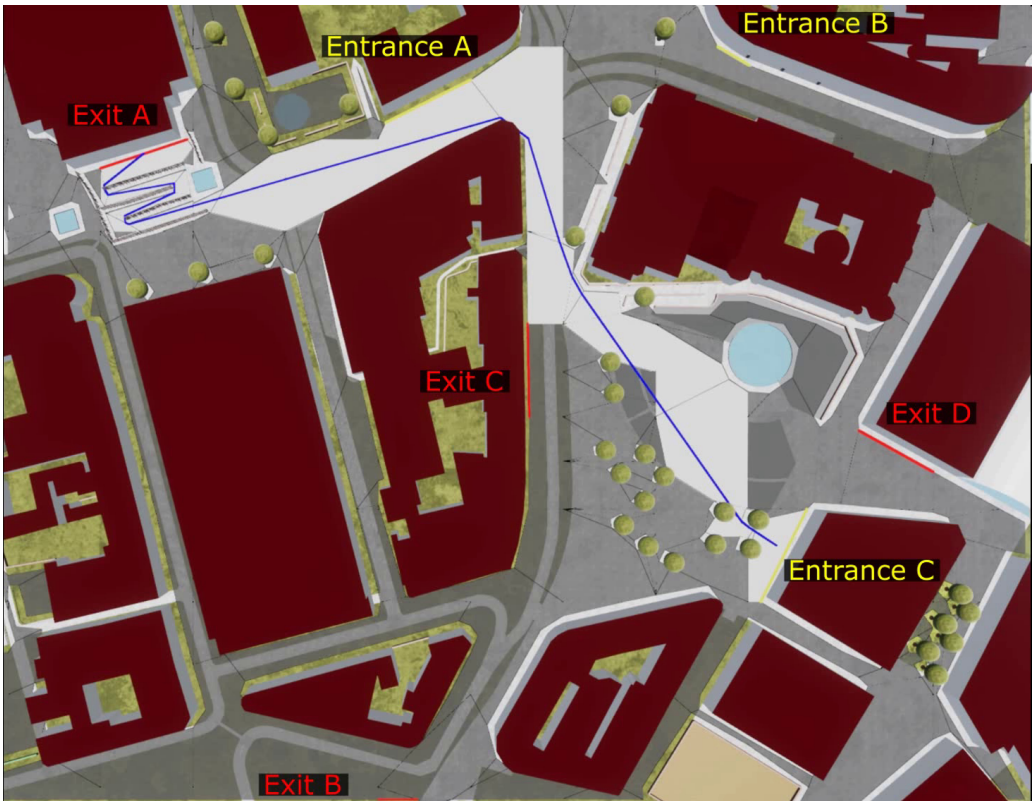
(a)



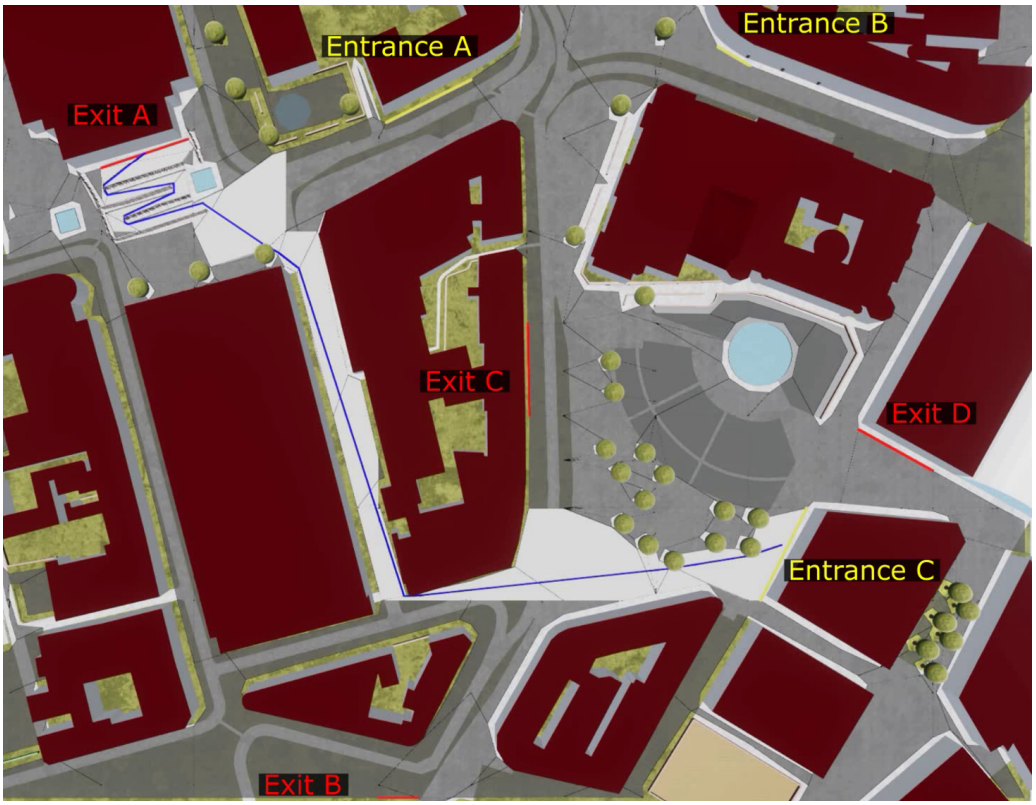
(b)

Figure 19: Crowd marshalling scenario. (a) The crowd to be marshalled spawns from Entrance C and moves towards Exit A. (b) Pedestrians walking around the city move from Entrances A and B to Exits B, C and D.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037



(c) (cont.)



(d) (cont.)

Figure 19 (cont.): Crowd marshalling scenario. (c) The shortest path that the crowd would follow without user intervention. (d) The desired path to be followed by the crowd while being marshalled.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

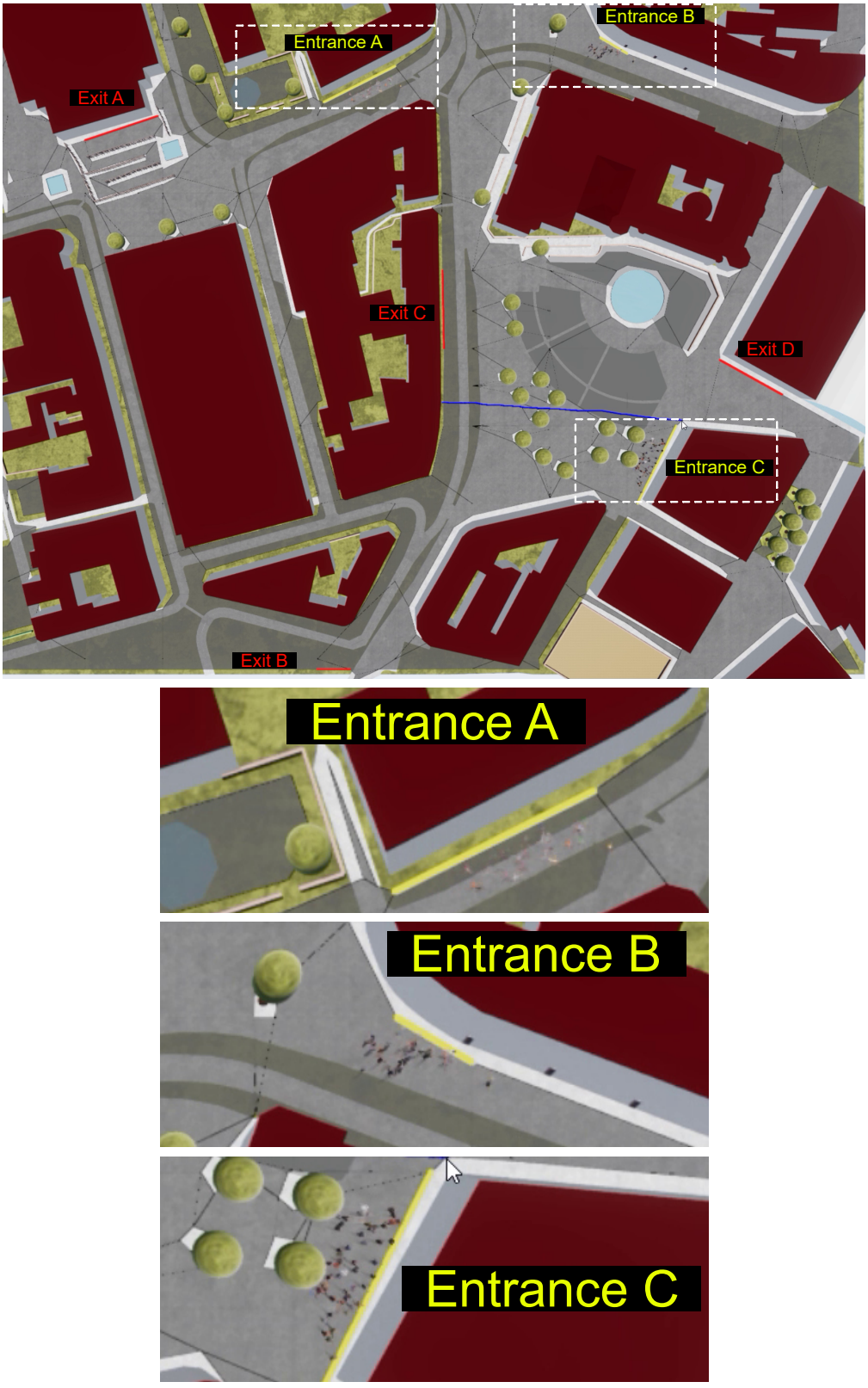


Figure 20: Simulation frames of a crowd being marshalled from Entrance C to Exit A in a city centre. Barrier sketched (blue line) by the user to block the crowd's path and avoid a congested area. Highlighted areas in the environment are enlarged to better show off the crowd.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

