DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Purpose-aware pedestrian flow visualization in 3D building models

Simon Brunner

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Purpose-aware pedestrian flow visualization in 3D building models

Zweckgebundene Visualisierung von Fußgängerströmen in 3D-Gebäudemodellen

Author:	Simon Brunner
Supervisor:	Prof. Dr.-Ing. Frank Petzold
Advisor:	Ata Zahedi M.Sc., Jimmy Abualdenien M.Sc.
Submission Date:	20.05.2022

I confirm that this master’s thesis is my own work and I have documented all sources and material used.

Munich, 20.05.2022 Simon Brunner

Acknowledgments

I would like to thank everyone who supported me during the development of this thesis.

First and foremost, I thank my supervisor Prof. Frank Petzold for accepting the topic of this thesis.

Then, I would like to thank my advisors Jimmy Abualdenien and Ata Zahedi for their continuous support and valuable feedback.

Furthermore, I would like to express my sincere thanks to the CEO of accu:rate, Dr. Angelika Kneidl, who made it possible to work on this topic and supported me in implementing the visualization.

Lastly, I would like to thank the Bundesministerium für Digitales und Verkehr for funding the BEYOND research project, in the context of which this thesis was created.

Abstract

This thesis analyzes what constitutes a 3D visualization of crowd simulations that supports analysts when investigating simulation results. A key requirement for such a visualization is a simple and clear representation that allows simulation results to be quickly and accurately understood. For that purpose, it employs findings from visual analytics, by collecting standard principles, as well as pitfalls, that should govern every visualization. Then, a proof-of-concept is implemented in the crowd simulation software crowd:it. Conducted in the course of the research project BEYOND, the implementation employs construction data that comes from Building Information Modeling (BIM) models and displays it in a three-dimensional view that joins the data with the simulation results. The implementation is evaluated in the means of a user study that was conducted with crowd:it customers, as well as other experts in the field of crowd simulations. The visualization was positively received as a tool to expand reports and convey structural information regarding to clients. For the analysis that is conducted by experts that commonly work with a model, the visualization was only perceived as supplementary to classical analysis. In the future, the proof-of-concept should be expanded with additional features that were out of the scope of the thesis, in order to make it viable for commercial use.

Zusammenfassung

Diese Arbeit untersucht, was eine 3D-Visualisierung von Personenstromsimulationen ausmacht, die Analysten bei der Untersuchung von Simulationsergebnissen unterstützt. Eine zentrale Anforderung an eine solche Visualisierung ist eine einfache und übersichtliche Darstellung, die es ermöglicht, Simulationsergebnisse schnell und präzise zu erfassen. Zu diesem Zweck werden Erkenntnisse aus dem Bereich der Visual Analytics genutzt, indem Grundprinzipien, aber auch Fallstricke gesammelt werden, die für jede Visualisierung gelten sollten. Anschließend wird ein Prototyp in der Personenstromsimulationssoftware crowd:it implementiert. Die im Rahmen des Forschungsprojekts BEYOND durchgeführte Implementierung verwendet Baupläne, die aus Building Information Modeling (BIM)-Modellen stammen, und stellt sie in einer dreidimensionalen Ansicht dar, die die Daten mit den Simulationsergebnissen verbindet. Die Implementierung wird im Rahmen einer Nutzerstudie evaluiert, die mit crowd:it-Kunden sowie weiteren Experten aus dem Bereich der Personenstromsimulation durchgeführt wurde. Die Visualisierung wurde als Werkzeug zur Erweiterung von Ergebnisberichten und zur Vermittlung von strukturellen Informationen gegenüber Kunden positiv aufgenommen. Für die Analyse, die von Experten, die regelmäßig mit dem Modell arbeiten, durchgeführt wird, wurde die Visualisierung nur als Ergänzung zur klassischen Analyse wahrgenommen. In Zukunft soll der Prototyp um weitere Funktionen erweitert werden, die außerhalb des Umfangs dieser Arbeit lagen, um ihn für den kommerziellen Einsatz nutzbar zu machen.

Contents

1 Introduction

Crowd simulations are an important part of modern design processes. While the construction plans used for crowd simulation are usually available as two-dimensional drafts, a three-dimensional representation is becoming increasingly important with the advent of Building Information Modeling (BIM), in which construction data is combined and visualized. BIM functions as an all-encompassing working practice into which data from every contributor involved in the project is entered. This also makes it possible to comprehensively anchor crowd simulations in the BIM process. One aspect of this anchoring is the visualization of simulations in three dimensions - both in the design phase, in which the parameters required for the simulation are set by an expert, and in the evaluation phase, in which the data generated by the simulation must be visually comprehended and evaluated. This thesis implements a three-dimensional visualization that displays both crowd simulations and BIM models in a combined view.

1.1 Motivation

With the rising computational power of graphics hardware, three-dimensional visualizations are increasingly important. Simulation visualization in general is a common topic in scientific literature. Visualizing crowd simulations in particular is both investigated in literature and available in commercial software.

Scientific literature mostly focuses on the simulation itself, using the visualization only as a rudimentary tool to display the results. When specifically focusing on visualization, rendering performance is the most prominent concern. Commercial software, on the other hand, often aims to create a visually pleasing visualization. This kind of visualization is useful if results are to be shown to stakeholders or clients, however, it may be disadvantageous when a qualified analyst tries to investigate the data that has been generated by the simulation.

This thesis aims to join the findings from scientific literature and commercial software to create a visualization that is applicable for commercial software, while also employing scientific principles to support an analyst or experienced user in investigating simulation results. In order to achieve this, it applies principles from the scientific field of visual analytics.

1.2 Goal

The 3D visualization is implemented as a proof of concept in the crowd simulation software crowd:it. Crowd:it is a commercial software that allows carrying out agent-based crowd simulations. It is developed by accu:rate GmbH[1]. So far, the software only features a two-dimensional visualization of the construction plans and simulation results. This thesis expands this by implementing and evaluating a basic 3D view according to the principles of visual analytics. At the time of writing this thesis, the author is employed at accu:rate GmbH.

In order to employ three-dimensional construction data, the visualization needs to integrate BIM models. At the time of writing this thesis, the company is involved in the research project BEYOND that is concerned with this matter. This thesis is carried out as part of this research project.

1.3 Outline

This chapter has so far set out the motivation for this work and what objectives are to be achieved.

Section 2 summarizes the most important technical details regarding crowd simulations in general and the simulation model that is used in crowd:it.

Section 3 outlines tasks and goals of the research project BEYOND that accu:rate is affiliated with.

Section 4 provides an overview of existing visualizations for crowd simulations. It presents exemplary implementations both from scientific literature, as well as commercial software other than crowd:it. This serves as a means to confirm the claims made in 1.1.

Section 5 summarizes essential findings from scientific literature regarding visual analytics. It presents basic concepts regarding visualization and interaction that are applied in the implementation. It collects standard principles that should be employed, as well as pitfalls that should be avoided, in a checklist-style manner.

Section 6 then presents the implementation of the 3D visualization. It explains how and why specific decisions were made. For that purpose, it commonly refers to the concepts introduced in Section 5. The section also explains technical details regarding the implementation, as these often directly influenced the design of the visualization.

Section 7 evaluates the implementation. For that purpose, a user study was carried out that gathered feedback from crowd:it customers and other experts in the field of crowd simulations.

Section 8 summarizes the thesis and the results. Finally, some suggestions for further improvement and development are presented.

2 Agent-based crowd simulation

2.1 Overview

Crowd simulations aim to model the real world as closely as possible, in order to analyze a scenario and make accurate assessments, e.g. regarding egress times. A model needs to comprise two main components: Geometry and Behaviour.

On one hand, a model needs to incorporate the structure of a scenario, i.e. geometry like walls, stairs, and escalators. This information is highly available through construction plans. Therefore, a simulation can incorporate geometry to a very fine degree. Historically, crowd simulations have mostly been two-dimensional. With the advent of BIM, many models have been adapted to include 2.5D or 3D data, such as stairs and escalators, among others.

On the other hand, a model needs to take human behavior into account. It needs to consider pathfinding on a large scale, i.e. how a person navigates towards its destination. Some models also incorporate the movement of a person on a small scale, down to the individual steps that a pedestrian makes.

All this information may be important for 3D Visualizations. Some information is used by all crowd simulation models, like the geometry of the scenario. Then, there is information that is specific to or generated by certain models. In order to develop a clear and readable visualization, one needs to decide, what information needs to be included in the visualization. Therefore, it is important to have a profound understanding of the used model.

The following sections will describe the basic structure of crowd simulations. Then, an overview of existing models for crowd simulation will be given. Finally, the model used in the software crowd:it will be explained in more detail.

2.2 Composition of a crowd simulation

Every simulation contains geometry, like walls, that obstructs pedestrians in their movement. It may also contain geometry that interacts with pedestrians other than blocking their movement. Stairs, for instance, alter the movement of pedestrians, by slowing them down and changing their behavior when evading other pedestrians. Then some objects allow pedestrians to stop completely, while they are moved by the object itself – for example, escalators or elevators. Crowd simulation models need to account for objects like these.

A simulation also needs to contain objects that carry navigational information. Usually, pedestrians appear in a source (origin) and disappear in a sink (destination). A scenario may contain multiple origins and destinations. It may also contain intermediate destinations that should be targeted by pedestrians before they head for their final destination. These objects are set up by a human modeler, who assigns pedestrians to a path that contains the specific destination that they should target. Usually, the model supports the modeler by offering a wide toolset that allows to create highly complex setups.

Lastly, a simulation contains the pedestrians (agents) themselves. They influence each other, in that agents may repel or attract one another. Usually, agents are not allowed to overlap. The exact effects that agents exert on each other depend on the details of the model that is used.

2.3 Classifying different models for crowd simulation

There exists a wide variety of crowd simulation models, which aim to reproduce realistic behaviour, while still being as simple as possible. In order to categorize them, Chraibi et al. [2] introduce three layers of agent behavior: The strategic level, the tactical level, and the operational level. On the strategic level, agents decide, which destination (and possibly intermediate destinations) they should target. The specific routing is applied on the tactical level, i.e. the agent identifies the pathway they need to walk to reach their destination. These two levels are usually composed by a modeler, who sets up origins and destinations.

The third layer of agent behavior is the operational level. This layer concerns the immediate short-term walking behavior of agents. It ensures that agents step towards their destination while avoiding collision with obstacles and other agents. Crowd simulation models differ mostly in their description of the operational level. Therefore, it is reasonable to differentiate them according to criteria regarding the operational level.

Chraibi et al. separate the simulation models according to seven characteristics: Microscopic vs. macroscopic, discrete vs. continuous, deterministic vs. stochastic, rule-based vs. acceleration-based vs. velocity-based, heuristic vs first-principles, and high vs. low fidelity. When classifying the simulation model used within crowd:it, it is best described as a microscopic, continuous model. The other characteristics play a subordinate role. The following section elaborates on this classification and provides a basic description of crowd:its simulation model.

2.4 Crowd simulation with crowd:it

The implementation of the concepts described in the course of this thesis is done exemplary within the simulation software crowd:it. It is necessary to gain a basic understanding of the simulation model implemented in the software, in order to evaluate the decisions made regarding the content of the 3D Visualization. This section classifies the model at hand according to the characteristics described in the previous section. Then, it provides a short description regarding crowd:its implementation of the operational level.

Crowd:it is an agent-based simulation model. In an agent-based, i.e. microscopic simulation, each pedestrian is simulated as a separate entity. Each agent carries individual properties that may vary according to a given distribution. These properties include size and velocity, amongst others. A macroscopic simulation, on the other hand, aggregates these properties into averages. In recent years, there has been an effort to expand crowd:it with aspects of both micro - and macroscopic models into a hybrid model. This will not be addressed in the 3D Visualization, as this thesis focuses on microscopic simulation models.

Crowd:it is also time and space continuous, i.e. attributes like the position of an agent might take any real value, instead of only integer values, like it is the case with discrete models. It is important to note that crowd:it historically has been time-discrete: The simulation is divided into separate timesteps in which all agents move simultaneously. Recently, this has been changed to an event-based architecture, where the step of an agent is an event that may be scheduled at any time. However, the simulation output is still discretized into timesteps, for the historic reasons noted above. This is an important factor regarding the visualization. The exact effects of this will be discussed in section 6.12

The last substantial component regarding crowd:its simulation model is the mechanism that determines the walking behavior of agents. Crowd:it implements a modified version of the Optimal Steps Model (OSM)[3, 4]. The OSM assigns a scalar potential to each point of the two-dimensional plane: The next target of an agent exerts an attractive force (i.e. negative potential), while obstacles and agents result in repelling forces (i.e. positive potential). These are summed into a total potential for each point. For each step that an agent makes, they sample a circular disk around their current position and choose the point with the lowest potential.

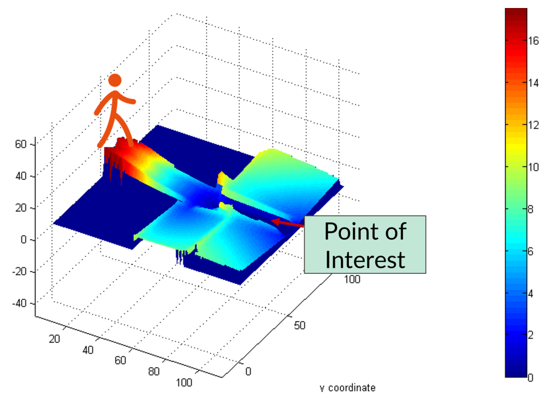

Figure 2.1: Visualization of the potential - the arrow marks the most attractive position (image provided by accu:rate [1])

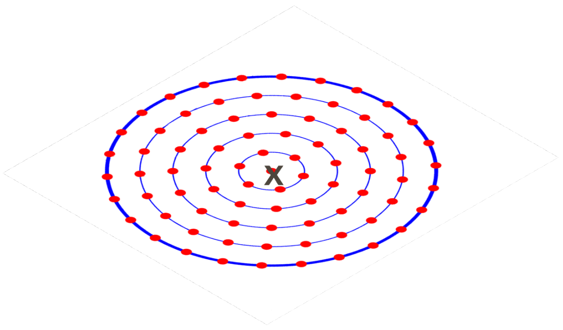

Figure 2.2: Sample points on a circular disc around the agent (image provided by accu:rate [1])

It is important to note that the vanilla OSM inherently only contains two-dimensional information about the position of an agent. Crowd:it expands this by introducing stairs, escalators, and elevators which carry 3D information. Beyond that, there are efforts to fully integrate 3D-BIM Models into the simulation model. This is elaborated further in section 3.

3 The BEYOND research project

3.1 Overview

This thesis is written in the course of the research project BEYOND. The project aims to make construction planning more cost-efficient and fast, while at the same time improving quality. For this purpose, Building Information Modeling (BIM) and crowd simulations will be combined: "KI-basierte Optimierung in frühen Entwurfsphasen von Gebäuden mithilfe der Analyse von Personenströmen in intelligenten Gebäudemodellen" (AI-based optimization in early design phases of buildings using crowd simulations in smart building models). The participating partners are accu:rate GmbH, TUM, DB Netz AG and SSF Ingenieure AG. This section provides a brief description of the basic process of BEYOND and outlines the contribution this thesis will make to the research project.

The project integrates crowd simulations into the planning process of buildings at an early stage. Crowd simulations are computationally intensive tools, rendering them ineffective in a design phase that is subject to frequent changes. In order to make it possible to simulate a vast number of variations in early design phases of the building, the project aims to utilize deep learning. The deep learning model developed in the course of the project analyzes the parameters that come from the digital model of the building and makes predictions regarding the pedestrian flow. This goal is achieved by means of seven work packages ("Arbeitspaket") that are conducted by the project partners. The following sections summarize the work conducted during each work package, in order to provide a quick overview of the project.

3.2 Work packages

Requirements Analysis

This first work package took the form of a kickoff workshop that served to identify the requirements for the project and match them with technical feasibilities. Here the baseline for the following work packages was established - the results of this work package are reflected in the following work packages.

BIM model analysis and extension as a basis for parametric studies

This work package aims to create an algorithm that generates parametric ifc models of railroad stations. Besides the development of the algorithm itself, it also defines a good selection of parameters and boundary conditions used for the development of the algorithm. Parametric means that the algorithm may be parametrized with input values that determine the structure of the building. One conceivable parameter, for instance, would allow setting the desired number of rail tracks: The resulting ifc model will contain the specified number of rails, while the algorithm makes sure to obey the given boundary conditions and adjust the surrounding geometry accordingly.

Interfaces for bidirectional data exchange between the BIM authoring tool and the simulation tool

To ensure the compatibility of the applications developed in the individual work packages, adequate interfaces have to be defined. This work package establishes a bidirectional interface between the parametric model and the simulator, in order to extract geometric and semantic data from the model and transfer simulation results back into the model after a simulation has been carried out.

Extension of existing simulation models to enable the automated process

In order to utilize crowd simulations, BIM models need to be converted to a format that is suitable for crowd simulations. This work package implements a converter that converts an IFC-File that was defined and generated in the course of the previous work packages into such a format. In practice, the file format of crowd:it serves as a proof of concept, here.

The necessity to be able to generate large datasets for machine learning exerts some additional requirements on the converter: It needs to be able to convert a large number of IFC files that are ready for a simulation, without the obligation of additional manual operations inside crowd:it. The converter makes use of the interface defined in work package 3 in order to set up a finished project that can immediately be simulated.

To complement this, the simulation kernel of crowd:it is also adjusted to be able to fully utilize the information available through IFC.