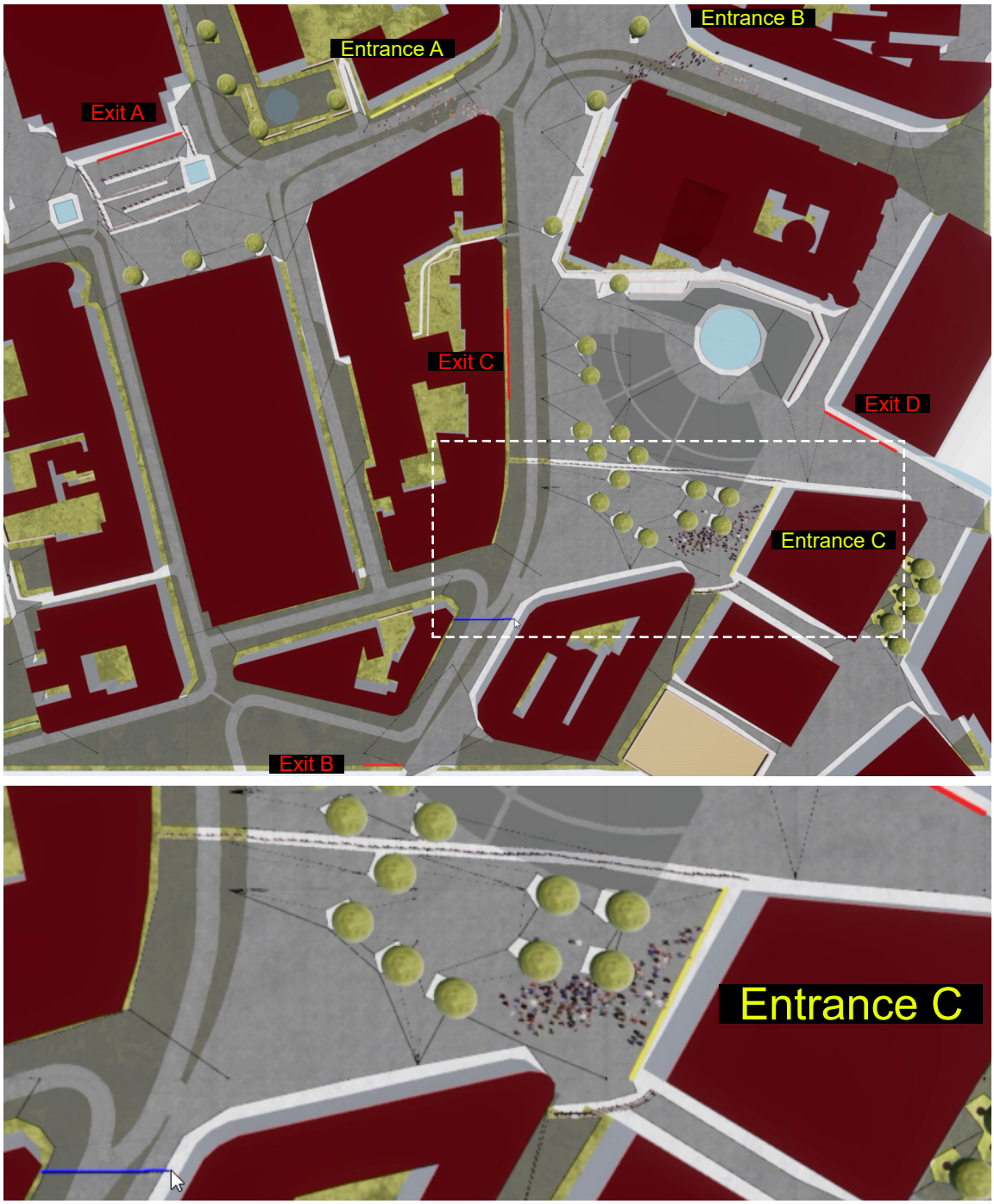


Figure 21: Simulation frames of a crowd being marshalled from Entrance C to Exit A in a city centre. User continues to create obstacles to define the route to be followed by the crowd. The highlighted area in the environment is enlarged to better show off the barriers and the crowd.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

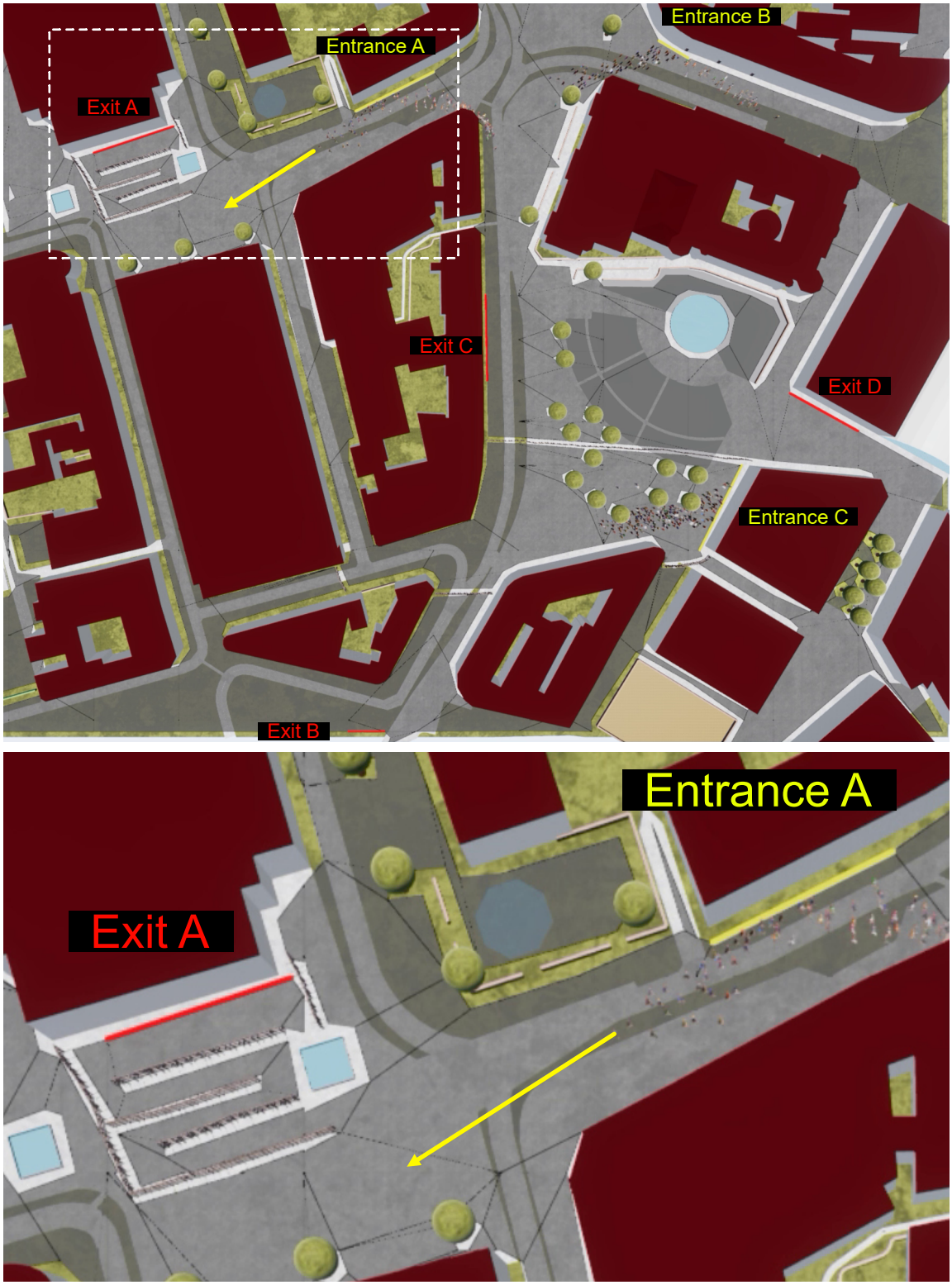


Figure 22: Simulation frames of a crowd being marshalled from Entrance C to Exit A in a city centre. Pedestrians moving from Entrance A to Exit B recalculate their shortest path, walking around the buildings on the left. The highlighted area in the environment is enlarged to better show off the Entrance A crowd and the new path.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

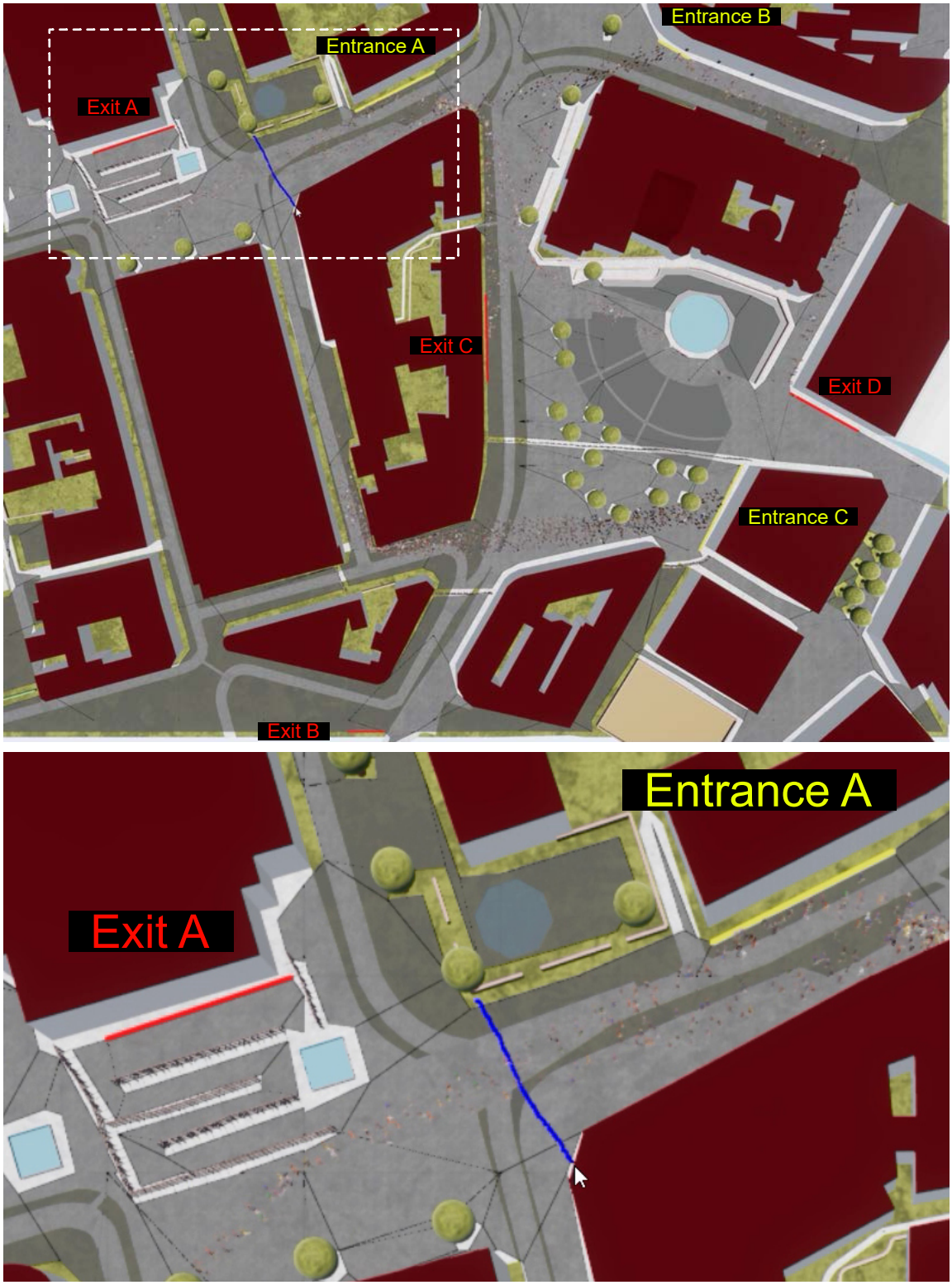


Figure 23: Simulation frames of a crowd being marshalled from Entrance C to Exit A in a city centre. The new route followed by A-to-B pedestrians is blocked with a barrier before the crowd arrives. The highlighted area in the environment is enlarged to better show off the new barrier to protect the marshalled crowd.

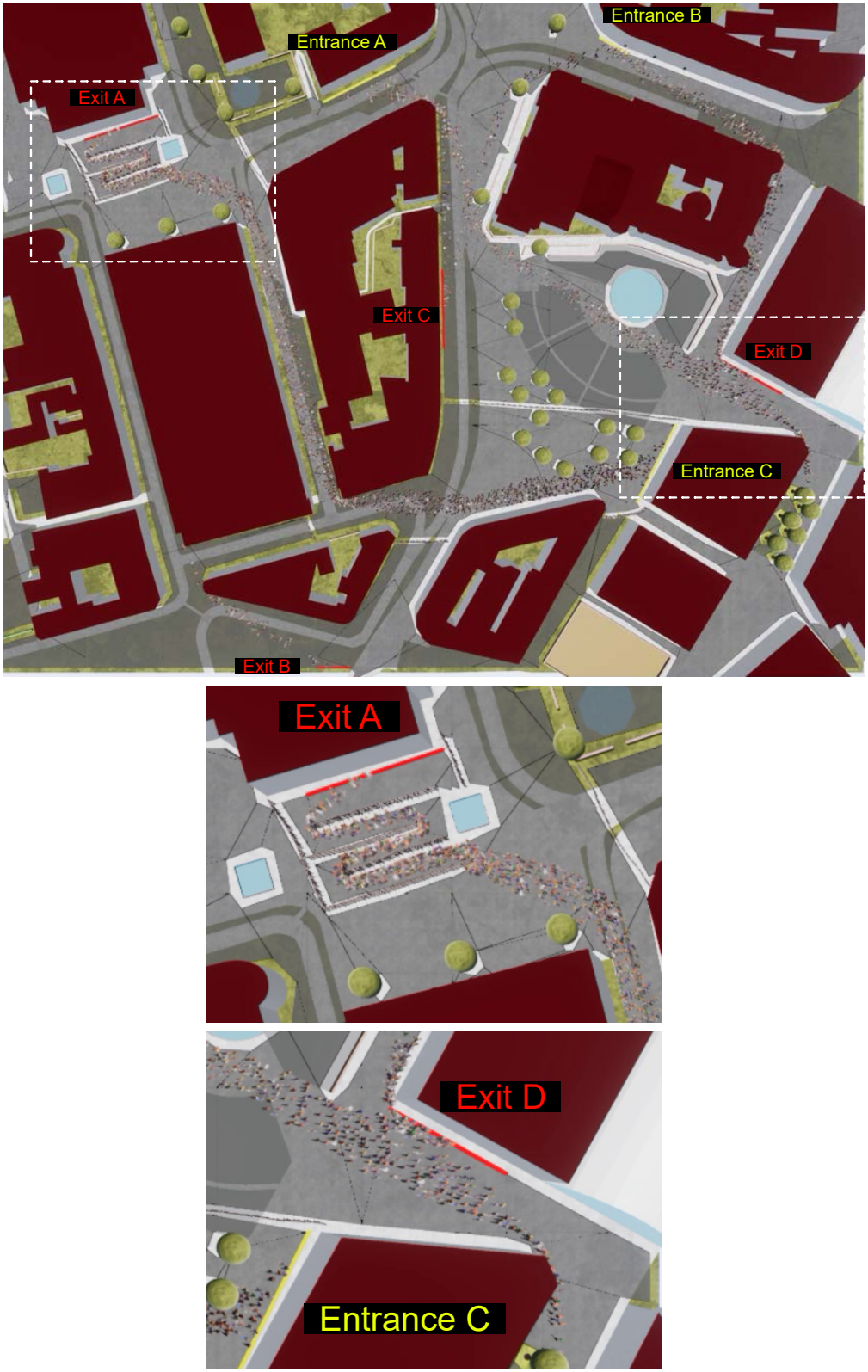


Figure 24: Simulation frames of a crowd being marshalled from Entrance C to Exit A in a city centre. The first pedestrians of the crowd arrive at Exit A. Highlighted areas in the environment are enlarged to better show off the crowd movement.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

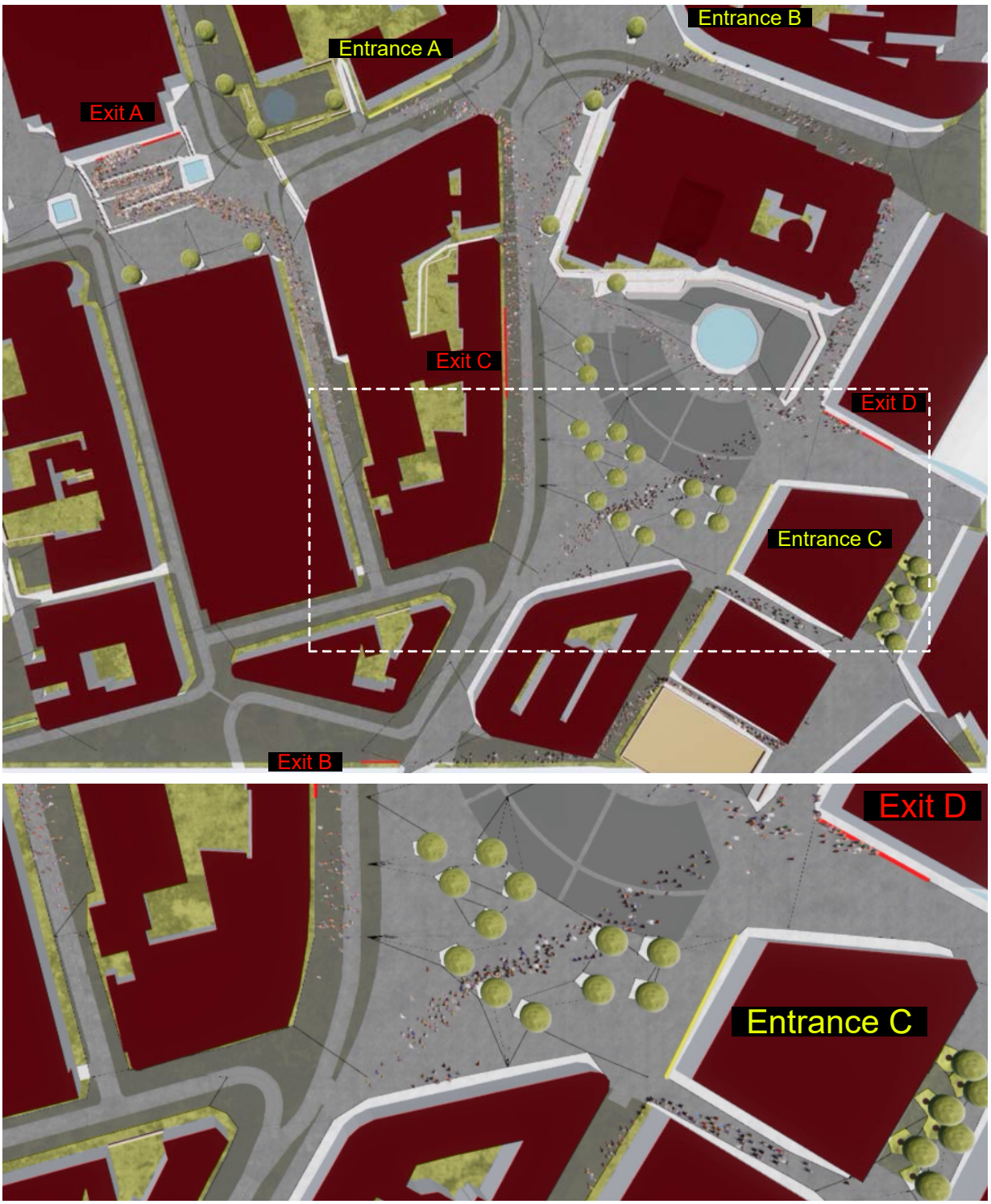


Figure 25: Simulation frames of a crowd being marshalled from Entrance C to Exit A in a city centre. Barriers begin to be deleted as the crowd walks past certain regions. Pedestrians who are not part of the crowd can then continue on their preferred path. The highlighted area in the environment is enlarged to better show off the crowd reforming their path.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

5 Grid vs Navmesh

Our approach makes use of a tiled navmesh unlike many previous examples which have used a grid-based approach. This section compares our tiled navmesh approach with a recent grid-based approach [GM17] using three criteria: environment representation and sketch accuracy, memory usage and computation time.