Lastly, the simulation results should be returned to the construction model. This is implemented in the course of this thesis: The construction model is viewed in a joined view with the simulation results.

Application of machine learning approaches and integration into simulation models

Crowd simulations usually require a considerable amount of manual effort and high computational capacities. This work package aims to develop a deep learning model that predicts simulation results in order to enable a straightforward comparison between variants of the building.

To generate a large dataset for machine learning, the parametric model from work package 2 and the IFC-Converter from work package 4 are utilized: By varying the parameters offered by the parametric models, a set of models is generated, which is then converted and simulated. The resulting simulations may either serve as a training set for the deep learning model or as a means to verify the model. Once the deep learning model is established, a simple user interface is developed that allows employing the AI in practice.

Extension of the IFC standard

IFC is an exchange format for BIM models. This work package aims to extend the IFC standard by introducing the results of this project to the international working group “MVD Proposal for Fire Safety Engineering and Occupant Movement Analysis”.

The standardization enables seamless communication between various BIM-Tools and crowd simulators.

Dissemination: Testing of the developed prototype

This work package intends to evaluate the prototype that was developed in the course of the project. The prototype is presented to a professional audience from DB in order to gather feedback and assess the results of the project. Also, the prototype is tested for practicability.

4 Existing 3D Crowd Visualizations

There exists a variety of crowd visualizations that have been published in scientific literature, as well as in commercial software. Understanding the concepts that have been established with these existing visualizations is essential when creating a new implementation. This section briefly summarizes the findings from these visualizations.

4.1 Visualizations from Scientific Literature

In "A simulation as a service methodology with application for crowd modeling, simulation and visualization", Wang et al.[5] developed a cell-based crowd simulation that integrates BIM tools. The components that were developed comprise, amongst others, Cell-DEVS modeling, which is used to create the crowd model, a data collection component that extracts data from IFC-files, and a 3D visualization.

The 3D-visualization tool is implemented as a prototype using the computer graphics program 3ds Max[6]. A separate parser loads simulation results into the program. 3ds Max offers a large number of advanced tools for rendering and animation. This allowed them to prepare animations for the agents and employ 3ds Max to create smooth transitions between the animations. The basic agent model provided by the tool is a simple cuboid. If direction information is available, the tool also provides a realistic agent model, as well as an arrow model. The GUI offers options to show and hide agents and floors, and to focus on a specific area or agent.

The major strength of this work is that it assembles many components, like collecting the data from the BIM model, simulation and visualization into one system. The system offers different solutions for each component. Furthermore, the user can replace or modify individual components. However, from the viewpoint of a visualization, it is a mere prototype. It allows to visually comprehend the scene but does not offer advanced tools to analyze the data.

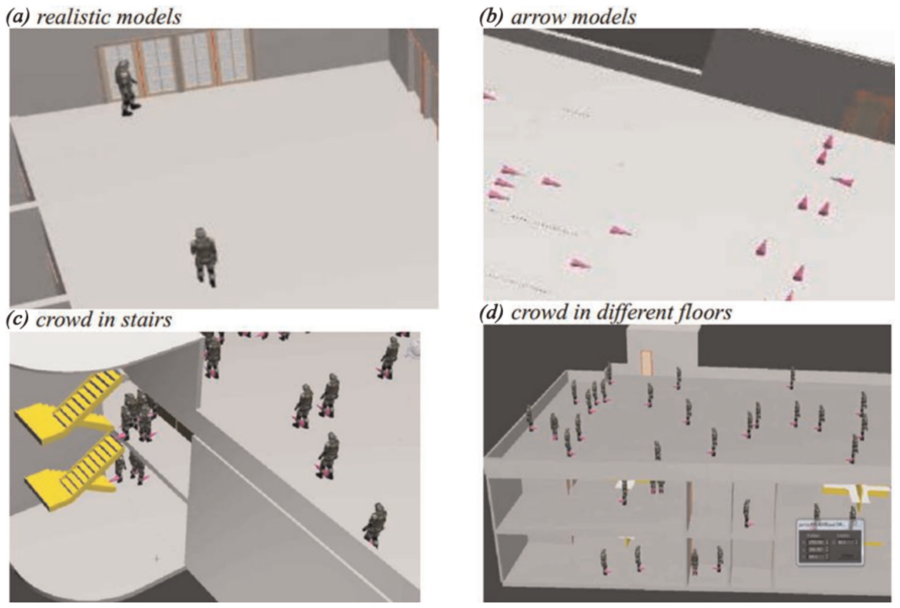

Figure 4.1: Realistic and arrow models [5]

In "Crowd Data Visualization and Simulation", Diaz et al.[7] created a real-time crowd simulation that employs GPU-processing and GPS data to simulate real cities. They employ an agent-based approach to crowd simulation: Preprocessed trajectories that are extracted from the GPS data are used to steer agents, which also adhere to a social force model.

The visualization shows realistic, animated agent models. The authors use dynamic materials to achieve variety. The scene obtains a realistic look by applying textures to the geometry.

The main target and simultaneously the most important asset of the work was to create a computationally efficient simulation and visualization. On their target system, a framerate of 35.71 was achieved when rendering one thousand characters. Due to this focus, the visualization is only a rudimentary tool that is not aimed at a comprehensive analysis of the simulation results.

In the course of his doctoral thesis "Crowd Simulation and Visualization", Perez[8] developed a visualization for a microscopic agent-based crowd simulation that employs a social force model. The simulation is performed on a High-Performace Computing cluster. The visualization aims to show the results without the need for simplification,

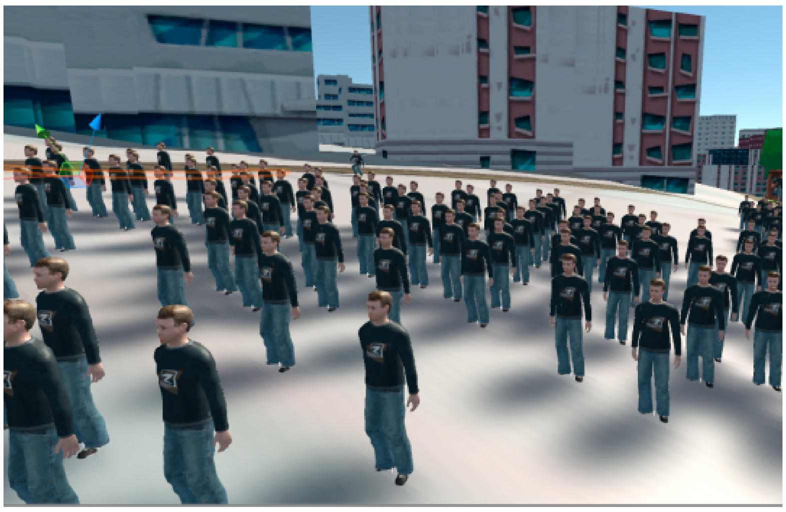

Figure 4.2: Realistic agent models [7]

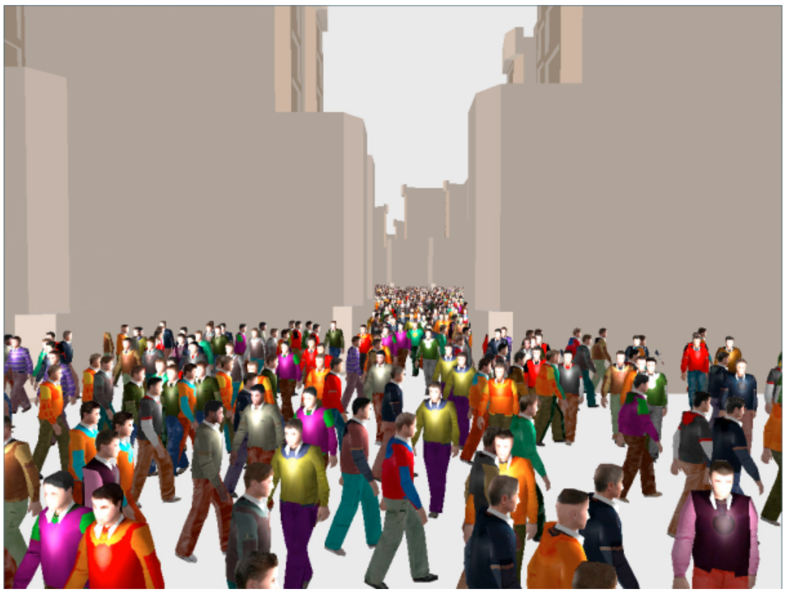

Figure 4.3: Variety with dynamic materials [7]

which would hide relevant details. The author developed three different configurations for that purpose. The In-Situ configuration is executed on a client that is connected to the same LAN as the cluster. The client receives images of the simulation results. This enables to simulate and visualize very large simulations. The Streaming configuration is aimed at systems that are not directly connected to the LAN of the cluster. It streams simulation data to the client, which then performs the rendering. Lastly, the Web configuration allows viewing the simulation results on any device, as the application streams the rendered images to the client. The author compares this to current cloud gaming technology.

The visualization itself employs realistic agent models and allows to render a realistic environment by accessing different data sources. The author argues that this is an effective technique to prepare users to face real events. However, the implementation focuses on visualizing the results of the simulation. Therefore, it does not offer advanced means to interact with the visualization that allows to conduct a thorough analysis of the results.

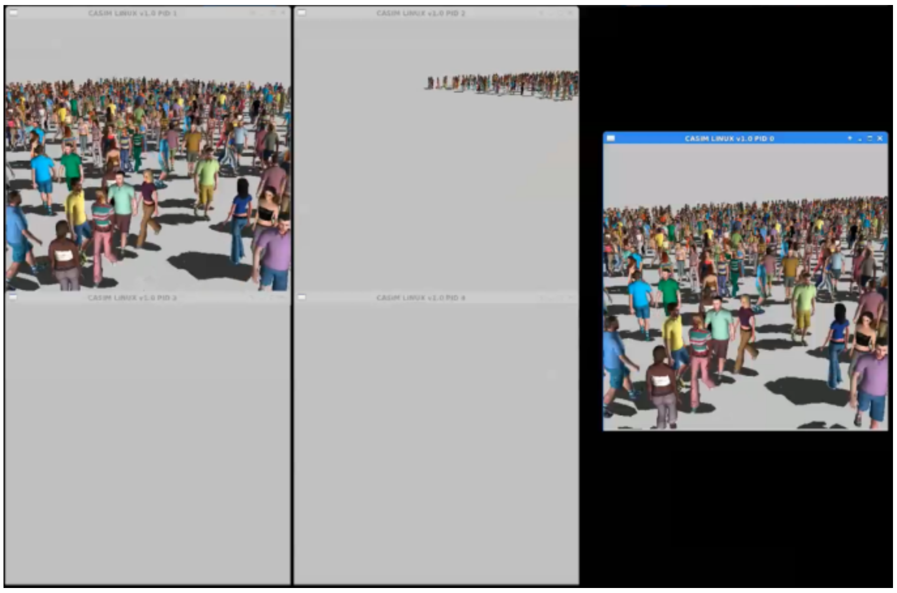

Figure 4.4: Agent models [8]

4.2 Visualization from Commercial Software: Pathfinder

Pathfinder is an agent-based crowd simulation software developed by Thunderhead Engineering[9]. Pathfinders 3D Visualization is highlighted on the company's website as a prominent feature and selling point. Therefore, it is an adequate candidate to evaluate the current state of 3D visualizations for crowd simulations.

As a commercial software, pathfinder supports an abundance of features. It allows to import several file formats, including IFC to import BIM models. Regarding the analysis of simulation results, pathfinder offers various tools to investigate the data from the simulation. Charts allow plotting output data, for example, flow rates. For the visualization, the company prominently advertises realistic, animated human models that may be changed by importing custom models. However, the software also features a simplistic view that allows to visualize agents as simplistic cylinders. Additional tools for analysis allow to integrate contour plots into the visualization: These color the floors of a model depending on some specified data. This is comparable to a heatmap. Apart from visualizing agent movement, pathfinder also allows to import and visualize smoke and fire data.

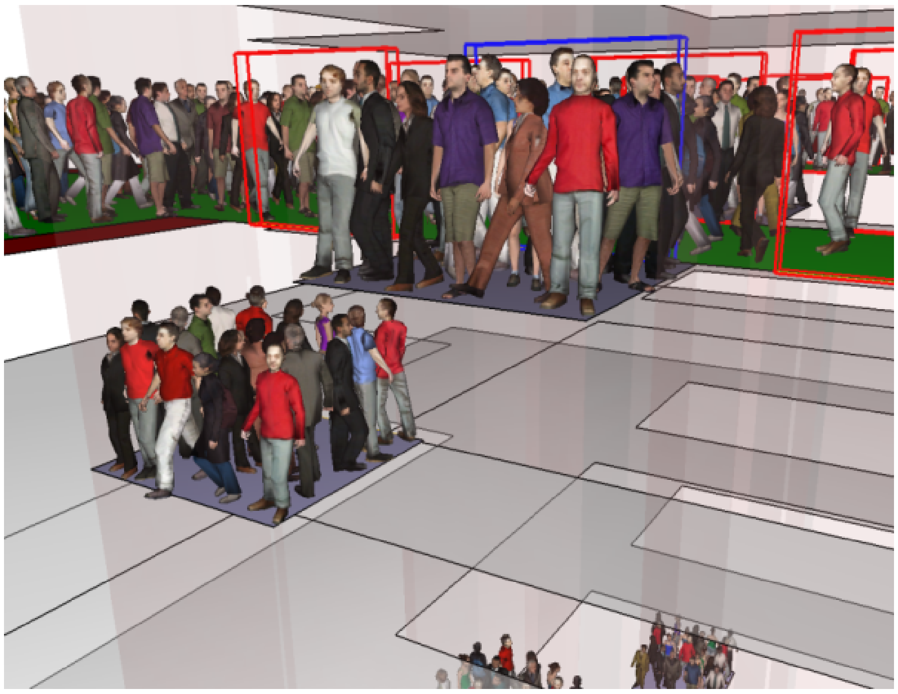

Figure 4.5: Realistic Agents in Pathfinder [10]

Pathfinders 3D visualization is an advanced tool for visualizing crowd simulations. It provides clear visuals, while the software itself still offers many tools for analyzing the results. The focus on pretty visuals is a valid approach to generate a pleasing visualization when presenting the results to stakeholders and clients. Employing the visualization from an analytical point of view is possible. Still, implementing more tools for analysis and interaction is the main improvement that was identified in the course of this thesis.

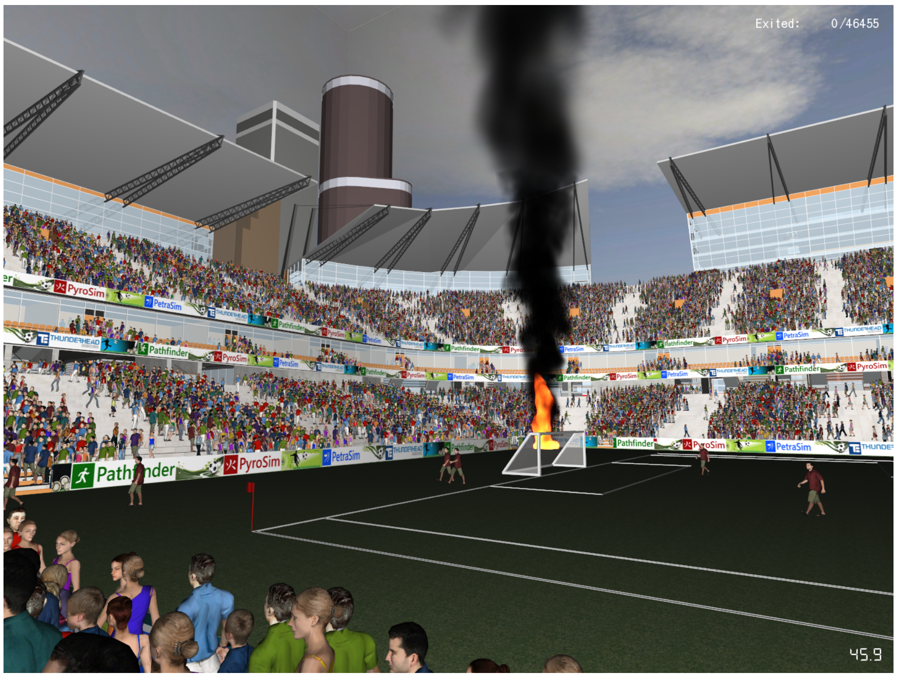

Figure 4.6: Visualization of Smoke and Fire [11]

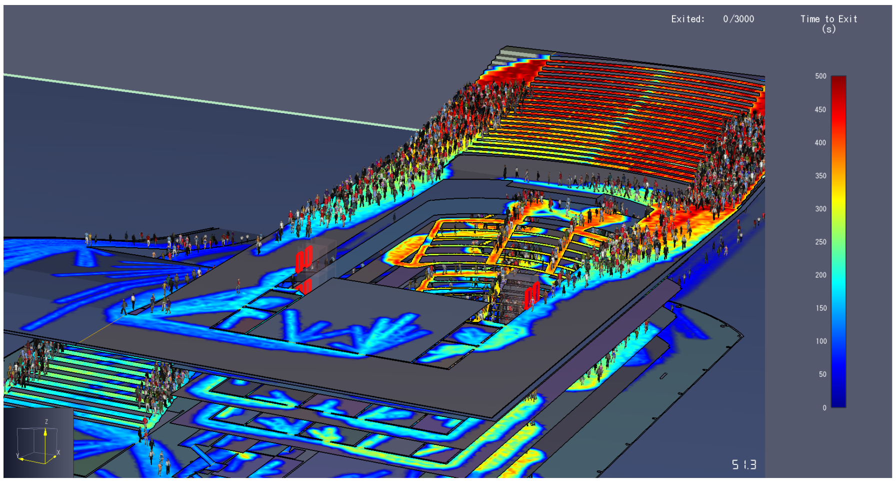

Figure 4.7: 3D Heatmap in Pathfinder [12]

5 Visual Analytics

5.1 Purpose

The implementation that is described in section 6 employs techniques from the field of visual analytics. This aims to ensure a clear, simple, and readable visualization that enables the user to analyze the data that was generated during the simulation. This section summarizes the theoretical foundations regarding visual analytics, by supplying an overview of common concepts and techniques. On one hand, this lays out a theoretical and scientific baseline that the implementation must adhere to. On the other hand, this aims to justify the decisions made regarding certain implementation details. The explanations in section 6 will therefore commonly refer to the concepts that are explained in this section.