5.1 Environment Representation and Sketch Accuracy

The part of the environment to be represented is shown in Figure 26(a). For a grid-based approach, the representation accuracy of the grid depends on its resolution. Low resolution grids use bigger cells which cover larger areas of the environment, e.g. walkable surfaces and obstacles. However, a cell can only be marked as empty or occupied, leading to misrepresentation of the environment. This issue is evident in objects with circular shapes and when straight walls are not aligned with the grid, as shown in the top row of Figure 26, and also when curves and lines are sketched at angles to the underlying grid.

The navmesh provides a more accurate representation of the environment since it is based on the geometry of the model. Reducing the voxel size results in a better representation but also increases the number of polygons.

An advantage of the navmesh over the grid is that polygons cover larger areas compared to the cells of the grid, therefore fewer polygons are required to cover the entire walkable surface. The bottom row of Figure 26 shows the same zoomed area of the environment represented by navmeshes with different voxel size. As the voxel size is reduced, the number of polygons increases and the accuracy of the representation improves. Sketch precision is also better for the navmesh approach, since the navmesh does not require a small voxel size to represent a line in a reasonably accurate way, whereas the grid approach must increase its resolution.

In both the grid-based approach and the navmesh-based approach, increasing the accuracy of the representation impacts on the the memory used and the computation efficiency. Bigger environments require more data to be accurately represented, therefore a trade-off has to be made between environment representation, memory usage and computation time.

5.2 Memory Usage

A grid is represented by a two-dimensional array with each element set to empty or occupied. However, another structure is required to store the shortest paths to the goals. [GM17] used a flowmap per exit (as in [KRR10]) to store these paths. The flowmaps are grids where every cell stores a force directing agents to their target. Table 3 shows the memory used by grids of different sizes with multiple exits.

			Structure memory (kB)			
			No. exits			
Grid size	No. cells	Grid memory (kB)	1	3	5	7
128x128	16,384	128	256	512	768	1,024
256x256	65,536	256	1,024	2,048	3,072	4,096
512x512	262,144	512	4,096	8,192	12,288	16,384

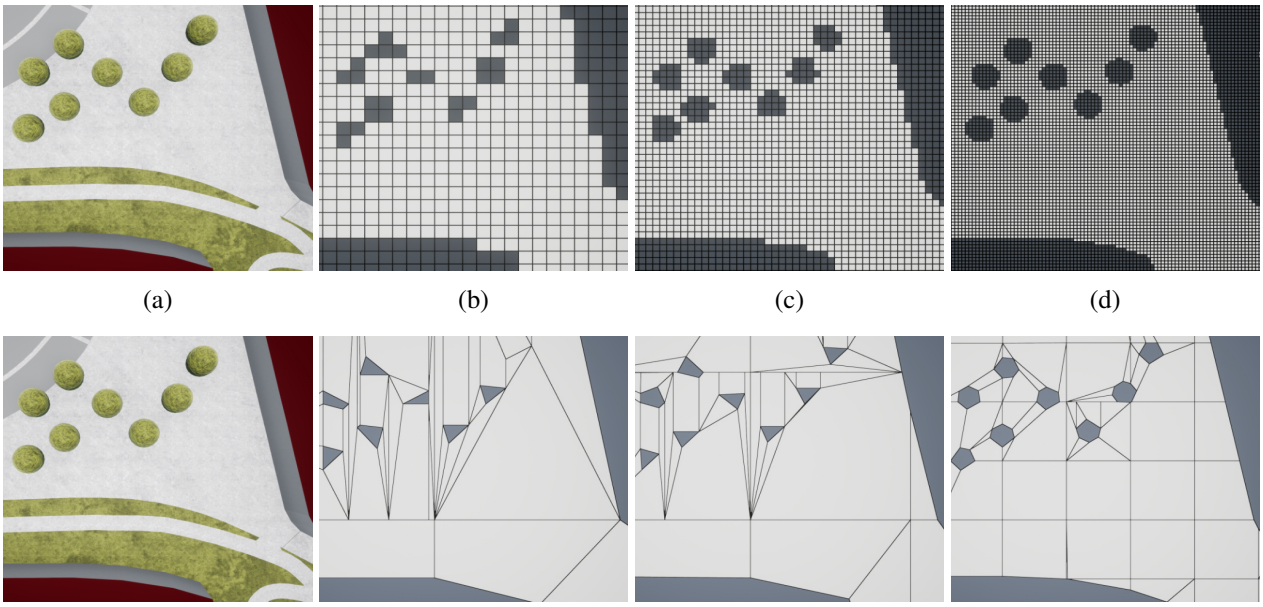
Table 3: Memory used by the grid approach with three different grid sizes.

The navmesh is represented by a structure that stores tiles, polygons, vertices and polygon adjacency, as well as other information required to find paths between polygons. In addition, an extra structure is defined to store the shortest routes, entrances, exits, areas, storyboards and a search grid used to accelerate the polygon search to determine the position of each agent. A disadvantage of the navmesh is that finding the polygon in which an agent is located requires testing the agent position against each polygon, whereas in the grid, the location of the agent can be directly mapped to a grid cell. Table 4 shows the number of tiles, polygons and vertices generated for three voxel sizes and also the memory used by the navmesh structure. Table 5 shows the memory used by the additional structure with different voxel sizes and multiple exits.

Voxel size	No. tiles	No. polygons	No. vertices	Memory used (kB)
0.5	44	516	1,363	80.2
0.25	168	911	2,643	163
0.1	921	2,213	7,310	499

Table 4: Number of elements generated by the navmesh with different voxel size, and memory used.

Figures 27 and 28 compares the memory used by three grids and three navmeshes, each with different sizes. It illustrates that the memory used by a grid is directly proportional to its resolution and the number of exits. When the number of cells is quadrupled, the memory usage is also increased fourfold. The grid approach has poor scalability. For the navmesh, memory



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 26: Grid and navmesh representation of the environment. Top row shows three grids with different resolutions: (a) original environment, (b) 128x128, (c) 256x256 and (d) 512x512. Bottom row shows three navmeshes with different voxel size: (e) original environment, (f) 0.5, (g) 0.25 and (h) 0.1.

	Structure memory (kB)			
	No. exits			
Voxel size	1	3	5	7
0.5	260	263	265	267
0.25	379	387	386	390
0.1	701	710	719	728

Table 5: Memory used by the structure storing the shortest paths with multiple exits.

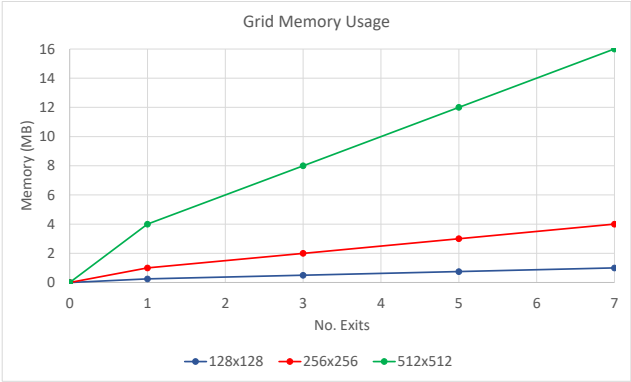


Figure 27: Memory used in megabytes (MB) by three grid sizes and multiple exits: 128x128 (blue), 256x256 (red) and 512x512 (green).

usage also increases with the number of tiles, polygons and exits, however, the growth rate is lower. The navmesh approach scales better with the size of the environment. In terms of memory usage, it is a more suitable option for large environments.

5.3 Computation Time

To compare the performance of both approaches, the time taken by three functions was measured: construction, update and pathfinding. A grid is built by using vertical raycasting to segment the world into cells. The update time is the amount of time taken to change the values of the grid when the environment is dynamically updated. Pathfinding uses a wavefront propagation algorithm to calculate the distance from each grid cell to the corresponding goal and the result is smoothed to create more realistic paths. Table 6 shows the time taken by these functions for three grids with multiple exits. The update times are similar since only the affected cells are updated. However, the paths must be recalculated, which requires more time as the grid resolution increases.

The process of building and updating a navmesh is described in Section 3. Table 7 shows the times for these processes and the pathfinding algorithm. The construction time is an issue for the navmesh approach. For a static environment, the navmesh can be created just once in an offline stage. For a dy-

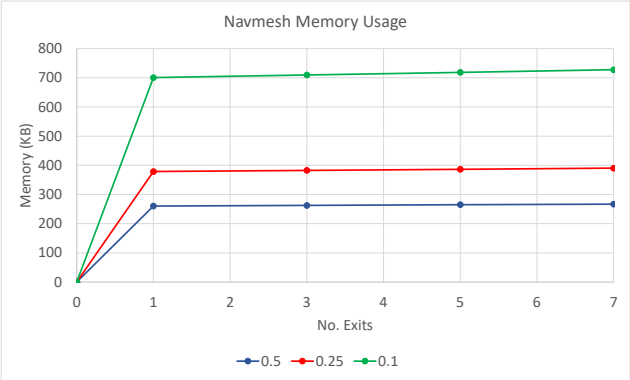


Figure 28: Memory used in kilobytes (kB) by three navmeshes with different voxel size and multiple exits: 0.5 (blue), 0.25 (red) and 0.1 (green).

namic environment, such as a sketching environment, the navmesh must be recalculated when the environment changes. However, using the tiled approach of RECAST, where the mesh is updated locally, ameliorates the cost. The update time is similar for each case since the number of voxels per tile is the same.

Figure 29 plots the time taken by the pathfinding algorithm for different size grids and navmeshes. For the navmesh, the major issue is that this time increases as the number of polygons grows. Sketching barriers, flowlines and areas creates more polygons, thus exacerbating the problem. Higher computation times could compromise the real time interaction with the simulation. This may be an issue in complex environments or with a large number of destinations.

6 Dynamic environment knowledge

Modifying the environment in real-time presents new challenges. In the system described in the previous sections, the environment changes made by the user are immediately mapped to a single navmesh, which is used to guide agents through the environment. Pedestrians react instantly to user sketches. Each person recalculates their route based on all the obstacles in the environment, including any newly sketched barrier which may not yet be visible. This issue raises the question of when agents should react to dynamic changes.

			Pathfinding (s)			
			No. exits			
Size	Build (s)	Update (s)	1	3	5	7
128x128	0.071	0.00003	0.003	0.008	0.019	0.029
256x256	0.279	0.0001	0.031	0.050	0.075	0.135
512x512	0.944	0.0005	0.062	0.251	0.324	0.45

Table 6: Time taken to create and update three grids of different sizes and to find the shortest paths for several goals.

			Pathfinding (s)			
			No. exits			
Size	Build (s)	Update (s)	1	3	5	7
0.5	0.134	0.003	0.003	0.009	0.015	0.024
0.25	0.492	0.002	0.009	0.031	0.054	0.078
0.1	3.433	0.004	0.076	0.253	0.427	0.597

Table 7: Time taken to create and update three navmeshes with different voxel sizes and to find the shortest paths for several goals.

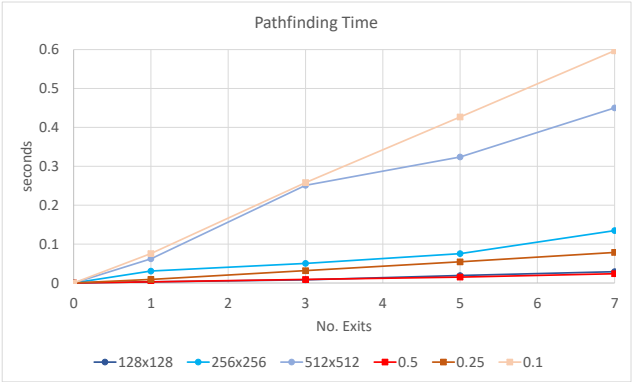


Figure 29: Time taken in seconds to calculate the shortest paths in grids and navmeshes with different sizes.

Previous work has proposed solutions to represent dynamic environments using constrained Delaunay triangulations [KBT04], adaptive roadmaps [SGA+07] and navmeshes [vTCG12]. However, these structures are immediately updated after dynamic obstacles are introduced into the environment and there is no discussion of when the crowd should react to the new environment. We explore this issue by implementing possible solutions and considering their performance implications to find the most suitable alternative for producing a more realistic simulation. Two main types of environment knowledge are considered: immediate and dynamic. Immediate, as its name suggests, means that pedestrians gain knowledge of any environment modification immediately after the user input. Dynamic means that pedestrians react to changes either when they see or hear about the change.

6.1 Immediate knowledge

Here, pedestrians gain knowledge of any environment modification immediately after the user input. This has the potential to produce unrealistic behaviour since agents that are located far away from the modified area change their path without seeing the change. Furthermore, any new pedestrians spawn with entire knowledge of the new environment, and thus follow the recalculated shortest path. This behaviour would be plausible in the unlikely scenario where all the pedestrians are notified in real-time (i.e. by a news item on their mobile phone, radio, etc.) of the dynamic updates, but this is an unlikely scenario. A route closure due to an ongoing incident is not necessarily immediately known to everyone in an environment.

6.2 Dynamic knowledge

Two kinds of dynamic reaction are identified: seeing (proximity-based) and hearing (time-based). This produces more realistic behaviour as people adjust their path only when they become aware of a change to the environment. We'll focus on barriers. A new barrier may block a route or a route may become available when a barrier is removed. For seeing knowledge, agents could be equipped with a visual sensor, as in [PHDL07]. We use a simpler approach based on radial distance. Agents "see" barriers based on proximity from the centre of the barrier. Time-based knowledge depends on a timer associated with a barrier sketched by the user. The barrier broadcasts knowledge of the change to all pedestrians after the elapsed time.

6.3 Comparison

Immediate knowledge within a simulation is the easiest solution. It requires no extra data structures to handle different pedestrians. Only one copy of the navmesh is required to handle all the obstacles sketched by users. Consequently, only one path calculation per polygon is performed per exit after each environment update. But gaining immediate knowledge of a change does not accurately model the real world.

In the dynamic approach, each agent has a different knowledge of a changing environment depending on what they have discovered, whether by seeing or hearing. A way to support dynamic knowledge is to use more than one navmesh for the environment. We'll again focus on barriers. A naive implementation would be to have every agent store its own navmesh and update it every time a barrier is discovered. However, this is unfeasible for large crowds and it is an inefficient solution since the same navmeshes are repeated thousands of times. A more suitable alternative is to have as many navmeshes as combinations of barriers: number of navmeshes = 2^n, where n is the number of barriers. In this manner, pedestrians would use the navmesh that includes the barriers they have discovered so far. A disadvantage of this implementation is that the navmesh path finding time and memory used grow exponentially, for both dynamic approaches, based on the number of barriers, as shown in Figures 30 and 31.

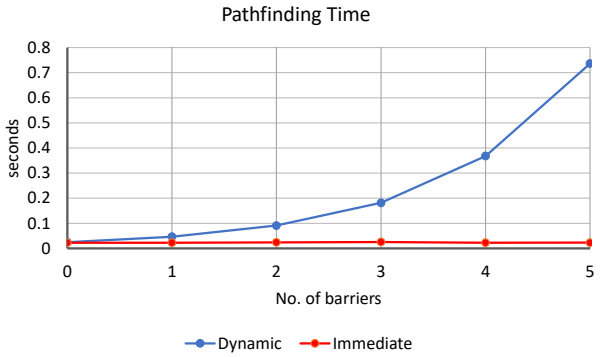


Figure 30: Comparison of the time taken in seconds to calculate the shortest path with multiple barriers by the dynamic and immediate knowledge approaches. The time for dynamic knowledge is the same for both approaches: vision and time.

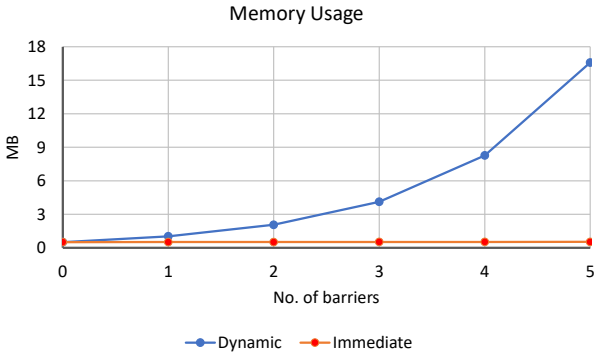


Figure 31: Comparison of the memory used by the navmeshes in the the dynamic and immediate knowledge approaches with multiple barriers. The memory for dynamic knowledge is the same for both approaches: vision and time

Figures 32 and 33 show frames of the same simula-

tion running for each different knowledge approach, immediate, seeing and seeing+hearing. Figure 32 shows barriers being added. Figure 33 then shows the barriers being deleted. The left column in each figure is labelled Immediate knowledge, the middle is Vision knowledge and the right is Vision+Time knowledge, corresponding to immediate, seeing and seeing+hearing knowledge, respectively. In each simulation, agents move from the yellow entrance at the top to the red exit in the bottom building. As barriers appear and disappear during a simulation, pedestrians adjust their path accordingly, depending on the type of environment knowledge. Agents are represented by coloured circles depending on the navmesh they are using to clearly show the environment knowledge they possess at that time. In the simulations with immediate knowledge (Immediate column), agents are always shown in red since all the modifications are made to one navmesh. For other kinds of knowledge, different colours are used.

Consider the second row of Figure 32, after sketching the first barrier. In the Immediate column, the agents are walking along the second corridor between the buildings on the left. In the Vision and Vision+Time approaches only pedestrians that have seen the barrier (blue circles) are trying to walk around the building. In the third row of Figure 32 a second barrier has been sketched and the spawning pedestrians in the Vision+Time simulation have turned blue since they have been notified of the existence of the first barrier. Agents with only Vision knowledge are still following the original path until they walk within the barrier proximity radius.