5.2 Overview

Visual Analytics is an interdisciplinary scientific field that is concerned with the visualization of large datasets. Historically, it is "an outgrowth of the fields of scientific and information visualization"[13]. Both of these fields are concerned with visualizing data. Keim et al.[14] distinguish the two as follows: Scientific visualization applications deal with visualizing large amounts of data coming from sensors or simulations. The authors cite flow visualizations or slicing techniques for medical illustrations as examples. Information visualization, on the other hand, is defined "as the communication of abstract data relevant in terms of action through the use of interactive interfaces"[14][p.78]. However, they point out that visual analytics encompasses more than just visualization: Visual analytics comprises both visualization and analysis. This means, in particular, that among many ways to display the data, a visualization must be developed that both represents the data properly and allows a detailed analysis.

The following definition of Visual Analytics describes this requirement through two sub-areas:

"Visual analytics is the science and practice of analytical reasoning by combining computational processing with visualisation. These are tightly-coupled using interactive techniques so that each informs the other."[15][p.3]

According to this definition, visual analytics is composed of two components: Computational methods and interactive visual interfaces. Computational methods "do what computers are good at"[15][p.3]: They process data by transforming or summarizing it. They need to be run by an analyst, as they rely on a formal specification of the task that needs to be performed on the input data. Interactive visual interfaces allow such an analyst to "do what human analysts are good at"[15][p.3]: They evaluate the results of the computational methods and implement an iterative cycle of applying computations and interpreting the emerging results. This results in bidirectional communication: On one hand, the visual interface needs to visually represent the results of the computations in a way that enables the human analyst to properly interpret the results. On the other hand, it needs to allow the analyst to adjust the computational methods, for example by fine-tuning parameters. Therefore, visual analytics defines an interaction with the computer, instead of simply visualizing results.

5.3 Common Elements in Visual Analytics

The book "Visual Analytics for Data Scientists"[16, 17] provides a comprehensive yet straightforward collection of general concepts from visual analytics. Therefore, the summary in this section will mainly be based on this book.

The authors divide the principles of interactive visualization into two main categories: Visualization and interaction. The latter of those two distinguishes visual analytics from plain visualization. The following sections will describe concepts presented within the book, regarding both visualization and interaction.

5.3.1 Visual variables

Based on the works of Bertin[18], the authors identify several visual variables employed by a visualization to encode data: position, size, value (lightness), color, texture, orientation, and shape, as well as dynamic visual variables like motion or animation.

Each visual variable can be utilized to represent a certain aspect of the data at hand.

Visual marks are used to represent a data item that is placed within the space spanned by the position visual variable. Visual marks can be classified by their dimensionality, i.e. points (0D), lines (1D), areas (2D), and volumes (3D).

Each of the visual variables can be categorized by its perceptual properties. The authors cite the following categories from Bertins work:

Ordering: Variables within this category can be ordered according to some natural order. Positional values, for example, can be ordered along an axis.

Distance: Comprises variables from the *Ordering* category that also allows to easily determine a distance between values. The authors take the difference between the length of two bars as an example. On the other hand, it is much harder, if not impossible, to visually determine a distance between two colors.

Ratio: This is very similar to the *Distance* category. It allows determining proportions between values, for example, a line being twice as long as another line.

Selectivity: Comprises variables where it is easy to distinct individual values of a variable, like, for example, color.

Associativity: Encompasses variables that allow to easily focus on multiple marks with the same value, while neglecting marks with different values.

Figure 5.1 provides practical examples that allow to easily comprehend these categories. The *Distance* and *Ratio* categories are combined as *Quantitative*.

Values of visual variables are related to each other through one or multiple of these categories. When constructing a visualization of some data, it is decisive to take into account that points of data may be linked with each other according to the same relationships. The authors infer from this what they call the “fundamental rule of visualization”: It is indispensable “to strive at representing components of data by visual variables so that the relationships existing between the values of the visual variables correspond to the relationships existing between the elements of the data components.”[16][p.59]

In order to apply this principle, the authors introduce three levels of organization for a data component: *Nominal* (which corresponds to associative and selective variables), *Ordinal* (corresponding to ordered variables), and *Numeric* (corresponding to quantita-

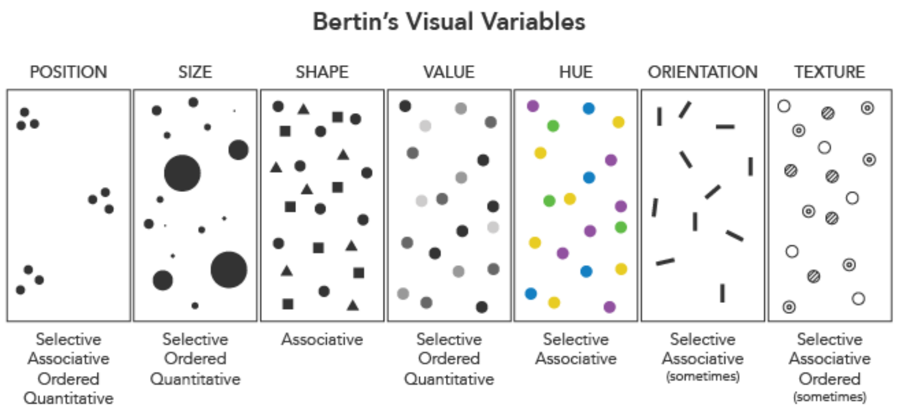
Bertin's Visual Variables
POSITION
SIZE
SHAPE
VALUE
HUE
ORIENTATION
TEXTURE
Selective Associative Ordered Quantitative
Selective Ordered Quantitative
Associative
Selective Ordered Quantitative
Selective Associative
Selective Associative (sometimes)
Selective Associative Ordered (sometimes)

Figure 5.1: Visual variables [19] as cited in [16]

tive variables). For example, a numeric data component that constitutes ratios between data points must be visualized by a visual variable from the *Ratio* category, like size.

The authors point out another constraint on the choice between visual variables: A visual variable must encompass at least as many visually distinguishable values as there are data points in the corresponding data component. If this is neglected, an analyst might confuse data points by mistake.

5.3.2 Standard principles for visualizations

The authors provide nine "general principles of visualization" that should be followed when creating a visualization. These will be listed here together with a short explanatory description. Section 6, which is concerned with the concrete implementation that was developed in the course of this thesis, will then refer to these principles, where applicable.

1. "Utilise space at first": The authors identify space, i.e. the positional visual variable, as the most expressive among all visual variables.

2. "Respect properties": This refers to the "fundamental rule of visualization", in that properties of visual variables must match relations between data items.

3. "Respect semantics": This prompts one to respect common knowledge or knowledge that is assumed to be known by the user.

4. “Enable seeing the whole”: A visualization should make it possible to get a complete overview of all data while avoiding creating visual clutter.

5. “Include titles, labels, and legends”: Allows for correctly interpreting the visualization that is shown.

6. “Avoid excessive display ink”: Decorations or other distracting components should be avoided, as they clutter the display without providing relevant information.

7. “Consider employing redundancy”: This principle states that there may be cases where it is beneficial to use multiple visual variables to encode a single data component. This intends to increase display readability or pattern recognition. The authors point out that this does not contradict the principle to “avoid excessive display ink”, as the redundant visualization has a purpose, rather than adding irrelevant visual elements.

8. “Enable looking at data from multiple perspectives”: This principle features two aspects. On one hand, this refers to the interaction 5.3.4 aspect of Visual Analytics, by offering different perspectives via interaction. On the other hand, this points to the necessity to offer multiple representations when the number of data components exceeds the number of available visual variables.

9. “Rely on interactivity”: This also hints at the interaction aspect of Visual Analytics. One example that the authors point out, is to utilize interaction to enable an analyst to access exact data values that are otherwise represented by suboptimal visual variables.

5.3.3 Limitations and pitfalls

The authors point out several possible issues of a visualization. Some of them are limitations, where compromises need to be made in order to get rid of them. Others are common pitfalls that should be avoided entirely. Critically evaluating a visualization may help identify these.

This section will list all limitations provided by the authors and provide a short description for each of them. The section regarding the implementation will then refer to these, whenever one of the limitations turned out to be a factor in practice.

1. "Implicitly defined and unchecked assumptions": This issue arises when a visual representation inherits assumptions from a statistical summary. This is most often the consequence of a poor choice of a visualization technique. An example provided by the authors is a box plot that is misleadingly used to represent a multimodal distribution. Resolving this issue usually simply involves switching to a different visualization technique.

2. "Lack of visual resolution": With larger data sets, this problem becomes increasingly relevant, since it becomes impossible to show all data items with limited display space. Introducing zooming and panning may help with this issue, accepting that it is infeasible to visualize all data items simultaneously. Another option to resolve the issue is to utilize aggregations or data reduction.

3. "Visual clutter": This arises when a visualization is overloaded due to an abundance of data. It may also arise if the visualization is arranged poorly, by choosing improper marks or decorations to represent the data. The authors suggest either combining a closeup view with an aggregated view or employing data reduction methods.

4. "Obvious patterns obliterate more interesting patterns": Even when it is ensured that a visualization is clear of visual clutter, there may still be an issue with expected patterns obscuring some of the more interesting patterns. This can be resolved by normalizing the data according to the expected property.

5. "False patterns": False patterns are usually artifacts that emerge when trying to create an aesthetically more pleasing visualization, like using smoothing techniques. A poor choice of marks or visual variables may also lead to this issue.

6. "Slow rendering for large datasets": This can be resolved by employing data reduction.

One further pitfall that is particularly interesting for this thesis is formulated in "Visualization Analysis and Design"[20]: "No unjustified 3D". The author argues that there are many cases where it is sufficient or even superficial to limit a visualization to two dimensions. A 3D visualization comes with many issues, for example, perspective distortion, the possibility of occluding important information, and an additional cognitive load to understand complex scenes. These are all costs of depth cues, i.e. cues that are needed in order to perceive depth information. One notable example of a depth cue is shadows: They are needed in order to infer information regarding depth, structure, and the height of an object with regard to the ground plane. On the flipside, shadows generate visual clutter that may distract the viewer. Therefore, the author suggests employing 3D visualizations only for data that comprises inherently three-dimensional structures, as is often the case with spatial data.

5.3.4 Interaction

The authors provide an extensive list of types of interaction with a visualization. This section summarizes the most significant techniques among these in the context of 3D representations.

The authors sort these techniques into five categories:

1. Changing data representation.
2. Focusing and getting details.
3. Data transformation.
4. Data selection and filtering.
5. Finding corresponding information pieces in multiple views.

It is important to keep these in mind when designing the interaction with a visualization, as they convey the purpose of an interaction.

Zooming and panning

There exists a large number of techniques that may be used to focus on some detail or a sub-portion of the data that is displayed. Among the most commonly used techniques are zooming, which shows a detail in a larger screen area than before, and panning, which allows to shift the section of the data that is shown.

An additional technique that is related to zooming and panning is rotation. This especially gains in importance when dealing with a 3D visualization: The ability to rotate the displayed data is indispensable. In a 3D environment, data marks commonly occlude each other. One approach to resolve this is by using rotations to get different perspectives on the same data.

One slightly more sophisticated application of zooming and panning is what the authors call focus+context, where some part of the data is shown in a zoomed view, while a separate view shows the complete dataset. When applied to spatial information, this is comparable to a "minimap", but this approach may just as well be applied to e.g. time intervals.

Hovering + popup window

Another common technique aimed to focus on detail allows viewing exact data values by displaying a popup window when hovering over a specific area of the display. This also opens up possibilities to display a lot more apart from exact values from the dataset, like showing results from calculations or data aggregation, or even yet another visualization.

Color re-scaling

Color re-scaling is a technique that is applicable if the data is represented using the color visual variable. Color re-scaling allows viewing a subportion of the data using the full-color spectrum, resulting in increased visibility for individual values. One specific application of this is converting a sequential color scale to a diverging color scale by specifying a reference value, as displayed in figure 5.2.

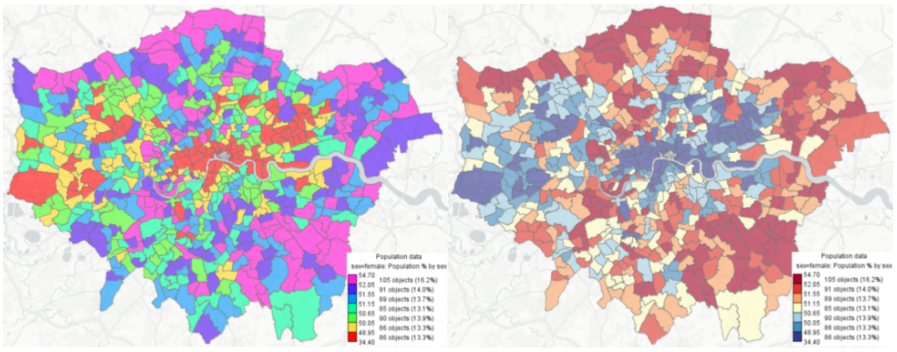

Figure 5.2: Color re-scaling applied to a map [16]

Selection and filtering

Selection and filtering are two complementary techniques to reduce the data that is shown. Selection means actively selecting data that should be visualized exclusively. Filtering is the opposite, i.e. removing data from the visualization. Both of these can be applied concerning certain properties: Attribute values of data points, position in time or space, or relationship to other data points. It is also possible to apply these directly to certain data points – the authors call this an entity-based filter.

Highlighting

Highlighting is a special application of a selection. A selected subset of the data is emphasized in the visualization, for example, by utilizing a different color. The remaining dataset stays visible. There exist many operations to achieve this, however, one technique noted by the authors is brushing. Brushing allows selecting data items by hovering or dragging the cursor over the display, or by clicking individual items.

The highlight must be applied to all views of the data if a visualization involves multiple views of the same data points. This allows one to orient oneself and link the data items between the individual views.

Common symbolism

Common symbolism is a technique used to identify corresponding information pieces in multiple views. One example of this is highlighting, as described above. In general, matching colors or visual marks between multiple views are examples of common symbolism.

Data transformation

Transforming data may also be utilized as a means of changing how the data is visualized. Aggregation, for example, reduces the amount of data by getting rid of details, which in turn allows gaining a general overview of the data that is available. A special case of this is discretization, which maps continuous values into fixed intervals.

Heatmaps in crowd:it are an example of this. Heatmaps, in general, provide a 2-dimensional color coding, representing some numeric distribution. Heatmaps may be continuous, but in crowd:it, they are used as a cell-based (i.e. discretized) aggregation. 6.19 shows a heatmap that was created in the default 2D-View of crowd:it.

Another common data transformation is smoothing. It allows getting rid of negligible fluctuations, in order to highlight meaningful trends and patterns. A basic example is taking the mean of the available data points, but there exist many more sophisticated smoothing implementations.

Changing data representation

As described in 5.3.1, data is represented in the means of visual variables. Consequently, changing the representation of data is usually accomplished by changing the visual variable that is used to represent the data. These kinds of transformations allow an analyst to view the data in a different context.

An example of this is a time plot, which employs the visual variable position to decode data entries as line chart. Here, changing the data representation can be achieved by changing the underlying type of chart that utilizes a different visual variable. One exemplary suggestion by the authors transforms the mentioned line chart to a mosaic plot, where the data is encoded using the color and value visual variables.

6 Implementation

6.1 Sample scenario

All examples that supplement the description of the implementation details are taken from the same sample scenario. This allows comparing the examples with each other. Also, perceiving the examples in a familiar scenario eases understanding. However, the implementation details apply to all scenarios compatible with crowd:it. The visualization was continuously evaluated against varying scenarios during the development process.

The sample scenario is a result of the parametric modeling process of BEYOND. It resembles a rudimentary train station. Figure 6.1 shows the three floors of the model in the 2D-View of crowd:it. The lowest floor represents the actual departure platform. Agents appear inside the red "trains" (i.e. origins) and walk towards several stairs and escalators. They move up to an intermediate floor, where they take yet another set of stairs and escalators to arrive at the top floor. The agents disappear, once they reach the green destinations.

6.2 Framework

The visualization was developed within the software crowd:it. This comes with the advantage that there is no need to implement a framework for the visualization that, for example, deals with reading the simulation file. Therefore, the visualization directly builds on top of the crowd:it architecture. It is integrated into crowd:its rendering pipeline alongside the 2D visualization. crowd:it is written in Java 11.

3D rendering was done using libGDX[21]. LibGDX is a game development framework based on OpenGL. It comes with modules for graphics (providing access to OpenGL ES 2.0[22]), input handling, math, and physics. Furthermore, it contains modules regarding

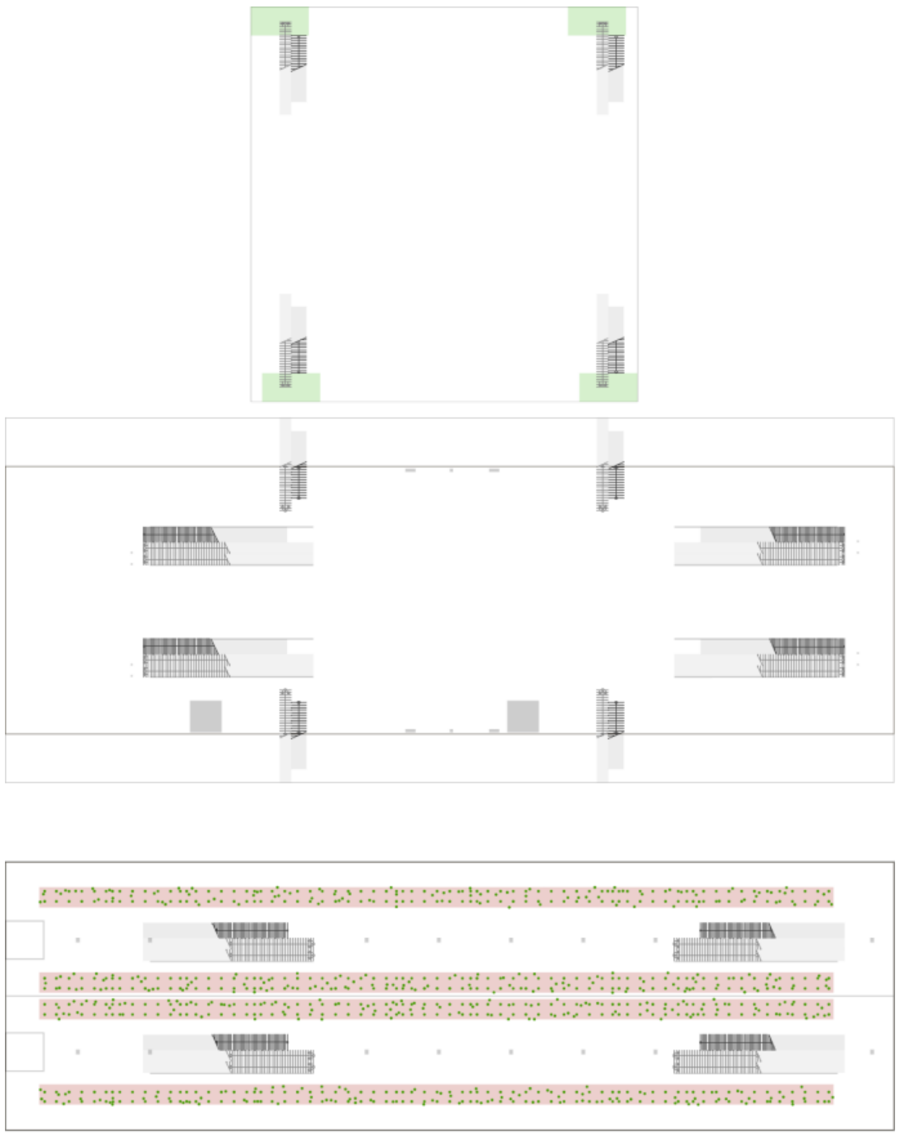

Figure 6.1: Sample Scenario in the 2D-view

audio, I/O, and networking, however, these were not used for the implementation.

Crowd:it uses the JTS Topology Suite[23] for geometry operations. As this provides only limited support for 3D geometries, libGDX was preferably used for geometry calculations.

Lastly, some basic modeling was done using blender[24].

6.3 The User Interface

The 3D visualization is displayed in an additional window that is separate from the normal crowd:it application. It offers several UI-Elements that allow for modification of the visualization. The top of the window displays several buttons that allow modifying the view of the visualization. They are explained in sections 6.7 and 6.10.2. The control panel on the right allows to enable or disable specific aspects of the visualization. The UI-Elements in the control panel are explained in sections 6.6, 6.8 and 6.9. Lastly, a play bar at the bottom allows playing through the simulation time.

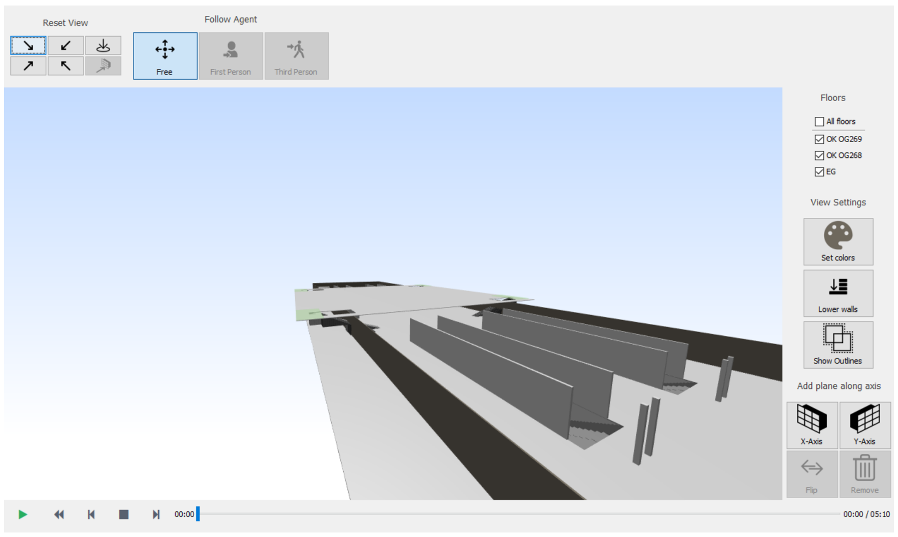

Figure 6.2: User Interface of the 3D Visualization

6.4 Geometry construction

The visualization was developed with the intent to work with any crowd:it project. Therefore, it does not directly include any three-dimensional structures that come from IFC-Files. Instead, IFC-Files are first converted to a regular crowd:it file and then visualized like any other crowd:it project. This results in the necessity to recreate three-dimensional structures from the inherently two-dimensional crowd:it projects. This section describes the algorithm that transforms 2D geometry into 3D meshes.

Geometry in crowd:it is described as a list of 2D (X, Y)-coordinates that may either be closed (i.e. it is a *polygon*), or open (*linestring*). Each geometry resides on a floor that introduces some additional input parameters: Each floor has an elevation (i.e. the height at which agents walk) and a height. In a sound scenario, the elevation and height of a floor always add up to the elevation of the next higher floor. However, crowd:it does not enforce this constraint.

The result of the algorithm is a list of 3D-Meshes. Each mesh consists of a list of 3D (X, Y, Z)-vertices (with the Y-Coordinate representing the height), a list of normals (one normal per vertex), and a list of indices into the vertex list that defines the faces of the mesh.

Therefore, a basic 3D mesh can be created as follows:

1. Transform the 2D coordinates to 3D vertices, using the elevation of the corresponding floor as Y-Coordinate.
2. Generate another set of vertices by elevating the vertices from 1 to a height defined by $elevation_{floor} + height_{floor}$
3. Generate the side faces by iterating over the vertices: v_i, v_{i+1}, $v_{i(elevated)}$ and $v_{i(elevated)+1}$ define a quad. Append indices that define two triangles using the quad of the segment to the index list.
4. Generate the top and bottom faces by triangulating the geometry defined by the initial list of vertices. For this, an EarClippingTriangulator[25] implemented in libGDX was used.

This algorithm is designed to work with polygons. Therefore, linestrings needed to be converted to polygons before applying the algorithm. This was achieved by buffering them by a constant amount using JTS, resulting in "thick" geometry. This thickness needs to be sufficiently low in order to avoid the impression of collisions between agents and walls. On the other hand, this allows to enable backface culling for rendering: Backface culling excludes faces from rendering if their normal points away from the camera[26], which improves rendering performance. This wouldn't be possible with flat surfaces, as it would make meshes invisible when looking at them from the "back".

The process described above does not yet include the calculation of normals. The render pipeline employs vertex normals rather than surface normals: The normal of a fragment is interpolated during rendering, using the normals specified for each vertex of the face that a fragment belongs to. Therefore, a workaround was needed in order to create sharp edges: Vertices were duplicated, such that each face of the mesh consists of a separate set of vertices where all normals point in the same direction. Otherwise, a cubical wall would be rendered like a sphere. More details regarding shading are described in section 6.9.

The primary purpose of the algorithm is to generate meshes for obstacles, i.e. walls. In order to use it for different purposes, slight modifications need to be made. For example, simulation objects like origins and destinations should not extend from the ground level to the ceiling. This would occlude agents, as simulation objects usually are no obstacles, and agents are allowed to walk inside them. Therefore, a fixed, very low height was used for simulation objects.

Stairs and escalators also require slight adjustments for the algorithm. Stairs and escalators are defined as a rectangular area, a tread depth, and a direction vector that indicates the direction in which agents may walk up or down. Simply applying the algorithm to the entire rectangle would result in a single cuboid, which is not a faithful representation of a stair's geometry. To resolve this, the rectangle defining a stair was split into sub-geometries: Given the direction vector and tread depth, each of these sub-geometries resembles a single step of a stair or escalator. These were then transformed individually. The height of each step was calculated in a way that is consistent with how crowd:its simulation kernel calculates the elevation of an agent on stairs. This results in a stair model that is simple, yet appropriate for the purpose of the visualization.

6.5 Agent model

Besides the geometry of a scenario, there is also a need to create a three-dimensional representation for agents. To achieve this, a mesh was modeled using Blender, which is then placed in the scene at the position of each agent. This section describes the considerations that were made when deciding on an agent model.

Each agent is described by a 3D position in time. That is, each agent has a trajectory that maps from time in seconds to (X, Y, Z) – coordinates. More details on the trajectory of an agent is described in the section regarding trajectories 6.10.

Additionally, each agent has a torso diameter that defines the body of the agent. The torso consists of a *hardshell* and a *softshell*. The hardshell is smaller than the softshell. The collision detection in the simulation kernel ensures that hardshells of agents can never overlap. Furthermore, agents are disallowed from penetrating a wall with their hardshell. The softshell also defines an area where agents usually do not collide with each other. However, softshells are allowed to overlap in certain scenarios. An agent model should recognize these properties of an agent.

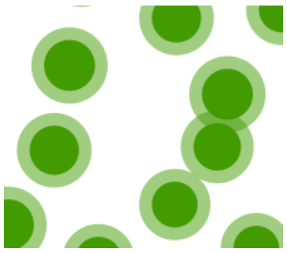

Figure 6.3: Agents and their softshells

There are two main concepts regarding the choice of an agent model: On one hand, one could choose a realistic, humanoid, textured model, like it is used in video games or movies. On the other hand, one could decide to use a very simple, rather abstract representation of a human, like a cylinder. According to the principles to avoid excessive display ink 5.3.2 and avoiding visual clutter 5.3.3, the first of the two approaches rather hurts the visualization than helping it. Furthermore, having a realistic agent model could lead to severe performance issues when trying to render a very large dataset that contains thousands of agents. Therefore, the visualization opts for a rather simple agent model.

The agent model is not quite as simple as a cylinder. It resembles a very basic humanoid silhouette that consists of a head and a body. This immediately conveys that the model represents an agent. The intent is to allow a user to quickly comprehend the scene and make the model more easily distinguishable from the remaining geometry.

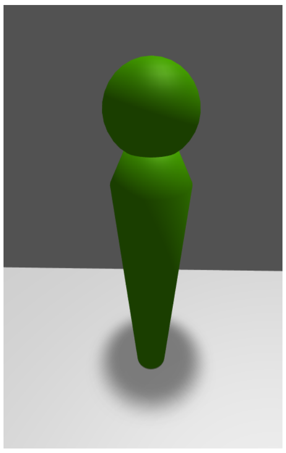

Figure 6.4: Agent model

The software Blender was used to create the humanoid model. A simple sphere represents the head. The lower part of the body is a cylindric shape, where the radius slightly extends outwards with increasing height. Shortly before joining with the head, the radius decreases again.

The model is scaled in the X and Z directions in such a way, that the radius of the bottom of the cylinder matches the hardshell radius of the corresponding agent. This is one of the two reasonable options to scale the model, of which scaling to the softshell is the other alternative. Scaling by a different factor or neglecting scaling at all, would convey false information 5.3.3. As the softshells are allowed to overlap in certain scenarios, scaling to the softshell would lead to collisions between the agent models. As this might seem like agents experienced disallowed collisions during the simulation, a choice for the former option was made. The model is not scaled in height, i.e. all agents have the same height, no matter the hardshell radius. This decision was made, as a user can hardly infer the information regarding the torso size that would be encoded with this. Instead, it might even convey false information, in that the physical height of an agent was utilized during the simulation.

The decisions concerning the scaling imply that the agent model does not yet encode any information regarding the softshell of an agent. Several options were considered in order to include the softshell in the visualization. One possible approach is strongly oriented towards the 2D representation of an agent: In 2D, the hardshell is represented by an opaque circle, while the softshell is a larger, transparent circle. A similar approach in a 3D environment would be to represent the softshell by fully covering the agent model with a second, transparent model. However, this would necessarily lead to a lot of visual clutter, especially in very crowded scenarios. Apart from that, having a lot of transparency might severely decrease performance. Therefore, this idea was abandoned.

A second approach represents the softshell by adding an additional cylinder to the agent model. This cylinder would need to have a very low height, similar to a pedestal. The cylinder may or may not be transparent. This is a viable approach, albeit introducing mild visual clutter. When evaluating this idea during the development process, it was suggested to project the cylinder down to the surface that the agent is walking on, in order to reduce the clutter even more. This would result in a decal-like appearance. As the agent shadows are already represented using decals, the decision was made to scale these shadows to the size of the softshell. This decision is certainly debatable, however, having a softshell overlapping with the shadows would most likely lead to visual clutter. A detailed explanation regarding the shadows can be found in section 6.9.2.

6.6 Color

Color is one of the visual variables described in section 5.3.1. It can be used to encode properties of the underlying data. Color also exerts a strong effect on how a scene is perceived. Therefore, choices regarding the color must be carefully examined. This section illustrates how the implementation assigns a color to the geometry of walls, simulation objects, and agents.

One major consideration concerning the specific implementation at hand involves the 2D visualization that already exists in crowd:it. The software assigns a default color to all objects, while also offering vast possibilities for customization. The user can use the coloring window to assign a color to objects like obstacles – the software distinguishes between closed and open walls, which correspond to geometric linestrings and polygons, as described in section 6.4. Additionally, a color may be assigned to agents and specific types of simulation objects, like origins or destinations. It is also possible to select individual simulation objects and assign a separate color to them.

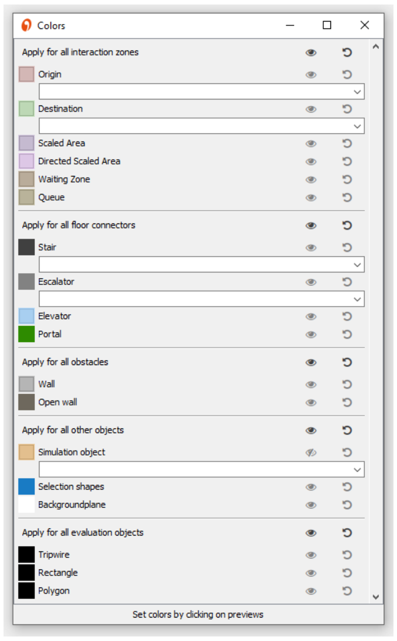

Figure 6.5: Coloring Window UI

The 3D visualization aims to use the same colors as the 2D visualization. This employs the principle to include *common symbolism* 5.3.4 in multiple views. However, it does not directly use exactly the same colors, for the reasons laid out in the following.

A user can set up any color schema for their scenario, however, the visualization should most certainly be robust regarding the default color settings. That is, a visualization with the default settings should be reasonably good, without the necessity to manually or automatically change the settings in order to gain a valuable representation. One major complication with this is that crowd:it makes extensive use of transparency in its 2D visualization. For example, the default color of closed walls is completely black but carries an alpha value of 50. This makes it look like they are drawn using a light gray color, due to the blending with the white background. The reasoning behind this is that this makes it possible to perceive structures if multiple obstacles lie on top of each other.

However, this approach does not work anymore in a 3D environment, as the transparent color is not blended against a white background anymore. Moreover, rendering all structures transparent is creates a very cluttered and incomprehensible display, while also carrying severe performance implications. Therefore, adopting the original colors

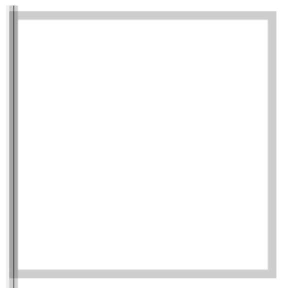

Figure 6.6: Walls in the 2D view

is not an option.

One approach to overcome this is to create a new color scheme for 3D that does not employ transparency. However, even if this also applied to the 2D view, this still undermines the intent of the common symbolisms practice, as users might be used to the default color scheme or some other color scheme that they created themselves.

Another concept might be to simply drop the alpha value of a color and keep the red, green, and blue values. This also undermines the common symbolism principle, though, as this leads to colors that are considerably darker than the original colors.

Therefore, the implementation adopts an approach that generates opaque colors from the original colors by blending them with white. This technique generates an image that is very faithful to the original colors, to the extent that the user cannot distinguish between the two colors. This allows the 3D visualization to eliminate any issues with transparency, as the generated colors are completely opaque.