7 Conclusions

This paper has presented a solution for real-time control of virtual crowds without the need for technical knowledge and complex parameter tuning. Users can guide the pedestrian flow by sketching directly in the environment while the simulation is running. Multiple elements can be defined by sketching or clicking: entrances/exits, obstacles to block paths, flow lines to guide agents, waypoint areas, and storyboards to specify the journeys of the pedestrians. A timeline interface administers the simulation of events through the day. The underlying navigation approach used is a navmesh created with a modified version of the open source tool RECAST. The navmesh is a better alternative for navigation, since it represents the environment more accurately and requires less memory than the grid-based approach, although computation time is similar for both approaches. However, the navmesh approach has some limitations. The initial construction time (which can be done in an offline step) and the addition of new elements increases the number of polygons and consequently the path-finding computation time. Initial work has been done to reduce the number of polygons, but further work is required. We also addressed the issue of when a pedestrian should know about a change to the environment. Three kinds of knowledge were considered: immediate, seeing and hearing. The latter two increase the memory requirements for the environment. This is an area for further work. We also intend to run a user study looking at the ease of use of the interface as environment complexity increases.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

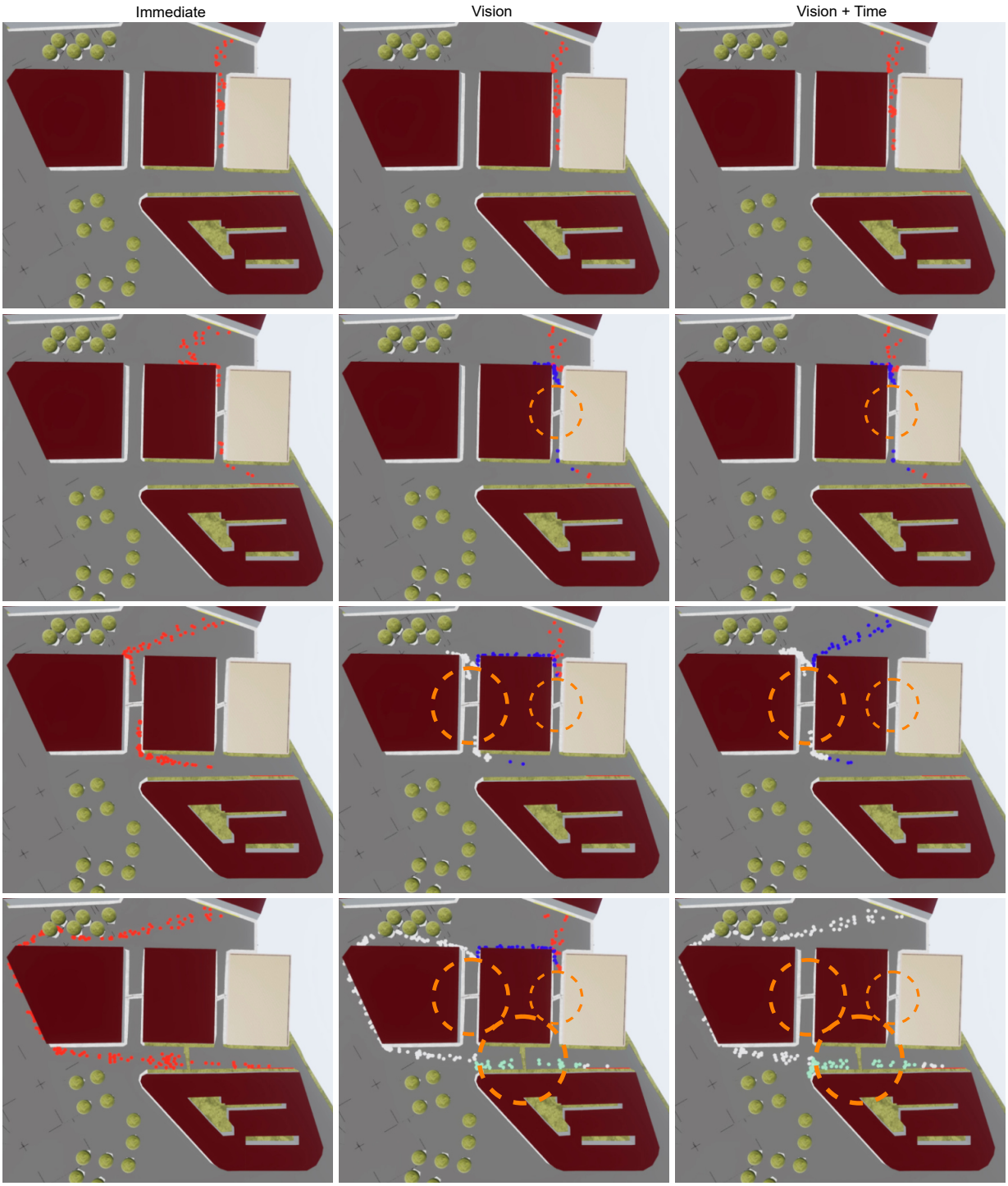


Figure 32: Frames of a simulation running for each different knowledge approach. Pedestrians move from the yellow entrance at the top to the red exit in the bottom building whilst barriers appear at different times to block their path. Agents are represented by coloured circles depending on the environment knowledge they possess at that time. The left column shows Immediate knowledge, where agents react instantly to any environment change. The middle column shows the Vision approach, where pedestrians become aware of barriers when they "see" them by walking inside their proximity radii (dashed orange circles). The right column illustrates the Vision+Time time solution, where agents become aware of the existence of a barrier in two ways: by seeing them, as in the Vision approach, or by being notified a short period of time after the barrier was created.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

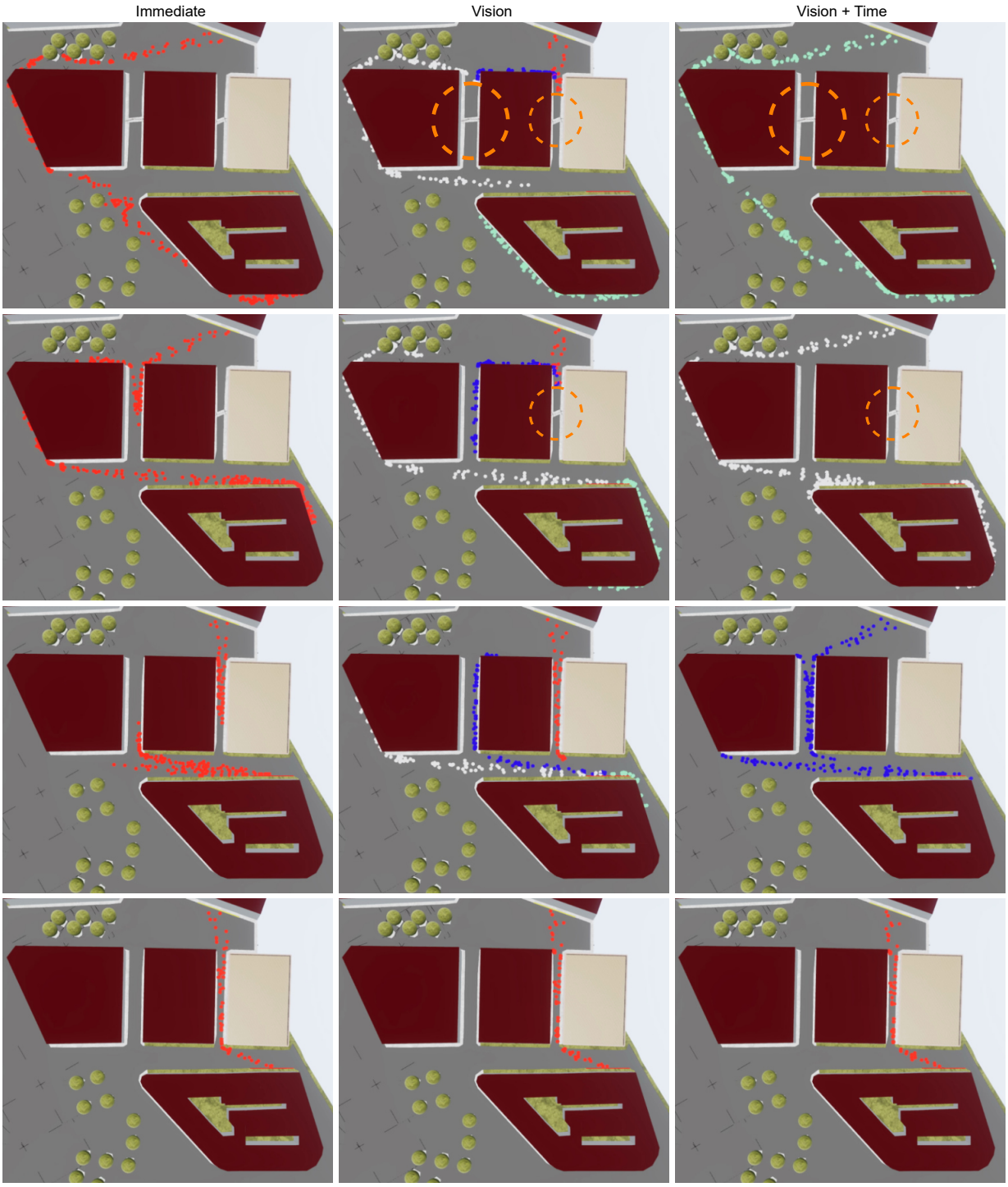


Figure 33: Frames of a simulation running for each different knowledge approach. Pedestrians move from the yellow entrance at the top to the red exit in the bottom building whilst the barriers disappear at different times to modify their path. Agents are represented by coloured circles depending on the environment knowledge they possess at that time. The left column shows Immediate knowledge, where agents react instantly to any environment change. The middle column shows the Vision approach, where pedestrians become aware of barriers when they "see" them by walking inside their proximity radius (dashed orange circles). Thus, agents do not adjust their current path when barriers disappear. The right column illustrates the Vision+Time solution, where agents become aware of an environment change in two ways: by seeing it, as in the Vision approach, or by being notified a short period of time after the modification.

urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037

References

[AG13] Ateş Akaydın and Ugur Güdükbay, *Adaptive grids: an image-based approach to generate navigation meshes*, Optical Engineering **52** (2013), no. 2, 027002, ISSN 0091-3286, DOI 10.1117/1.OE.52.2.027002.