However, it is important to keep in mind that this also comes with its own issues. For example, a user might explicitly choose a transparent color for an object in the 3D view, in order to be able to see the geometry behind that object. The technique described above prevents this, without completely disabling the object. One option to resolve this issue is to make this accessible to the user, i.e. the user may decide if they want to blend the colors. This is not a part of the implementation described in this thesis.

The technique described above is applied to obstacles and agents. Simulation objects, on the other hand, are rendered using their original color including the alpha value. As opposed to obstacles, simulation objects usually do not collide with agents, that is, agents are allowed to pass through simulation objects. Therefore, rendering them opaque would obstruct the view towards agents that pass through a simulation object.

Stairs and escalators are an exception to this, as agents cannot walk inside a stair, but have to scale it instead.

Apart from the considerations regarding color and their alpha values, crowd:it also allows to completely disable an object. This option is accessible as a separate button in the user interface. The concerns described above do not apply to this property. Therefore, objects are not rendered in 3D, if they are disabled in the 2D view.

The default color for agents is a shade of green, which consequently is used in the 3D view, as well. Apart from setting a custom color using the coloring window, crowd:it also allows to color agents corresponding to certain properties. For example, the software allows to color agents with a different color for each origin that agents appeared in. Another example is to color agents by their velocity, which results in a color gradient that indicates if an agent is currently fast or slow. These color settings are directly transferred to the 3D visualization.

6.7 Camera

The visualization uses a perspective camera that can be moved freely through the scene. There exist numerous well-established options on how to maneuver through a 3D environment using a mouse and a keyboard. They are known from various 3D applications, like modeling software, game engines, and video games, amongst others. This section describes how the camera controls were implemented for crowd:its 3D view.

When creating the camera controls, the well-established controls in the 2D view needed to be considered. The 2D camera is moved by dragging the middle mouse button, that is, by holding down the mouse wheel. Scrolling with the mouse wheel allows zooming into the scene. Dragging, while pressing the left mouse button, shows a selection rectangle that allows the user to select simulation objects or pedestrians. Clicking left holds a single-select functionality. Clicking right opens a context menu while dragging right currently does not hold any functionality in the 2D view.

The 3D camera controls aim to replicate this behavior as closely as possible. This allows users to quickly adapt to the three-dimensional view, without the need to learn and internalize a new set of camera controls. Also, having a severe difference in the camera controls might irritate users, when frequently changing between the two views.

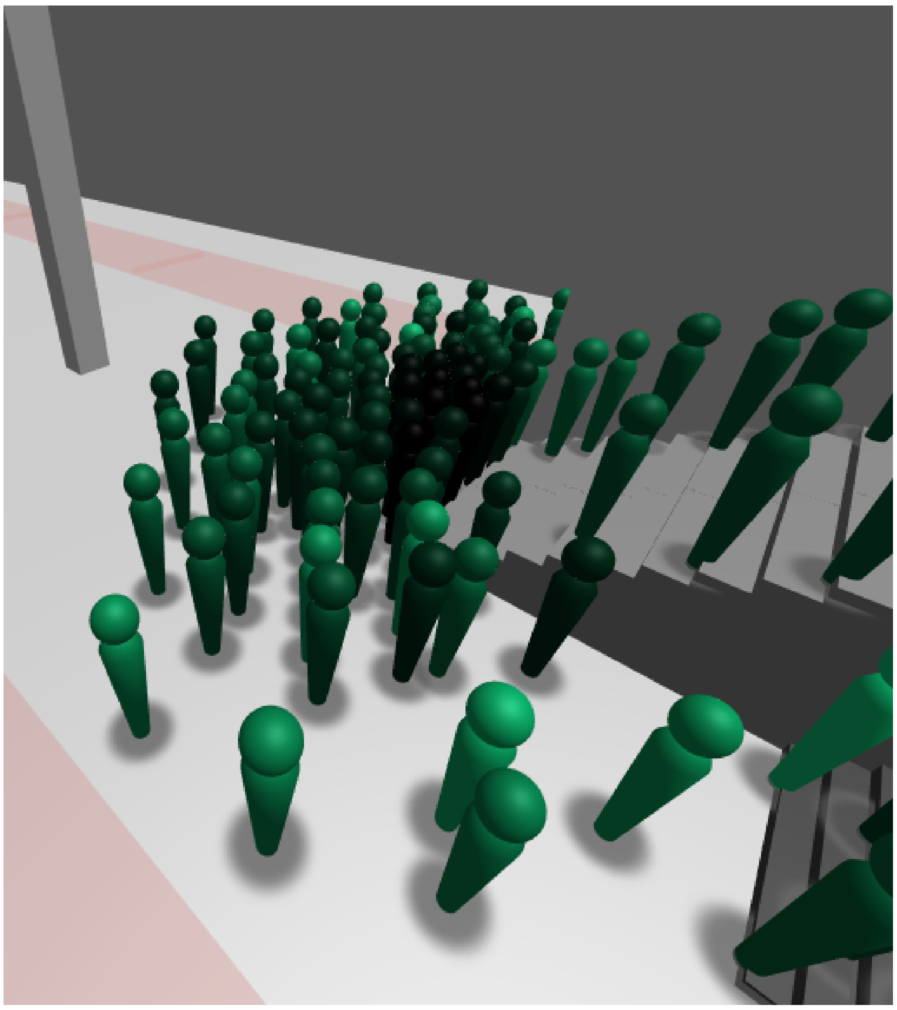

Figure 6.7: Agents, with their color determined by the current velocity

The 3D camera is moved laterally by dragging while holding the middle mouse button, similar to how it is done in the 2D view. Scrolling with the mouse wheel moves the camera forwards along the z-axis of the camera coordinate system. To the user, this seems very similar to the zooming behavior known from the 2D view. Then, there is the additional need to rotate the camera, which is done by dragging with the right mouse button. This choice was made as this kind of interaction with the right mouse button is currently not used in the 2D view. The functionality to select simulation objects is not implemented in the 3D view. However, the button mapping for the current camera controls allows adding this implementation at a later point, without interfering with the current allocation.

There are two main options to implement rotating a camera in a 3D environment. First, it is possible to rotate the camera around itself, similar to a first-person view. This approach is well known in video games but rather uncommon in conventional software applications. Then, there is the option to rotate the camera around another point of the 3D coordinate system. This point, the target, may be some arbitrary point, like the origin of the coordinate system, or it may be chosen by the user. This implementation adopts the second approach.

Concretely, the implementation updates the target point to the location that the user clicked on when starting to drag the mouse. To achieve this, the 3D location needs to be reconstructed from the 2D screen coordinates of the mouse position. The implementation realizes this by projecting the coordinates back into 3D world coordinates. This is only possible, if pixel-wise depth information is present, which is usually accomplished by rendering the depth value of each pixel to a depth buffer separate from the screen. As this depth buffer is already present in order to use it for post-processing, as explained in section 6.9.3, this approach proved to be most practical compared to other approaches.

The target point for the camera rotation is continuously updated to the last location that the user clicked on. This may either be a click that started a rotation, or a click that initiated a camera translation. The target point is not updated when scrolling with the mouse wheel, that is, moving the camera back and forth. The target is also not updated when the user clicked on a pixel that carries no depth information, because it covers the background, rather than a part of the geometry.

The camera controls described above allow moving the camera freely by only using the mouse. Beyond that, some additional buttons that allow resetting the camera view to key positions were added to the user interface. Four buttons enable the user to reset the camera to a point some units above the scenario, facing the center from four different perspectives. A fifth button allows resetting the view to a top-down perspective that resembles the image in the 2D view. These five key perspectives make it possible to gain an overview of the scenario. Moreover, some users might find it difficult to get to grips with the camera controls, especially if they are not yet used to working with 3D applications. The “reset buttons” allow them to get back on track if they got lost in the scenario.

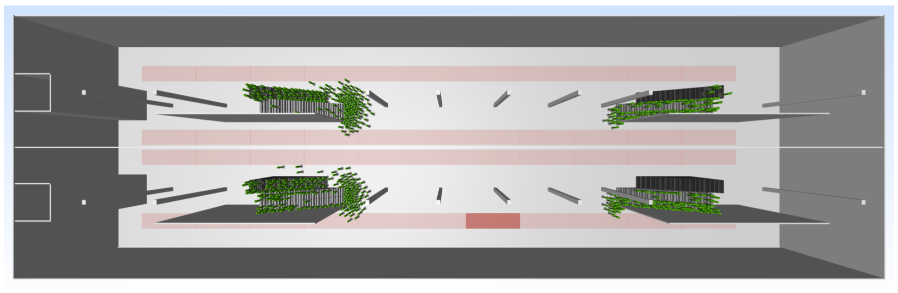

Figure 6.8: Top-down view

6.8 Gaining an Overview

Section 5.3.2 established that a visualization should enable the user to “see the whole”. Setting the camera perspective to a key position, as described in section 6.7, adheres to this principle to some extent. However, there exist several additional possibilities that allow the user to gain an overview of the entire scenario. This section describes, what the implementation at hand offers in this regard.

The key barrier to overcome when trying to comprehend the whole scenario, are walls and other obstacles obstructing the view of the user. Most scenarios represent buildings and are therefore completely surrounded by walls. Therefore, most techniques described in this section focus on hiding some of the geometry in the scene, without removing details that are vital for understanding the picture.

A scenario usually consists of multiple floors that are placed on top of each other. Crowd:its 2D view only allows one to view one floor at a time, but the 3D view displays all floors at the same time. On one hand, this is one of the key advantages over the 2D view. On the other hand, this often rather obstructs the view than enhance it. Therefore, a menu on the right side of the visualization window allows to disable individual floors, i.e. exclude them from rendering. This enables the user to decide if they want to view all floors at once, a selection of floors, or even only a single floor, as it is in the 2D view. This comes of course with the drawback that information is hidden, as pedestrians that reside on a floor that is currently disabled are not rendered.

One approach that aims to solve this is realized in the form of a button that allows to *lower the walls*. Selecting this button restricts the height of each obstacle to a fixed size. This causes walls to be much lower than the height of the pedestrians, allowing the user to look “inside” the building from the side, rather than from the top. This feature is especially convenient when tracking the trajectory of an agent over multiple floors, as the view towards the agent is never obstructed, even if the agent changes the floor.

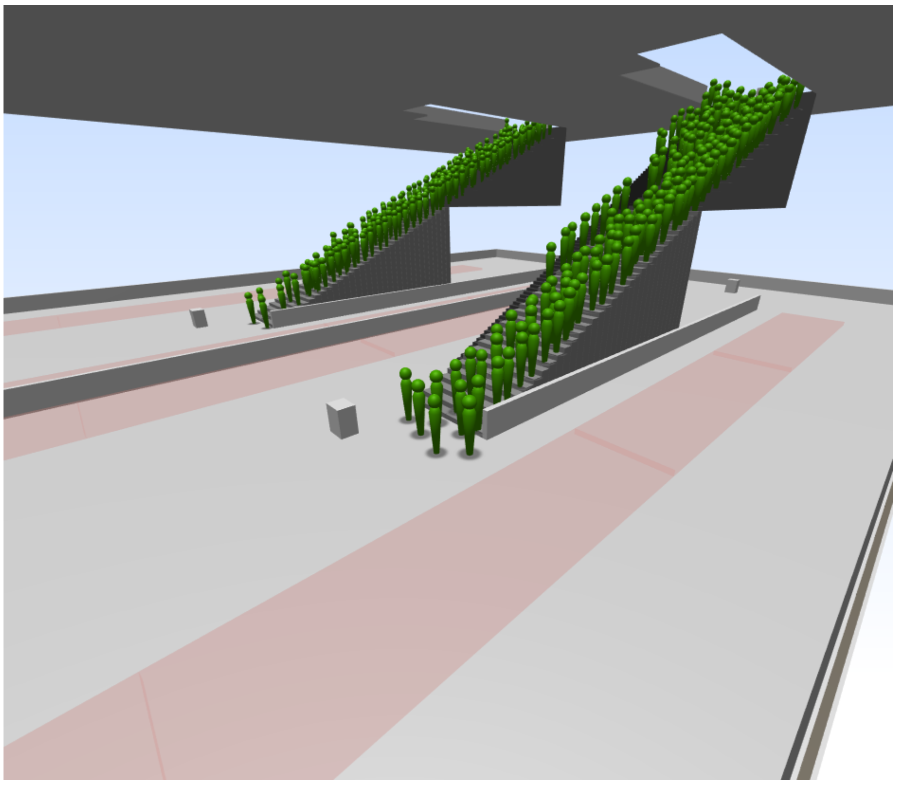

Figure 6.9: The lower-walls feature allows to look into the building from the side

One exception to this concept is stairs and escalators. The height of these is never limited since they are an essential part of an agent's trajectory. However, crowd:it allows to disable individual simulation objects, which may be used to hide specific stairs or escalators if they do happen to obstruct the view of a user.

The *lower walls* feature can be thought of as a horizontal cut through the scenario. With *planes*, there also exists a complementary feature that allows to vertically cut the scenario. A plane is defined by a normal that indicates the orientation of the plane and a float that indicates the distance of the plane from the origin. If a plane is defined, it prevents rendering the entire scene on one side of the plane, while the geometry on the other side of the plane is rendered as usual, according to the following formula displayed in 6.1[27].

The menu in the 3D view offers a section that allows adjusting settings regarding a

Listing 6.1: Signed distance

```
float signedDistance = dot(p, n.xyz) + n.w;
if(signedDistance < 0) discard;
//p: 3D position of the pixel that is currently rendered
//n.xyz: normal of the plane
//n.w: distance if the plane
```

plane. Two buttons switch the interaction mode of the 3D application to allow the user to define a plane along the X- or Z-axis, respectively. While hovering with the mouse above geometry, a transparent preview of the plane is rendered. Once the user clicks, the plane is saved and the interaction mode changes back to the normal mode. The geometry on one side of the plane is now excluded from rendering. The normal of the plane corresponds to the axis that was selected by the user. To calculate the distance from the origin, the screen position is projected into world space, as described in section 6.7. The distance is then given according to the formula `distance = -worldPosition.dot(normal)`, where `worldPosition` is the projected position of the mouse and `normal` is the normal of the plane.

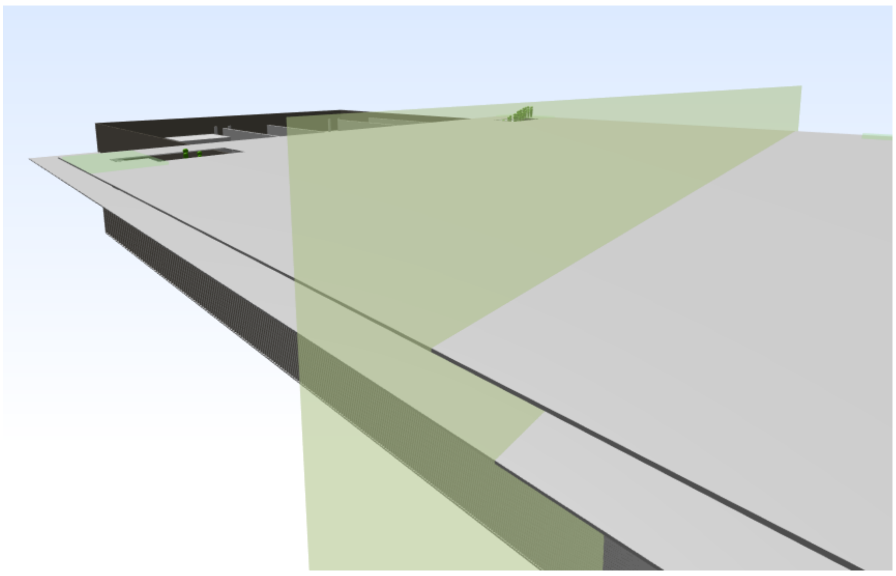

Figure 6.10: Transparent preview of the a vertical plane

The side of the plane that is hidden is intrinsic to the calculations described above but seems arbitrary to the user. A flip button allows to "flip" the plane, i.e. change the side of the plane that is hidden. This is achieved by multiplying the normal and the distance with -1, which inverts the signed distance formula 6.1.

Once a plane has been defined, an additional button is enabled in the "Reset view" menu that is described in section 6.7. This button sets the camera position to a point in front of the plane and turns the look direction of the camera towards the plane. The distance to the plane is chosen in a way that ensures that the entire scene is encompassed in the image.

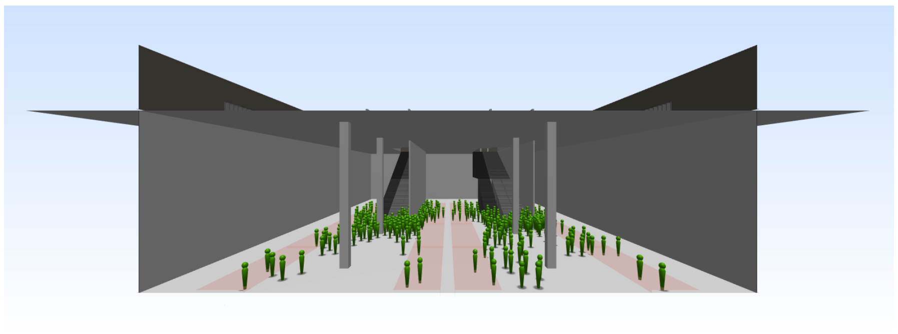
Figure 6.11: View orthogonal to the vertical clipping plane

Finally, a remove button allows removing a plane in order to render the whole scene again.

6.9 Shading

This section describes the technical details regarding the shading of the scene. This includes the basic shading model, the shadows of the agents, and a post-processing effect that renders outlines around the geometry of the scene.