[APKM15] Thomas Allen, Aleksandar Parvanov, Sam Knight, and Steve Maddock, *Using Sketching to Control Heterogeneous Groups*, Computer Graphics and Visual Computing (CGVC) (Rita Borgo and Cagatay Turkay, eds.), Eurographics Association, 2015, DOI 10.2312/cgvc.20151249, ISBN 978-3-905674-94-1.

[BKF16] Glen Berseth, Mubbasir Kapadia, and Petros Faloutsos, *ACCLMesh: curvature-based navigation mesh generation*, Computer Animation and Virtual Worlds **27** (2016), no. 3-4, 195–204, ISSN 1546-4261, DOI 10.1002/cav.1710.

[Che04] Stephen Chenney, *Flow tiles*, Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '04, Eurographics Association, 2004, DOI 10.1145/1028523.1028553, pp. 233–242, ISBN 978-3-905673-14-2.

[DB06] Douglas Demyen and Michael Buro, *Efficient Triangulation-Based Pathfinding*, Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1, AAAI Press, 2006, pp. 942–947, ISBN 9781577352815.

[GD11] Qin Gu and Zhigang Deng, *Formation Sketching: An Approach to Stylize Groups in Crowd Simulation*, Proceedings of Graphics Interface 2011 (Waterloo, CAN), GI '11, Canadian Human-Computer Communications Society, 2011, pp. 1–8, ISBN 9781450306935.

[GD13] Qin Gu and Zhigang Deng, *Generating Freestyle Group Formations in Agent-Based Crowd Simulations*, IEEE Computer Graphics and Applications **33** (2013), no. 1, 20–31, ISSN 1558-1756, DOI 10.1109/MCG.2011.87.

[GM17] Luis Rene Montana Gonzalez and Steve Maddock, *Sketching for Real-time Control of Crowd Simulations*, Computer Graphics and Visual Computing (CGVC) (Tao Ruan Wan and Franck Vidal, eds.), Eurographics Association, 2017, DOI 10.2312/cgvc.20171282, ISBN 978-3-03868-050-5.

[GM19] Luis Rene Montana Gonzalez and Steve Maddock, *A Sketch-based Interface for Real-time Control of Crowd Simulations that Use Navigation Meshes*, Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP, INSTICC, SciTePress, 2019, DOI 10.5220/0007344200410052, pp. 41–52, ISBN 978-989-758-354-4.

[GOL96] Edwin R. Galea, Matthew Owen, and Peter J. Lawrence, *The EXODUS Model*, Fire Engineers Journal **56** (1996), no. 183, 26–30.

[HAMB+14] Sandro Hauri, Javier Alonso-Mora, Andreas Breitenmoser, Roland Siegwart, and Paul Beardsley, *Multi-Robot Formation Control via a Real-Time Drawing Interface*, Field and Service Robotics: Results of the 8th International Conference (Berlin, Heidelberg) (Kazuya Yoshida and Satoshi Tadokoro, eds.), Springer Berlin Heidelberg, 2014, DOI 10.1007/978-3-642-40686-7_12, pp. 175–189, ISBN 978-3-642-40686-7.

[Hel91] Dirk Helbing, *A mathematical model for the behavior of pedestrians*, Behavioral Science **36** (1991), no. 4,

298–310, ISSN 1099-1743, DOI 10.1002/bs.3830360405.

[HM95] Dirk Helbing and Peter Molnar, *Social force model for pedestrian dynamics*, Physical Review E **51** (1995), no. 5, 4282–4286, DOI 10.1103/PhysRevE.51.4282.

[HOC14] Rowan Hughes, Jan Ondřej, and Guarev Chaurasia, *Sketch-Based Annotation and Authoring of Geometrically Sparse 3d Environments*, Tech. report, School of Computer Science and Statistics, Trinity College Dublin, 2014, TCD-CS-2014-01.

[HSK12] Joseph Henry, Hubert P. H. Shum, and Taku Komura, *Environment-aware Real-time Crowd Control*, Proceedings of the 2012 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU) (Jehee Lee and Paul Kry, eds.), SCA '12, Eurographics Association, 2012, DOI 10.2312/SCA/SCA12/193-200, pp. 193–200, ISBN 978-3-905674-37-8.

[HY09] D. Hunter Hale and G. Michael Youngblood, *Full 3D Spatial Decomposition for the Generation of Navigation Meshes*, Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'09, AAAI Press, 2009, pp. 142–147.

[HYD08] D. Hunter Hale, G. Michael Youngblood, and Priyesh N. Dixit, *Automatically-generated Convex Region Decomposition for Real-Time Spatial Agent Navigation in Virtual Worlds*, Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'08, AAAI Press, 2008, pp. 173–178.

[JCC+15] Kevin Jordao, Panayiotis Charalambous, Marc Christie, Julien Pettré, and Marie-Paule Cani, *Crowd Art: Density and Flow Based Crowd Motion Design*, Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (New York, NY, USA), MIG '15, Association for Computing Machinery, 2015, DOI 10.1145/2822013.2822023, pp. 167–176, ISBN 9781450339919.

[JPCC14] Kevin Jordao, Julien Pettré, Marc Christie, and Marie-Paule Cani, *Crowd Sculpting: A Space-time Sculpting Method for Populating Virtual Environments*, Computer Graphics Forum **33** (2014), no. 2, 351–360, ISSN 0167-7055, DOI 10.1111/cgf.12316.

[JXW+08] Xiaogang Jin, Jiayi Xu, Charlie C.L. Wang, Shengsheng Huang, and Jun Zhang, *Interactive Control of Large-Crowd Navigation in Virtual Environments Using Vector Fields*, IEEE Computer Graphics and Applications **28** (2008), no. 6, 37–46, ISSN 1558-1756, DOI 10.1109/MCG.2008.117.

[Kal05] Marcelo Kallmann, *Path Planning in Triangulations*, Proceedings of the IJCAI workshop on reasoning, representation, and learning in computer games (Edinburgh, Scotland), 2005, pp. 49–54.

[KBT04] Marcelo Kallmann, Hanspeter Bieri, and Daniel Thalmann, *Fully Dynamic Constrained Delaunay Triangulations*, Geometric Modeling for Scientific Visualization, Springer Berlin Heidelberg, 2004, DOI 10.1007/978-3-662-07443-5_15, pp. 241–257, ISBN 978-3-662-07443-5.

[KHHL12] Manmyung Kim, Youngseok Hwang, Kyunglyul Hyun, and Jehee Lee, *Tiling Motion Patches*, Proceedings of the 2012 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '12, Eurographics Association, 2012, pp. 117–126, ISBN 9783905674378.

[KHKL09] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee, *Synchronized Multi-character Motion Editing*, ACM Transactions on Graphics **28** (2009), no. 3, 79:1–79:9, ISSN 0730-0301, DOI 10.1145/1531326.1531385.

[KLLT08] Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Shigeo Takahashi, *Group Motion Editing*, ACM SIGGRAPH 2008 Papers (New York, NY, USA), SIGGRAPH '08, Association for Computing Machinery, 2008, DOI 10.1145/1399504.1360679, pp. 80:1–80:8, ISBN 9781450301121.

[KRR10] Twin Karmakharm, Paul Richmond, and Daniela M. Romano, *Agent-based Large Scale Simulation of Pedestrians With Adaptive Realistic Navigation Vector Fields*, Theory and Practice of Computer Graphics (John Collomosse and Ian Grimstead, eds.), Eurographics Association, 2010, DOI 10.2312/LocalChapterEvents/TPCG/TPCG10/067-074, pp. 67–74, ISBN 978-3-905673-75-3.

[KSB+12] Mubbasir Kapadia, Alexander Shoulson, Cory D. Boatright, Pengfei Huang, Funda Durupinar, and Norman I. Badler, *What's Next? The New Era of Autonomous Virtual Humans*, Motion in Games (Marcelo Kallmann and Kostas Bekris, eds.), Springer Berlin Heidelberg, 2012, DOI 10.1007/978-3-642-34710-8_16, pp. 170–181, ISBN 978-3-642-34710-8.

[KSKL14] Jongmin Kim, Yeongho Seol, Taesoo Kwon, and Jehee Lee, *Interactive Manipulation of Large-scale Crowd Animation*, ACM Transactions on Graphics **33** (2014), no. 4, 83:1–83:10, ISSN 0730-0301, DOI 10.1145/2601097.2601170.

[LCL06] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee, *Motion Patches: Building Blocks for Virtual Environments Annotated with Motion Data*, ACM Transactions on Graphics **25** (2006), no. 3, 898–906, ISSN 0730-0301, DOI 10.1145/1141911.1141972.

[MMHR16] James McIlveen, Steve Maddock, Peter Heywood, and Paul Richmond, *PED: Pedestrian Environment Designer*, Computer Graphics and Visual Computing (CGVC) (Cagatay Turkay and Tao Ruan Wan, eds.), Eurographics Association, 2016, DOI 10.2312/cgvc.20161304, ISBN 978-3-03868-022-2.

[MR05] Erik Millán and Isaac Rudomin, *Agent Paint: Intuitive Specification and Control of Multiagent Animations*, International Conference in Computer Animation and Social Agents (CASA), 2005.

[MT97] Soraia R. Musse and Daniel Thalmann, *A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis*, Computer Animation and Simulation '97 (Daniel Thalmann and Michiel van de Panne, eds.), Springer Vienna, 1997, DOI 10.1007/978-3-7091-6874-5_3, pp. 39–51, ISBN 978-3-7091-6874-5.

[OO09] Masaki Oshita and Yusuke Ogiwara, *Sketch-Based Interface for Crowd Animation*, Smart Graphics (Andreas Butz, Brian Fisher, Marc Christie, Antonio Krüger, Patrick Olivier, and Roberto Therón, eds.), Springer Berlin Heidelberg, 2009, DOI 10.1007/978-3-642-02115-2_22, pp. 253–262, ISBN 978-3-642-02115-2.

[OP11] Ramon Oliva and Nuria Pelechano, *Automatic Generation of Suboptimal NavMeshes*, Motion in Games (Jan M. Allbeck and Petros Faloutsos, eds.), Springer Berlin Heidelberg, 2011, DOI 10.1007/978-3-642-25090-3_28, pp. 328–339, ISBN 978-3-642-25090-3.

[OP13] Ramon Oliva and Nuria Pelechano, *NEOGEN: Near Optimal Generator of Navigation Meshes for 3D Multi-Layered Environments*, Computers & Graphics **37** (2013), no. 5, 403–412, ISSN 0097-8493, DOI 10.1016/j.cag.2013.03.004.

[PAB07] Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler, *Controlling Individual Agents in High-Density Crowd Simulation*, Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '07, Eurographics Association, 2007, pp. 99–108, ISBN 9781595936240.

[PHDL07] Xiaoshan Pan, Charles S. Han, Ken Dauber, and Kincho H. Law, *A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations*, AI & SOCIETY **22** (2007), no. 2, 113–132, ISSN 1435-5655, DOI 10.1007/s00146-007-0126-1.

[PvdBC+11] Sachin Patil, Jur van den Berg, Sean Curtis, Ming C. Lin, and Dinesh Manocha, *Directing Crowd Simulations Using Navigation Fields*, IEEE Transactions on Visualization and Computer Graphics **17** (2011), no. 2, 244–254, ISSN 1077-2626, DOI 10.1109/TVCG.2010.33.

[RCL+11] Yunbo Rao, Leiting Chen, Qihe Liu, Weiyao Lin, Yanmei Li, and Jun Zhou, *Real-time control of individual agents for crowd simulation*, Multimedia Tools and Applications **54** (2011), no. 2, 397–414, ISSN 1573-7721, DOI 10.1007/s11042-010-0542-y.

[Rey87] Craig W. Reynolds, *Flocks, Herds and Schools: A Distributed Behavioral Model*, SIGGRAPH Computer Graphics **21** (1987), no. 4, 25–34, ISSN 0097-8930, DOI 10.1145/37402.37406.

[Rey99] Craig W. Reynolds, *Steering Behaviors For Autonomous Characters*, Game Developers Conference, 1999, pp. 763–782.

[Rey00] Craig W. Reynolds, *Interaction with Groups of Autonomous Characters*, Game Developers Conference, 2000, pp. 449–460.

[RR08] Paul Richmond and Daniela M. Romano, *A High Performance Framework For Agent Based Pedestrian Dynamics On GPU Hardware*, Proceedings of EUROSIS ESM 2008 (European Simulation and Modelling), Universite du Havre, Le Havre, France, 2008.

[SBPO05] Barry G. Silverman, Norman I. Badler, Nuria Pelechano, and Kevin O'Brien, *Crowd Simulation Incorporating Agent Psychological Models, Roles and Communication*, First International Workshop on Crowd Simulation (V-CROWDS '05) (2005).

[SGA+07] Avneesh Sud, Russell Gayle, Erik Andersen, Stephen Guy, Ming Lin, and Dinesh Manocha, *Real-time Navigation of Independent Agents Using Adaptive Roadmaps*, Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology (New York, NY, USA), VRST '07, Association for Computing Machinery, 2007, DOI 10.1145/1315184.1315201, pp. 99–106, ISBN 9781595938633.

[SGC04] Mankyu Sung, Michael Gleicher, and Stephen Chenney, *Scalable behaviors for crowd simulation*, Computer Graphics Forum **23** (2004), no. 3, 519–528, ISSN 1467-8659, DOI 10.1111/j.1467-8659.2004.00783.x.

[SHW+18] Yijun Shen, Joseph Henry, He Wang, Edmond S. L. Ho, Taku Komura, and Hubert P. H. Shum, *Data-Driven Crowd Motion Control With Multi-Touch Gestures*, Computer Graphics

Forum **37** (2018), no. 6, 382–394, ISSN 0167-7055, 1467-8659, DOI 10.1111/cgf.13333.

[Sno00] Greg Snook, *Simplified 3D Movement and Pathfinding Using Navigation Meshes*, Game Programming Gems (Mark A. DeLoura, ed.), Charles River Media, 2000, pp. 288–304, ISBN 1-58450-049-2.

[TCP06] Adrien Treuille, Seth Cooper, and Zoran Popović, *Continuum Crowds*, ACM Transactions on Graphics **25** (2006), no. 3, 1160–1168, ISSN 0730-0301, DOI 10.1145/1141911.1142008.

[TYK+09] Shigeo Takahashi, Kenichi Yoshida, Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Sung Yong Shin, *Spectral-Based Group Formation Control*, Computer Graphics Forum **28** (2009), no. 2, 639–648, ISSN 0167-7055, DOI 10.1111/j.1467-8659.2009.01404.x.

[UCT04] Branislav Ulicny, Pablo de Heras Ciechomski, and Daniel Thalmann, *Crowdbrush: Interactive Authoring of Real-time Crowd Scenes*, Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '04, Eurographics Association, 2004, DOI /10.1145/1028523.1028555, p. 243–252, ISBN 3-905673-14-2.

[UT01] Branislav Ulicny and Daniel Thalmann, *Crowd simulation for interactive virtual environments and VR training systems*, Computer Animation and Simulation 2001 (Nadia Magnenat-Thalmann and Daniel Thalmann, eds.), Springer Vienna, 2001, DOI 10.1007/978-3-7091-6240-8_15, pp. 163–170, ISBN 978-3-7091-6240-8.

[UT02] Branislav Ulicny and Daniel Thalmann, *Towards Interactive Real-Time Crowd Behavior Simulation*, Computer Graphics Forum **21** (2002), no. 4, 767–775, ISSN 0167-7055, DOI 10.1111/1467-8659.00634.

[vTCG11] Wouter G. van Toll, Atlas F. Cook, and Roland Geraerts, *Navigation Meshes for Realistic Multi-Layered Environments*, 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, DOI 10.1109/IROS.2011.6094790, pp. 3526–3532, ISBN 978-1-61284-456-5, 978-1-61284-454-1.

[vTCG12] Wouter G. van Toll, Atlas F. Cook, and Roland Geraerts, *A navigation mesh for dynamic environments*, Computer Animation and Virtual Worlds **23** (2012), no. 6, 535–546, ISSN 1546-4261, 1546-427X, DOI 10.1002/cav.1468.

[XJY+08] Jiayi Xu, Xiaogang Jin, Yizhou Yu, Tian Shen, and Mingdong Zhou, *Shape-constrained flock animation*, Computer Animation and Virtual Worlds **19** (2008), no. 3-4, 319–330, ISSN 1546-4261, DOI 10.1002/cav.231.

[XWY12] Mingliang Xu, Yunpeng Wu, and Yangdong Ye, *Smooth and Efficient Crowd Transformation*, Proceedings of the 20th ACM International Conference on Multimedia (New York, NY, USA), MM '12, Association for Computing Machinery, 2012, DOI 10.1145/2393347.2396415, pp. 1189–1192, ISBN 9781450310895.

[XWY+15] Mingliang Xu, Yunpeng Wu, Yangdong Ye, Illes Farkas, Hao Jiang, and Zhigang Deng, *Collective Crowd Formation Transform with Mutual Information-Based Runtime Feedback*, Computer Graphics Forum **34** (2015), no. 1, 60–73, ISSN 0167-7055, 1467-8659, DOI 10.1111/cgf.12459.

[YCP+08] Hengchin Yeh, Sean Curtis, Sachin Patil, Jur van den Berg, Dinesh

Manocha, and Ming Lin, *Composite Agents*, Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '08, Eurographics Association, 2008, pp. 39–47, ISBN 9783905674101.

[YMDHC+05] Barbara Yersin, Jonathan Maïm, Pablo De Heras Ciechomski, Sébastien Schertenleib, and Daniel Thalmann, *Steering a virtual crowd based on a semantically augmented navigation graph*, First International Workshop on Crowd Simulation (V-CROWDS '05), 2005.

[YMPT09] Barbara Yersin, Jonathan Maïm, Julien Pettré, and Daniel Thalmann, *Crowd Patches: Populating Large-scale Virtual Environments for Real-time Applications*, Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (New York, NY, USA), I3D '09, Association for Computing Machinery, 2009, DOI 10.1145/1507149.1507184, pp. 207–214, ISBN 9781605584294.

[YT07] Qinxin Yu and Demetri Terzopoulos, *A Decision Network Framework for the Behavioral Animation of Virtual Humans*, Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation (Goslar, DEU) (Dimitris Metaxas and Jovan Popovic, eds.), SCA '07, Eurographics Association, 2007, DOI 10.2312/SCA/SCA07/119-128, pp. 119–128, ISBN 978-3-905673-44-9.

[ZLD15] Peng Zhang, Hong Liu, and Yanhui Ding, *Crowd simulation based on constrained and controlled group formation*, The Visual Computer **31** (2015), no. 1, 5–18, ISSN 1432-2315, DOI 10.1007/s00371-013-0900-7.

[ZZC+14] Liping Zheng, Jianming Zhao, Yajun Cheng, Haibo Chen, Xiaoping Liu, and Wenping Wang, *Geometry-constrained crowd formation animation*, Computers & Graphics **38** (2014), 268–276, ISSN 0097-8493, DOI 10.1016/j.cag.2013.10.035.

Citation

Luis Rene Montana Gonzalez, and Steve Maddock *A Sketch-based Interface for Real-time Control of Crowd Simulations that incorporate Dynamic Knowledge*, Journal of Virtual Reality and Broadcasting, 16(2019), no. 3, August 2021, urn:nbn:de:0009-6-53490, DOI 10.48663/1860-2037/16.2019.3, ISSN 1860-2037.