6.9.1 Lighting model

The visualization utilizes Blinn-Phong shading as lighting model[28]. Blinn-Phong shading is a local, pixel-based lighting model that aims to replicate locally realistic lighting. It considers the direction and color of the light, the normal of the polygon that is rendered, as well as some additional parameters, like material constants, the color of the surface, and ambient color. Position, color, and normal information are given per-vertex and interpolated across each polygon in order to enable a pixel-based calculation.

One reasonable alternative is Gouraud shading, which is calculated per-vertex and interpolated across each polygon in order to color the pixels. While being superior performance-wise, this approach results in poor visual results for large surfaces that contain only a few vertices.

Rendering the geometry without a lighting model results in the best performance, as each pixel is colored according to the fixed color of the geometry without the need for additional calculations. However, this results in a poor visual quality that makes it very hard to visually comprehend structures. A modification of this is to tint surfaces according to their angle to the camera. If performance issues arise with the implemented shading model, or if the visualization is to be used on low-end hardware, this is a valid approach to resolve the issues.

6.9.2 Shadows

One common issue with 3D visualizations is perspective distortion which makes it necessary to implement depth cues. Section 5.3.3 introduced shadows as a depth cue. Shadows are a useful tool to allow the user to infer depth information, but may also lead to visual clutter or hide details if used excessively.

Blinn-Phong shading, as described in section 6.9.1 is a local lighting model, which means that it does not take into account that objects may occlude the light source. Therefore, it is necessary to add additional steps to the rendering pipeline that are concerned with shadows. There exist various concepts to create realistic shadows when rendering a three-dimensional environment. Shadow mapping[29] is one of the most common among them. It is implemented, by applying a preprocessing step that creates a shadow map: First, the scene is rendered from the point of view of the light source. The depth values of each pixel are stored in a texture. Then, during normal rendering, the world coordinates of each pixel are restored, as described in section 6.7. The restored 3D coordinates allow comparing the distance of the pixel being rendered to the corresponding depth value in the shadow map. If the distance is larger than the stored depth, the pixel is in shade.

However, this approach comes with some issues. The most critical among these is that the quality of the shadows depends on the size of the shadow texture. A low-resolution results in poor quality, which comes in the form of a considerable amount of visual clutter. There exist many techniques that improve the quality of the shadows and enable soft shadows, but these are too demanding for a real-time scenario, considering the simplistic look that the visualization aims for. Additionally, larger scenarios require larger shadow maps to provide a consistent shadow quality, which increases computational load yet again.

One important point that needs to be considered is that issues regarding depth cues mostly affect the agents. Obstacles are affected to a lesser degree, as they are static geometry. In fact, rendering shadows of the geometry may even be harmful: There exists only one light source, so the shadow of a ceiling occludes the entire indoor scenery.

Therefore, the visualization implements a more simplistic approach that does not necessarily aim for realistic shadows. The shadows of the agents are replicated by rendering them as transparent circles that are projected on the ground. This is achieved by placing 2-dimensional textures on the ground as decals. This approach comes with the advantages of a visual depth cue while avoiding common issues with realistic shadows. Its visuals do not look as realistic as shadow mapping, or other approaches to achieving realistic shadows; however, realism is not an objective of the visualization.

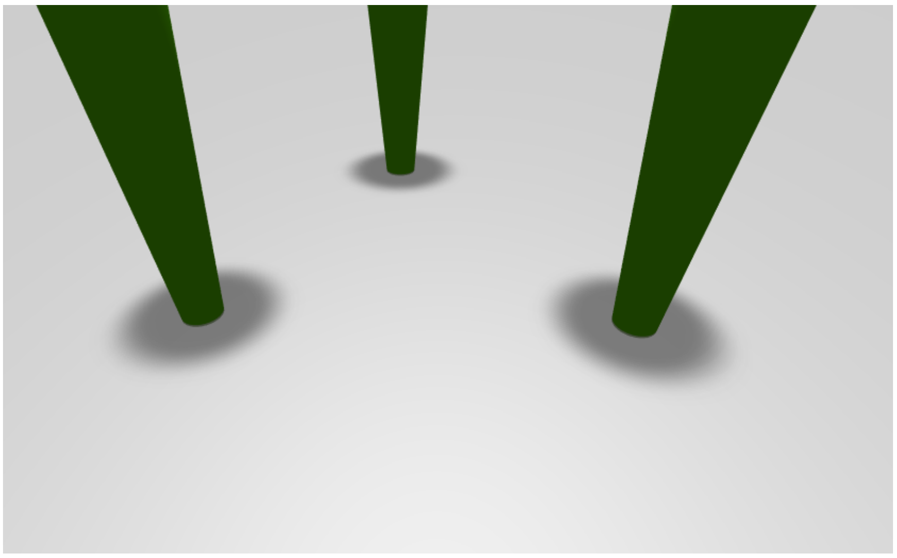
Figure 6.12: Agent shadows

6.9.3 Outlines

There persists one issue with the shading implementation described above: There are certain scenarios, where it is difficult to discern two different surfaces, due to the lighting conditions, as displayed in figure 6.13. One approach to overcome this issue is to additionally render the model as wireframe, which renders the connections between the vertices of a mesh. However, this renders all edges in a mesh, instead of only the edges where two models overlap. This makes the triangles that make up a model visible, resulting in a very cluttered look.

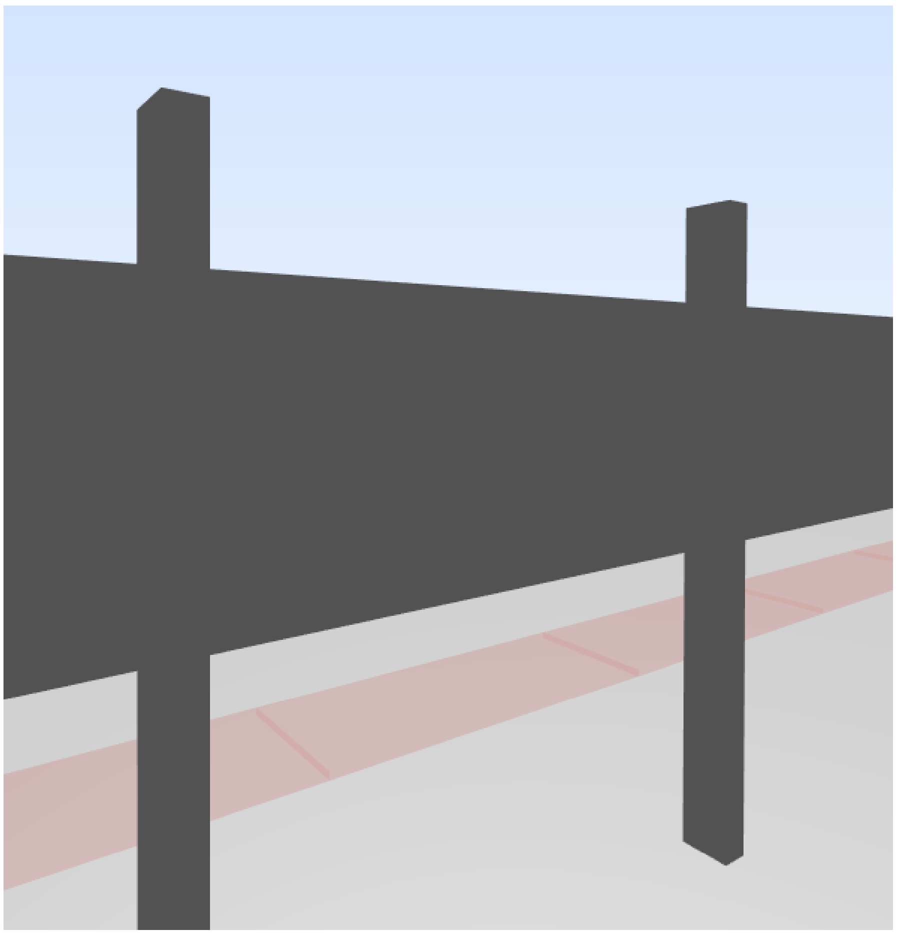

Figure 6.13: It is hard to discern the pillars from the wall in the back

Therefore, a different approach was adopted: A post-processing effect identifies edges between two objects and renders a black line between them. This is a post-processing effect, that is, it is applied after the scene has been rendered as described in the sections above. The basic approach is as follows: First, the scene is rendered to a render texture, which is an offscreen render target. Then a second rendering pass is applied that renders to the screen. Input to that render pass is the render texture from the previous render pass and the depth texture of the scene. To determine, where an outline should be placed, a Laplace filter[30] is applied to the depth texture. Laplace filtering is a common yet simple approach to edge detection. If the value calculated per pixel exceeds a certain threshold, the pixel is colored black.

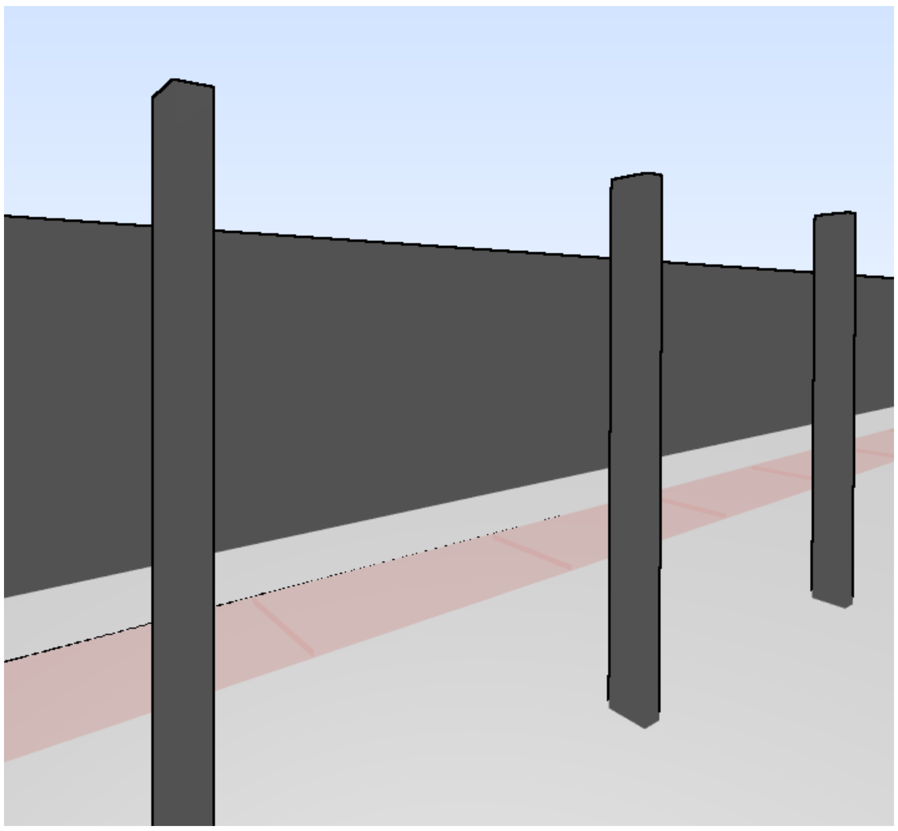
Figure 6.14: Outlines are rendered at the edges of the pillars

The outlines feature can be enabled and disabled with a button in the control panel to the right of the main window of the visualization.

6.10 Trajectories

The most important output of a crowd simulation is the trajectories of the agents. Therefore, a visualization should attach great importance to the visualization of the agent's trajectories. This section describes, how this is implemented in the visualization at hand.

6.10.1 Traces

A trajectory is a function of time that encodes a sequence of an agent's position in three-dimensional space. A common approach to visualize a trajectory is in the form that is similar to a video: The display shows the positions of all agents at one particular moment in time. Control buttons allow to play the visualization, that is, cycling through the timesteps that are available for the trajectories. This kind of visualization is the default view in the 2D section of crowd:it, as well as in the 3D visualization.

This has one major drawback, in that it is only possible to see one moment in time at once. This is only a partial view of an agent's trajectory, making it difficult to analyze the path that an agent takes through the scene. It is not possible to view the entire trajectory at once.

The 2D view of crowd:it allows to visualize an agent's trace by painting a linestring that connects the positions that are contained in a trajectory. This is a suitable approach for two-dimensional scenarios. However, since it is not possible to show more than one floor at a time, it is still only possible to see the segment of a trajectory that resides on the floor that is currently visible. One purpose of the 3D visualization is to resolve this issue.

The 3D view implements a similar approach to the traces that are available in the 2D section. It shows a sequence of arrows that periodically appear at an agent's position and point in the walk direction of the agent. The arrows are the visual mark that encodes the position in form of a glyph[31]. The feature is enabled by selecting an agent in the 2D view using the identify agents functionality. While the 2D view draws traces for all agents, the 3D view draws the arrows only for a single highlighted agent, due to the severe visual clutter that would arise when rendering multiple trajectories at once.

The arrows serve several purposes: They encode the position of an agent, as well as

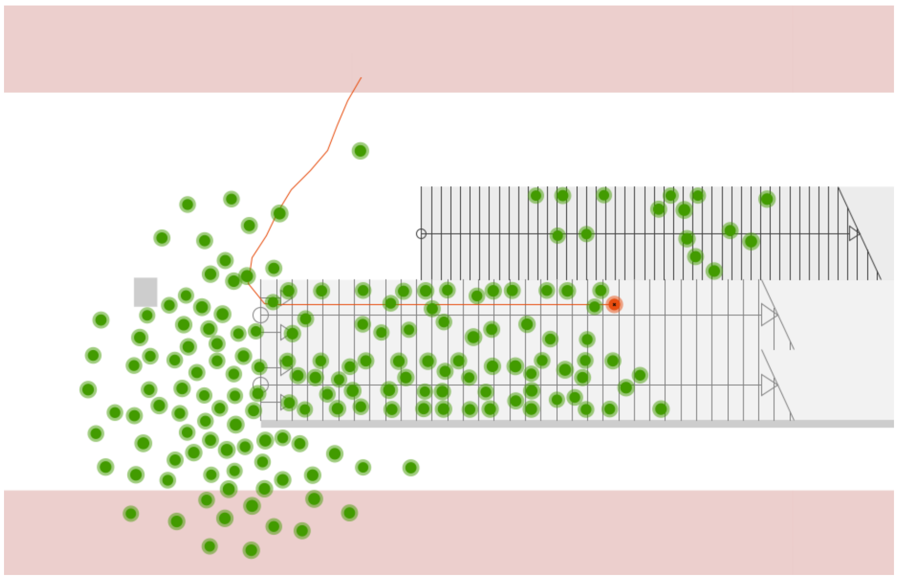

Figure 6.15: 2D trace of an agent

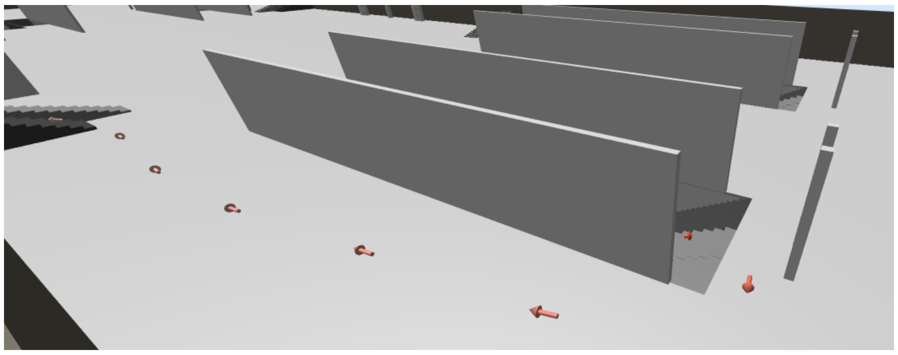

Figure 6.16: Arrows show the path that an agent traveled

their velocity and their direction of movement at each point in time. The position is encoded via the position of the arrow. The velocity of an agent is encoded by two visual variables, employing the principle to employ redundancy (see 5.3.2): Position and size. First, the spacing between the arrows is determined by the speed of the agent, as the arrows are placed in fixed time intervals. That is, an arrow is placed "every five seconds" of an agent's movement. Short distances between the arrows mean that the agent traveled slowly and vice versa. Second, the length of the arrows is scaled according to the speed of the agent. Long arrows mean that an agent traveled fast, and short arrows mean that an agent traveled slow. Each of these two visual variables by itself would not suffice to encode velocity, as it is not apparent that they are tied to the speed of the agent.

There remains one issue that occurs if the trajectory of an agent intersects itself, or if an agent waits for an extended amount of time at the same place: The arrows no longer form a smooth line from the origin to the destination, but accumulate in a cluster of arrows that is difficult to comprehend. To resolve this, the arrows are tinted according to the point in time that they belong to. At the start of an agent's path, arrows are colored rather dark. The further the agent progresses along its path, the brighter the color gets. This helps to identify if two arrows represent points in time that directly succeed each other. The base color of the arrow matches the color of the agent.

The trajectory only contains positions. Therefore, the walk direction of an agent needs to be inferred from these. The simplest approach to achieve this is to take the vector from one position in time to a position at another point in time. The two points in time may immediately follow in succession, but this is not a necessity. For the implementation at hand, the decision was made to include a larger gap: The implementation uses the points in time where the previous and next arrows are placed. The start and end positions of an agent are used at the boundary of a trajectory. If an agent does not move, the arrow points upwards.

6.10.2 Follow agent

An additional tool that allows analyzing the trajectory of an agent is the follow agent modes. These modes change the controls of the application and cause the camera to follow an agent while it traverses along its path. Selecting an agent in the 2D view enables three buttons at the top of the 3D visualization window that allows switching between three modes: Free, First Person, and Third Person.

The free mode resembles the standard mode that is always active when no agent is selected. The controls of this mode are described in section 6.7. If the user activates the first-person mode, the camera moves into the perspective of the selected agent. The position of the camera is inside the agent's head. It is rotated in such a way that it looks into the movement direction of the agent. The agent model itself does not obstruct the view of the user, due to the cameras near clipping plane and backface culling, as described in section 6.4. The third-person perspective works similar to the first-person perspective, with the difference that the camera resides in a position slightly behind the agent and looks towards the agent. When the first-person or third-person mode is enabled, the user cannot move or rotate the camera without switching to the free mode. Advancing the simulation time using a play bar that is available at the bottom of the visualization window moves the camera according to the movement of the agent.

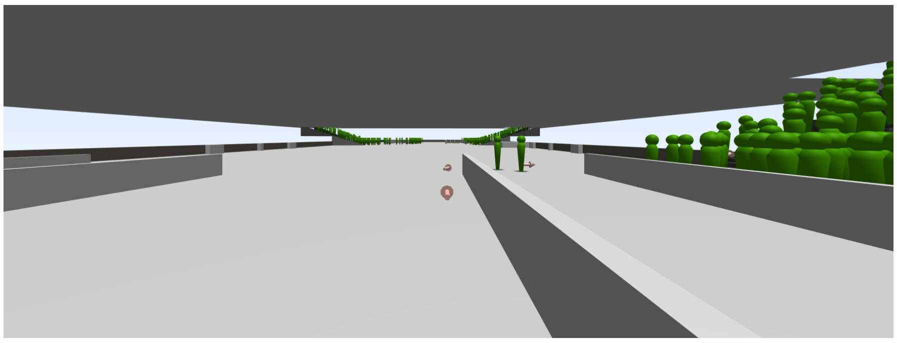
Figure 6.17: First-person view of an agent

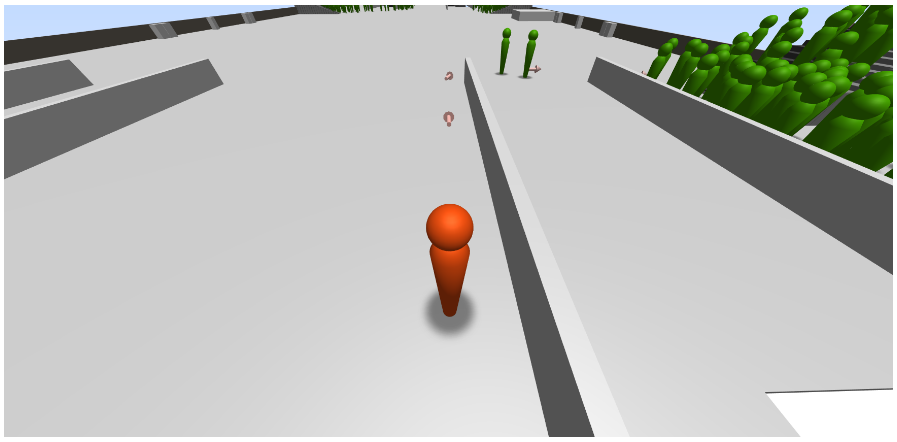
Figure 6.18: Third-person view of an agent

The follow-agent modes allow a user to immerse into the view of an agent and analyze the evacuation as if the user would be part of the simulation. This aims to improve the understanding of specific situations, for example, congestion in front of a stair or escalator. Therefore, the technical details of the follow-agent modes must be implemented in a way that coincides with the goal of these modes, that is, they should improve the user's immersion. One example of this is the calculation of the direction of movement of an agent.

Both the first-person and third-person modes need the direction of movement of an agent in order to calculate the rotation of the camera. Section 6.10 describes how a

Listing 6.2: Excerpt of the algorithm that finds the look direction of an agent

```
for (int i = currentTimestep + 1; i <= disappearsAt; i++) {
    Vector3 someFuturePosition = getPositionAt(i);
    if (pointToLookAtFarEnoughAway(currentPosition, someFuturePosition)) {
        return makeDirection(currentPosition, someFuturePosition);
    }
}
```

direction of an arrow is calculated that points in the direction of movement of an agent. However, this approach showed poor results when applied to the follow-agent modes, since it produced very abrupt changes in the camera's rotation. Therefore, a different implementation was created that aims to calculate a realistic "look-direction" of an agent.

The look-direction of an agent represents the direction that a real person would look into. This does not necessarily coincide with the direction of movement. For example, an agent might do a small sidestep in order to avoid collision with other agents. However, a real person would not actually look to the side if they would only do a small step and then continue their way in the direction that they previously walked into. A person much rather looks in the direction that they want to walk into.

The implementation solves this by using the direction to the position that an agent will be at "in two meters": The algorithm finds the first position among the future positions of the agent that is farther away than two meters. Listing 6.2 shows an excerpt of the algorithm that finds this position.

The algorithm applies smoothing to the trajectory of the agent. It has the advantage that it applies a larger smoothing if an agent makes many sidesteps while staying in the same region. This results in the effect that the calculated direction reproduces the direction in which the agent wants to go rather well.

6.11 Heatmaps

Crowd:it allows the calculation of several tile-based heatmaps. It aggregates data like the position or velocity of agents into quadratic cells with a size defined by the user. Some heatmaps allow the calculation of a time-based moving average. Each cell holds a scalar value. A color scheme that may be customized by the user determines the color of each cell, based on the scalar value that belongs to each cell. The 3D visualization renders these cells as three-dimensional transparent cuboids. They are usually placed on ground level. On stairs and escalators, their position is elevated in such a way that they reside on top of the steps of the stair.

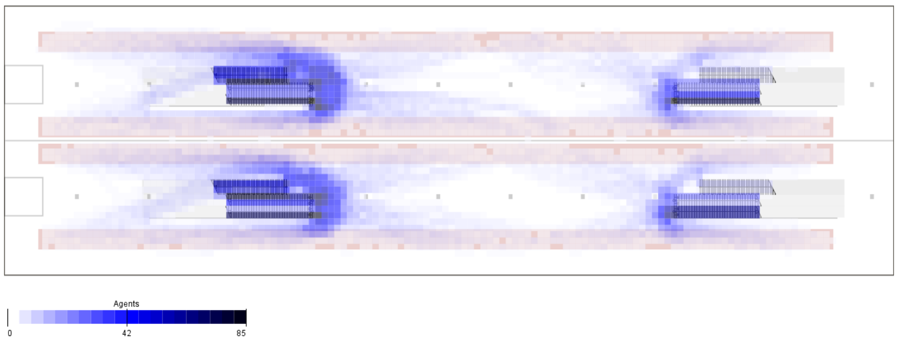

Figure 6.19: 2D heatmap

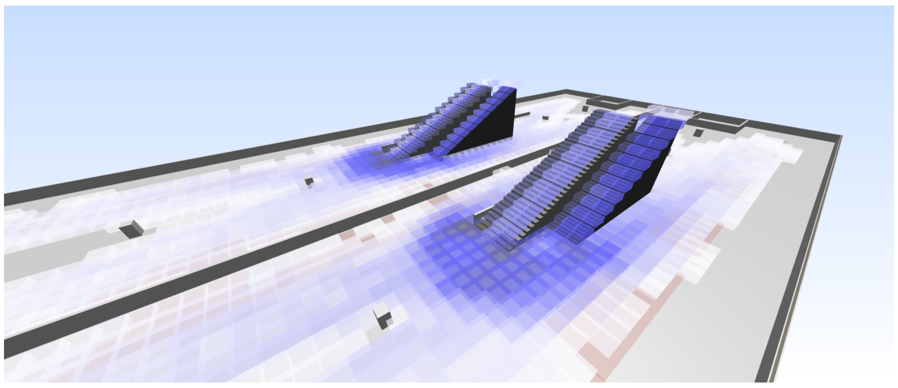
Figure 6.20: 3D heatmap

In the course of this thesis, one additional heatmap type was implemented. It represents the average walk direction of all agents per tile. This addresses one issue with the visualization of agent traces that was described in section 6.10: It is only possible to show arrows for one trajectory at a time – the trace visualization does not allow to display arrows for multiple agents at once, in order to avoid visual clutter. The heatmap solution that is proposed in this section avoids this clutter by aggregating the walk direction of each agent into cells and thereby limiting the number of arrows that are shown at once.

Listing 6.3: Calculate the average direction for an agent

```
public static final int WINDOW_SIZE = 4;
Map<Pedestrian, DirectionCalculator> directionCalculators;
Map<RegularGridIndex, Vector3> aggregateDirection;

void calculateValueForTile(Pedestrian ped, RegularGridIndex cell, int
    timeStep) {
    var directionCalculator = directionCalculators.computeIfAbsent(ped,
    p -> new WalkDirectionCalculator(p.getTrajectory(), WINDOW_SIZE));
    var direction = directionCalculator.calculateDirectionAt(timeStep);
    aggregateDirection.computeIfAbsent(cell, c -> new Vector3()).add(
        direction);
}
```

The calculation for the average walk direction is embedded into crowd:its architecture to calculate heatmaps: It is a "Pedestrian Heatmap Calculator" that allows calculating a heatmap for data based on agents. This calculator offers various entry points to calculate the heatmap. First, it allows performing cell-based calculations for each pedestrian at a given timestep. Every heatmap aggregates data over the entire simulation time, so this calculation is done once per agent for every time step in the simulation. The given cell depends on the current position of the agent at the given timestep. Listing 6.3 displays the calculations that are done to calculate the average direction for each agent.

The implementation uses the same direction calculator as was used in section 6.10 to calculate the direction of the arrows that represent an agent's trajectory. It saves the calculated direction in a hashmap that assigns each direction to one cell of the heatmap. The directions are added up, which means that steps in the same direction accumulate, while steps in opposite directions cancel each other out.

Once these calculations have been carried out, the architecture requires the calculator to finalize the results for each tile, in order to save a value for the heatmap. Listing 6.4 displays the code that is executed at this stage of the calculations.

The heatmap uses a float array to save the calculated value for each tile. In the case of the average direction, this float represents the angle of the direction that was calculated. To calculate this angle, the resulting vector from the previous step is retrieved from the hashmap. The vector needs to exceed a significant length – otherwise, no average direction can be calculated, since the walk direction of all agents canceled each other

Listing 6.4: Calculate the value for each heatmap tile

```
void finalizeResultForEachTile(int xPosInGrid, int yPosInGrid) {
    Vector3 direction = aggregateDirection.get(new RegularGridIndex(
        xPosInGrid, yPosInGrid));
    if (direction != null) {
        if (direction.len() > SIGNIFICANT_DIRECTION_LENGTH) {
            Vector2 projected = new Vector2(direction.x, direction.z);
            saveAngleAt(xPosInGrid, yPosInGrid, projected.angleRad() + Math
                .PI);
        } else
            saveAngleAt(xPosInGrid, yPosInGrid, NO_VALUE_PRESENT);
    }
}


private void saveAngleAt(int xPosInGrid, int yPosInGrid, float angle) {
    heatMapField[xPosInGrid][yPosInGrid] = angle;
}
```

out. Then, the angle of the vector is saved to the heatmap. Since the heatmap is only able to hold positive values, the calculated angle is shifted accordingly.

The resulting heatmap can be visualized in 2D, as well as in the 3D section. However, a color-coding for the direction, as shown in figure 6.21 is a poor representation for an angle, as it is hard to interpret for a human user. Therefore, a custom visualization was implemented in the 3D view: It recovers the angle from the heatmap and draws arrows instead of the cuboids. This allows a user to immediately infer the direction that most agents walked into.

Issues with the current implementation of heatmaps are discussed in section 6.12.

6.12 Technical Limitations

This section briefly discusses some shortcomings that became apparent in the continuous evaluation during the implementation.

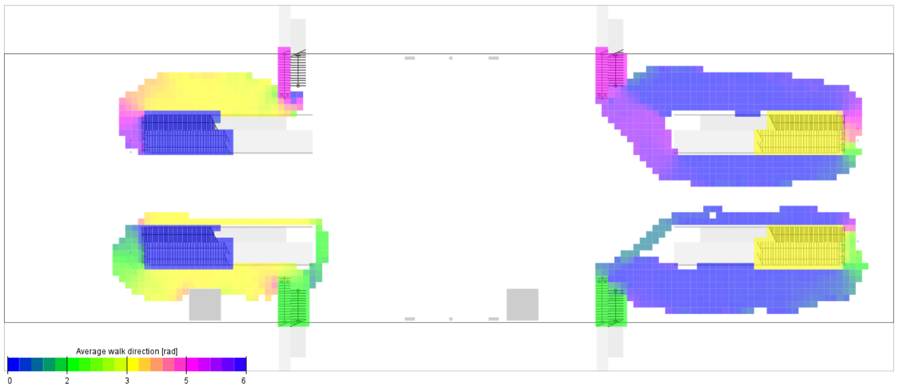

Figure 6.21: 2D heatmap that displays the average walk direction of an agent

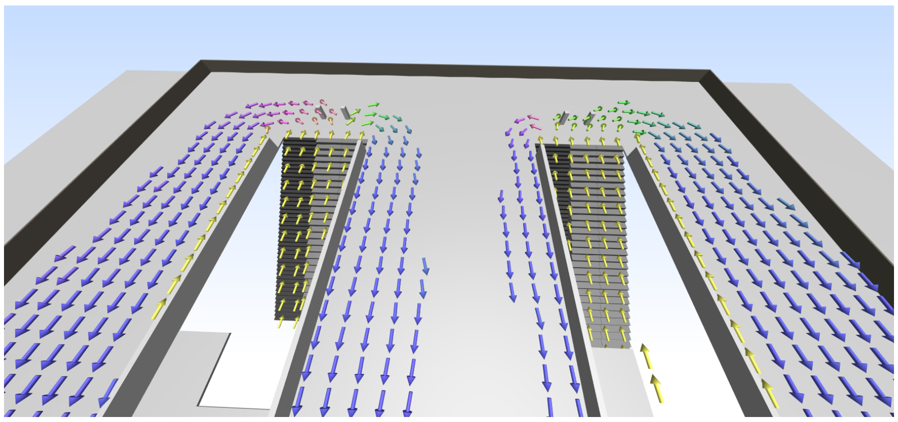

Figure 6.22: 3D heatmap that displays the average walk direction of an agent as arrows

Camera Target

The target point of the camera is currently calculated by using the depth buffer of the camera and inferring the three-dimensional location of the point that the user clicked on. This approach is easy to implement, however, it comes with its challenges. The main issue is that it is rather imprecise, especially if the clicked object is far away from the camera. With increasing distance, the area that is covered by a pixel expands as well, making it impossible to reconstruct a single point that the user clicked on.

Another issue with this system is that transparent objects also write to the depth buffer, like the plane that is rendered while a user defines a clipping plane. This interferes with the camera handling, as it makes it impossible to click on a point behind the plane, making defining a plane slightly inconvenient.

This is usually resolved by casting a ray from the camera position into the scene and performing collision detection with the scene geometry. Objects like the plane may be ignored by this algorithm. The same approach can also be used for user interactions that require the user to click on an object or select objects in the 3D view.

Heatmaps

The potential of heatmaps in three-dimensional scenes is not fully exploited since the software architecture for heatmaps exclusively targets 2D heatmaps that encode a single float value per cell. Artifacts of this technical detail are visible in the 3D view. For example, there are no heatmap cells below stairs. Due to the two-dimensional nature of the heatmaps, there is only one single cell that covers the corresponding 2D area. In 3D, the cuboids are elevated to the height of the steps of the stair, but there can never be a cuboid on top of the stair and another one directly below the first cuboid.

The fact that the heatmap only stores single floats also has some implications for the average direction heatmap. It is not only possible to calculate an average direction, but also the certainty, i.e. strength, of each direction. This could be visualized by adjusting the length of the arrows for the average direction heatmap. However, it is not possible to encode both the angle and the length of a vector with a single float.

Resolving these issues requires either some unusual workaround or a rework of the heatmaps' software architecture.

Smooth transition between timesteps

The simulation provides new agent positions every 0.5 seconds. If played in real-time, this results in a framerate of two frames per second. This is by far too low to experience a smooth video. Usually, the speed of the visualization is accelerated by a factor of ten, which resolves the issue. However, certain situations require slowing down the visualization speed, for example when using the Follow Agent mode. One could argue that proper immersion is only achieved when playing the simulation in real-time. This results in a very choppy view. One idea to resolve this is to interpolate the positions between timesteps, however, this is beyond the scope of this thesis.

7 Evaluation: User Study

7.1 Study Design

Once the implementation of the proof of concept was finished, a user study was conducted, in order to evaluate the implementation. The primary goal of the user study is to evaluate if the implementation meets the target that was defined in section 1. Specifically, it assesses, if the visualization is able to successfully support an analyst when interpreting simulation results. Additionally, the user study evaluates the general use of the visualization, i.e. the benefit it provides for the BEYOND research project and crowd:it as commercial software in general.

The user study was conducted in the form of one-on-one interviews between the author as the interviewer and the participants. First, an overview of the sample scenario and the corresponding 2D and 3D visualization was demonstrated. Then, the implementation was presented feature by feature. In order to gather feedback, context-dependent questions were asked. Afterward, a free discussion was held that depended on the individual feedback that was provided.

A total of five participants took part in the user study. Two participants are external customers of accu:rate. One participant is an accu:rate employee that conducts evacuation, process, and comfort analyses for the company. These three participants work with crowd:it regularly and are therefore well-versed in interacting with crowd:its two-dimensional view. One participant, who is an accu:rate employee working in the software department, only recently joined the accu:rate team and is therefore not yet used to crowd:its characteristics. Finally, one participant is a project partner of BEYOND who is affiliated with DB.

The participants had a very diverse view on the visualization. Some directly work with the software and perform crowd simulations, others are project partners and have an external view on the matter. Therefore, the questions in the one-on-one interviews took the background of the participant into account. Three of the study sessions were conducted remotely, while the remaining two sessions were conducted face-to-face. In the face-to-face sessions, additional usability testing was performed, where the participants operated the 3D visualization themselves.

All participants evaluated the 3D visualization as generally beneficial for the software crowd:it, as well as crowd simulations in general. Some features were perceived as very valuable, while others have room for improvement. The following sections first present the features that are considered a unique strength of the implementation. Then, suggestions for improvement are summarized.

7.2 Strengths

Follow Agent

Most participants perceived the Follow Agent modes as very valuable. The goal of the Follow Agent Modes was to improve the immersion of a user, help them to gain a clearer understanding of the scene, and aid them in investigating specific situations, like congestions. These modes were appreciated by most participants. One of them, in particular, pointed out that this is also a useful tool when presenting the results to a person outside of the field of crowd simulations, like clients.

Heatmaps and Average Walk Direction

The heatmap visualization was seen positively by most participants, although some of them prefer a flat, two-dimensional projection on the ground over the three-dimensional boxes. One participant noted that the 3D view makes it easier to comprehend, how exactly a specific heatmap emerges – especially for clients.

The average walk direction was seen as an outstanding feature. It summarizes all simulation results into a single timeframe. Therefore, it is possible to comprehend the basic results of a simulation without the need to press the play button and look at the simulation in a video-like manner. The average walk direction immediately conveys an understanding of a scene, even if a user has never seen the scenario before. The only remark that was raised regards the color of the arrows that are depicted with the average walk direction. They are perceived as rather distracting, as a user does not necessarily understand, what kind of information is conveyed by them. Therefore, it was suggested to paint all arrows with the same color.

There are also some technical limitations to heatmaps and the average walk direction. However, these were not discussed during the user study. Therefore, these are summarized in section 5.3.3.

General Impression

In general, the 3D visualization was assessed rather well. It was noted that the visualization makes it easy to comprehend the scenario as a whole. Opposed to the two-dimensional, "floor-by-floor" visualization, the 3D view enables one to look at the entire building at once and understand the building's structure. The Lower Walls feature was highlighted particularly often.

However, it must be noted that none of the participants really considered the visualization as a tool for analysts. It was much rather perceived as a tool to present simulation results to clients. One participant pointed out that an expert automatically creates an understanding of the structure of the building while working on the project. Therefore, they do not see an advantage in the 3D view when an analyst evaluates the simulation results. Most participants agreed that the visualization is more valuable when presenting the results to clients or for supplementing a final report.

Despite this circumstance, participants still appreciated the abstract agent model. One participant explicitly preferred it over realistic, animated agent models, since it does not portray a "pseudo-realty". Nevertheless, they also pointed out that clients sometimes explicitly demand realistic agent models.

7.3 Suggested improvements

Camera Controls

The camera controls aim to resemble the controls of the 2D-view as closely as possible, in order to provide controls that the current crowd:it users are familiar with. Viewed in isolation, this goal was accomplished. During usability testing, users that were already used to working with crowd:it were able to intuitively handle the camera controls. However, inexperienced users faced more troubles with the camera controls. The button mapping was debated, however, it seems implausible to change it to a mapping that is different from that in the 2D view. Apart from that, rotating the camera proved to be particularly difficult, even for experienced users. This is due to the fact that it turned out to be challenging to comprehend the concept behind the target point of the camera. The Reset View buttons helped with the issue but cannot replace camera controls that feel natural. One suggestion to resolve this issue involves rendering the camera target using a custom mesh. This would immediately convey the importance of that particular location in 3D space. Another suggestion was to switch the controls to a traditional first-person view that is primarily known from video games. This system would cause the camera to rotate around itself, as opposed to a target point separate from the camera.

One participant criticized the perspective nature of the camera. They suggested allowing the user to employ some sort of parallel projection[32].

Rendering

There were several remarks regarding how specific elements of the scenario are rendered. One participant criticized the model that is automatically generated for stairs and escalators. They suggested changing the model in a way that prevents "abrupt" disturbances in the geometry. Figure 7.1 visualizes the issue. They proposed to make the bottom of the stair either diagonal or to adjust it to the height of the agents.

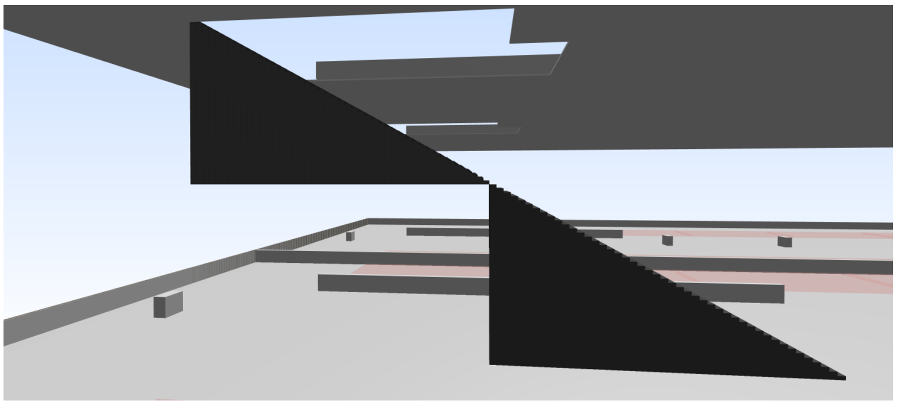

Figure 7.1: Stair model

Most participants rejected the arrows as visualization of the traces. They pointed out several issues with the traces: First, the arrow visualization is equivalent to a smoothing of the trajectory. Therefore, it may suppress important details, like detours in congestions or sidesteps. Second, they may hide essential parts of a trajectory when an agent travels with an elevator. If an elevator ride is comparatively short, the visualization may not render a single arrow that represents a point in time where the agent was inside the elevator. Therefore, the visualization would miss arrows that point upwards. Lastly, the arrow visualization violates the principle of common symbolism5.3.4. Due to these reasons, participants preferred either a simplistic line on the floor or a three-dimensional tube as visualization for the trajectories.

It was noted that the outlines are rendered too thick. Participants asked to draw the lines more finely, providing a more elegant picture. However, this is not a trivial task, since the thickness of the lines is directly tied to the algorithm that generates the outlines. Rendering fine lines requires implementing a different algorithm.

The possibility to show and hide simulation objects or entire floors was appreciated. However, participants still missed some features regarding the visibility of certain structures. For example, it is possible to show and hide individual simulation objects, however, this is not possible with obstacles. Users are only able to disable floors as a whole, but not individual walls. Additionally, participants demanded functionality to render obstacles transparent, similar to the transparent simulation objects. One participant suggested rendering obstacles as a transparent, gray, "ghost-like" structure, by omitting color information.

Quality of Life Improvements

Participants also pointed out a few quality of life improvements. One of them suggested making it possible to move a vertical clipping plane, once it has been set via the plane feature that is described in 6.8. Currently, it is only possible to first remove the plane, and then set up a new one.

Another suggested quality of life improvement is that it should be possible to modify the arrangement of the UI. This comprises both the windows of the 2D and 3D view, as well as individual UI elements. However, participants noted that it should suffice to directly integrate the 3D window into the application, for example, as a separate tab beside the floors. Currently, the 3D view is implemented as a separate window, for simplicities sake.

Opportunities with the IFC Standard

One significant improvement is to employ more of the advantages of the IFC standard. An obvious advancement is to make more use of the geometric data that is available in an IFC file. Converting the file to a crowd:it project results in a loss of information. Ideally, crowd:it would directly access the IFC file, which would enable the 3D visualization to display the exact meshes from the file, instead of recreating them as described in section refgeometry.

One participant noted that this would particularly increase the benefit for DB. The company currently shows great interest to increase passenger capacity in existing buildings and identifying bottlenecks in existing and new buildings[33]. Therefore, this improvement would allow to integrate crowd simulations via the BIM workflow into the "Strong Rail" strategy. Integrating additional component libraries that come from DB would make it possible to display agents that walk through an exact replication of the original model.

7.4 Result

The visualization received generally positive feedback from the participants. It is important to note that the primary goal to create a tool that mainly supports analysts was missed. While it may support analysis in rare cases, the visualization was predominantly seen as a tool that is used to display simulation results to clients. Since the implementation is a prototype, there remains some room for improvement. However, it is not far away from a first version being commercially viable. Section 8 discusses the most important next steps that are necessary to make the visualization ready for a public release.

8 Conclusion

8.1 Summary

The goal of this thesis was to create a three-dimensional visualization of agent-based crowd simulations that supports analysts when evaluating simulation results. The visualization should be integrated into the BIM workflow in order to join BIM models and crowd simulations. Principles of Visual Analytics were collected from literature in order to design the visualization for optimal support of analysis.

A proof of concept for the visualization was implemented in the crowd simulation software crowd:it. It employs results from the research project BEYOND in order to convert IFC files to crowd:it projects. Then the three-dimensional structure of the model is recreated and visualized. Several tools were developed that allow to manipulate the visualization in order to gain an overview of the scene or focus on details. During the development, the implementation was continuously evaluated in the accu:rate software department.

Once the implementation was finished it was evaluated in the course of a user study. Five participants were questioned in a one-on-one interview style, both remotely and face to face. The evaluation has shown that the implementation in its current form is only a secondary addition to the toolset of an analyst. However, it also revealed that the visualization is very beneficial when presenting simulation results to clients.

8.2 Next Steps

Assemble a releasable product

There remain some indispensable adjustments that need to be implemented in order to turn the prototype into a commercially viable product. This includes some quality of life improvements, like moving the 3D view into the same window as the remaining crowd:it application, instead of opening it in a separate window. Then, the camera controls need some adjustments. The rotation target of the camera needs to be rendered into the scene. Ideally, the controls are reimplemented in a way that employs ray casting instead of using the depth buffer. This also allows the implementation of direct user interaction with the 3D scene, that is, it makes it possible to select objects and agents in the 3D view. Other, minor quality of life improvements that were pointed out during the evaluation may also be implemented.

One very important feature is a tool that allows exporting screenshots and videos. This already exists for the standard 2D view in crowd:it, but the 3D view is currently missing this feature. The most basic implementation of a video export would render a video from a fixed camera position, playing the entire simulation from start to finish. However, the same result can also be achieved by using a basic software for screen recording. Therefore, it is advisable to create a more advanced video export. Implementing a keyframe feature that allows to animate the camera is essential. Animating other components of the simulation may be beneficial, as well. However, this should first be evaluated in the means of a requirements analysis.

Furthermore, some of the rather experimental features need a rework. For example, the arrows that display the trajectory of an agent may be replaced with a more simple tube or line. Also, it should be discussed if the outlines feature should be included in its current form, or if a rework is necessary.

IFC

Once the first version of the product was released, it may be evolved in several areas. A tighter coupling with the IFC standard is arguably one of the most important areas of development, as also pointed out in 7.3. This does not only apply to the 3D visualization but regards the crowd:it application as a whole: IFC needs a deeper integration into the crowd simulation process. The necessary standardization is currently developed with BEYONDs project partners as important collaborators.

List of Figures

Listings

Bibliography

[1] URL: https://www.accu-rate.de/en/ (visited on 05/03/2022).

[2] M. Chraibi, A. Tordeux, A. Schadschneider, and A. Seyfried. “Modelling of Pedestrian and Evacuation Dynamics.” In: *Encyclopedia of Complexity and Systems Science*. Ed. by R. A. Meyers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 1–22. ISBN: 978-3-642-27737-5. DOI: 10.1007/978-3-642-27737-5_705-1.

[3] B. Kleinmeier, B. Zönnchen, M. Gödel, and G. Köster. “Vadere: An Open-Source Simulation Framework to Promote Interdisciplinary Understanding.” In: *Collective Dynamics* 4 (Sept. 2019), pp. 1–34. DOI: 10.17815/CD.2019.21.

[4] A. Kneidl. “Methoden zur Abbildung menschlichen Navigationsverhaltens bei der Modellierung von Fußgängerströmen.” Dissertation. München: Technische Universität München, 2013.

[5] S. Wang and G. Wainer. “A simulation as a service methodology with application for crowd modeling, simulation and visualization.” In: *SIMULATION* 91.1 (2015), pp. 71–95. DOI: 10.1177/0037549714562994. eprint: https://doi.org/10.1177/0037549714562994.

[6] URL: https://www.autodesk.eu/products/3ds-max/overview (visited on 05/03/2022).

[7] L. A. Toledo Diaz, I. R. Rivas, K. Rodriguez, and I. Rudomin. “Crowd Data Visualization and Simulation.” In: *Procedia Computer Science* 139 (2018). 6th International Conference on Information Technology and Quantitative Management, pp. 622–629. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.10.201.

[8] H. Pérez. “Crowd simulation and visualization.” PhD thesis. Universitat Politècnica de Catalunya. Departament d’Arquitectura de Computadors, 2019. DOI: http://hdl.handle.net/10803/667688.

[9] URL: https://www.thunderheadeng.com/pathfinder/ (visited on 05/03/2022).

[10] URL: https://www.thunderheadeng.com/wp-content/uploads/2013/08/pathfinder_elevators_600x461.png (visited on 05/10/2022).

[11] URL: `https://www.thunderheadeng.com/wp-content/uploads/2017/11/StadiumFire.png` (visited on 05/10/2022).

[12] URL: `https://www.thunderheadeng.com/wp-content/uploads/2015/05/ContourPlots2.png` (visited on 05/10/2022).

[13] P. C. Wong and J. Thomas. "Visual Analytics." In: *IEEE Comput. Graph. Appl.* 24.5 (Sept. 2004), pp. 20–21. ISSN: 0272-1716. DOI: `10.1109/MCG.2004.39`.

[14] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. "Visual Analytics: Scope and Challenges." In: *Visual Data Mining: Theory, Techniques and Tools for Visual Analytics*. Ed. by S. J. Simoff, M. H. Böhlen, and A. Mazeika. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 76–90. ISBN: 978-3-540-71080-6. DOI: `10.1007/978-3-540-71080-6_6`.

[15] N. Andrienko, G. Andrienko, G. Fuchs, A. Slingsby, C. Turkay, and S. Wrobel. "Introduction to Visual Analytics by an Example." In: *Visual Analytics for Data Scientists*. Cham: Springer International Publishing, 2020, pp. 3–25. ISBN: 978-3-030-56146-8. DOI: `10.1007/978-3-030-56146-8_1`.

[16] N. Andrienko, G. Andrienko, G. Fuchs, A. Slingsby, C. Turkay, and S. Wrobel. "Principles of Interactive Visualisation." In: *Visual Analytics for Data Scientists*. Cham: Springer International Publishing, 2020, pp. 51–88. ISBN: 978-3-030-56146-8. DOI: `10.1007/978-3-030-56146-8_3`.

[17] N. Andrienko, G. Andrienko, G. Fuchs, A. Slingsby, C. Turkay, and S. Wrobel. "General Concepts." In: *Visual Analytics for Data Scientists*. Cham: Springer International Publishing, 2020, pp. 27–49. ISBN: 978-3-030-56146-8. DOI: `10.1007/978-3-030-56146-8_2`.

[18] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. ESRI Press, 2011. ISBN: 9781589482616.

[19] URL: `https://www.axismaps.com/guide/visual-variables` (visited on 05/10/2022).

[20] T. Munzner. "Visualization Analysis and Design." In: A K Peters/CRC Press, 2014. ISBN: 9781466508910.

[21] URL: `https://libgdx.com/` (visited on 04/08/2022).

[22] URL: `https://www.khronos.org/api/opengles` (visited on 04/08/2022).

[23] URL: `https://github.com/locationtech/jts` (visited on 04/08/2022).

[24] URL: `https://www.blender.org/` (visited on 04/08/2022).

[25] D. Eberly. *Triangulation by Ear Clipping*. URL: `https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf` (visited on 04/08/2022).

[26] URL: `https://en.wikipedia.org/wiki/Back-face_culling` (visited on 04/14/2022).

[27] URL: `https://iquilezles.org/articles/distfunctions/` (visited on 04/08/2022).

[28] URL: `https://en.wikipedia.org/wiki/Blinn-Phong_reflection_model` (visited on 04/14/2022).

[29] URL: `https://en.wikipedia.org/wiki/Shadow_mapping` (visited on 04/14/2022).

[30] URL: `https://en.wikipedia.org/wiki/Discrete_Laplace_operator` (visited on 04/14/2022).

[31] D. H. Chung, R. S. Laramee, J. Kehrer, and H. Hauser. "Glyph-Based Multifield Visualization." In: *Scientific Visualization: Uncertainty, Multifield, Biomedical, and Scalable Visualization*. Ed. by C. D. Hansen, M. Chen, C. R. Johnson, A. E. Kaufman, and H. Hagen. London: Springer London, 2014, pp. 129–137. ISBN: 978-1-4471-6497-5. DOI: `10.1007/978-1-4471-6497-5_13`.

[32] URL: `https://en.wikipedia.org/wiki/Parallel_projection` (visited on 05/03/2022).

[33] URL: `https://ir.deutschebahn.com/en/db-group/strategy/unsere-strategie-starke-schiene/` (visited on 05/03/2022).