Real-Time Simulation of Crowd Disasters

Christoph Lürig
University of Applied Science Trier
Trier, Germany
luerig@hochschule-trier.de

ABSTRACT

In this paper, we explain a crowd simulation technique that allows real-time simulation of crowds, mainly for evacuation scenarios. Crowd disasters with deadly accidents often happen in places with high people density, sometimes in combination with additional factors like fire or terrorist attacks. Having an interactive simulator for these phenomena offers the possibility to create a planning tool and to create an educational simulation that explains the essential dynamics of crowd disasters. The approach is based on the simulation of crowd density and gets modeled by a partial differential equation. In contrast to an agent and particle-based simulations, this approach works with a minimal set of assumptions and is easy to parallelize on the graphics card in CUDA.

We base the simulation model on three assumptions. The first assumption is that everybody tries to get to the next exit in minimal time. This decision is made purely on the current situation, and no prediction about the future is involved. This assumption is modeled with the Eikonal equation. The second assumption is a relation between people density and magnitude walking velocity of people in crowd situations. This relation has been published. The third assumption is that the amount of people except for intentional spawning and despawning stays constant. The continuity equation expresses the third assumption.

To validate the model, we demonstrate its capability to replicate several known effects, which include density clogging in corner situations, shockwave propagation effect once the density has reached 5-6 persons per square meter, and the effect of abruptly narrowing corridors.

The simulation core is open source under the MIT license and available on GitHub.

ABOUT THE AUTHORS

Christoph Lürig has been a professor for game programming at the University of Applied Science in Trier since 2009. He is teaching almost everything related to game programming. His interests include independent and serious games as well as flow in cooperation with psychologists, game metrics, and simulation.

Before becoming a professor, he worked for ten years in the video game industry in diverse programming positions. These included five years in Canada. He got credited for AquaNox 1 / 2, Splinter Cell Essentials / Conviction, Ironman PSP, and Army of two 40th day.

Christoph Lürig holds a Ph.D .in computer science in the field of scientific data visualization.

Real-Time Simulation of Crowd Disasters

Christoph Lürig
University of Applied Science Trier
Trier, Germany
luerig@hochschule-trier.de

1 INTRODUCTION AND RELATED WORK

Crowd and panic disasters often happen in the context of panic stampedes [Jacobs and 't Hart, 1992], in life-threatening situations like fire or terrorist attacks [Drury et al., 2009], or for rushes for a good seat in a stadium [Elliott and Smith, 1993]. Accidents are happening because of people getting crushed or getting trampled down (see [Pretoriuset al., 2015] or [Helbing et al., 2007]). In this paper, we mainly focus on the second type of accident.

Pedestrian simulation and crowd dynamics are a vivid research area. [Drury and Reicher, 2000] approach the subject matter from a psychological point of view. Another way to approach the problem is more from a physical point of view [Helbing and Molnár, 1995]. In this paper, we take a physical point of view. A third direction is to follow it more from a computer science and architectural point of view (see [Kleinmeier et al., 2019] and [Seitz, 2016]).

Physics-based simulations can be implemented by cellular automata-based simulation models that are fast and easy to implement but suffer from drawbacks in high-density situations. Additionally, one has to pay attention to their update sequence (see [Chenney, 2004] or [Nishinari et al., 2003]).

The second category of models is particle-based models, where individuals are simulated as particles. Those simulations models are either force-based models (see [Hirai and Tarui, 1975] and [Helbing and Molnár, 1995]) or models based on evasion heuristics (see [Luo et al., 2017] and [Bruneau and Pettre, 2015]). More recently, the solution to the Eikonal equation is also often used. It returns, as a result, the minimal geodesic path, which can either be the shortest or the fastest path from any point in the area towards a destination area (see [Hartmann et al., 2012] and [Köster and Zönnchen, 2014]). The Eikonal equation also plays a crucial role in the simulation model presented here. The third approach is to model the crowd as a mass continuum that follows the continuity equation and is therefore motivated by the models used in computational fluid dynamics [Kenjereˇ s and Zwinkels, 2019]. They also use the Eikonal equation in combination with the continuity equation. In this paper, we follow the approach with the continuity equation.

The central quantity of our simulation model is crowd density measured in persons per square meter. One motivation of that idea is the concept of crowd pressure used by [Helbing et al., 2007], which correlates accident probability with the product of crowd density and variance of walking velocity in a specific area. From the two-dimensional scalar field representing the crowd density, the magnitude of the walking velocity can be estimated using the empirically estimated mapping from [Weidmann, 1995]. Using the empirically estimated function is also the major conceptual difference to [Kenjereˇ s and Zwinkels, 2019]. They use a simple linear ramp. The magnitude of the walking velocity and the geographical circumstances are fed into a solver of the Eikonal equation. The solver returns the time needed at every point in the field to get to its nearest exit. A couple of different approaches exist that show how to solve the Eikonal equation in the most efficient way by finding the right propagation direction for the Godunov upwind scheme [Dang and Emad, 2014]. In our implementation, a straightforward implementation of the finite difference Godunov upwind scheme proved to be sufficient and resulted in a slim CUDA implementation with little branch divergence. Most of the numerical implementations of this work in CUDA [Sanders and Kandrot, 2010] follow the recurrence pattern with geometric decomposition [McCool et al., 2012]. CUDA implementations can also be done for particle-based simulation, as done by [Wockenfuss and Lürig, 2011]. Problematic in this approach is the memory access to obtain information from other particles. The paper [Wockenfuss and Lürig, 2011] has solved the problem by artificially limiting the number of particles that can reside in a space cell.

We compute the velocity as a two-dimensional vector field using the gradient of the solution of the Eikonal equation. Taken this vector field, the crowd density, and the continuity equation, we compute the time derivative of the density field [Batchelor, 2000]. The time derivative gets integrated using an Euler integration scheme [Schwarz and Waldvogel, 1989].

As every simulation model, our model is based on some assumptions. In our case, these are three assumptions.

1. Each individual tries to get to the closest exit/destination point as fast as possible. The steering decision is made purely on the current situation without anticipating the future. This assumption gets represented in the Eikonal equation.
2. The magnitude of the walking velocity can be derived from the people's density because of the underlying empirical law. The relation between density and velocity is expressed in the density-to-velocity mapping function.
3. People move according to the velocity vector field, but the total mass is conserved. We represent this in the continuity equation.

To summarize, the overall computation process is as follows. We start with a given density distribution of people. The density distribution and all other values are defined on a grid with a cell size of 0.5m∗0.5m. This density gets mapped to a velocity magnitude for the people walking. This velocity magnitude gets low pass filtered because of numerical reasons, as will be explained in section 2. Afterward, the Eikonal equation gets solved, which results in the time from every position in the simulated area to the closest exit point. The two-dimensional vector-field describing the velocity gets essentially computed by taking the gradient of the solution of the Eikonal equation. The time derivative of the density field, which gets computed by the continuity equation, gets then integrated upon the density field. We use density and magnitude of velocity to calculate crowd pressure and accident probability.

The remaining of the article is structured as follows. In section 2, we will discuss the mathematical and numerical aspects of the problem. Section 3 explains the overall processing flow and specifics on how things get mapped in CUDA. Section 4 showcases some simulation results and serves as a point of validation. Finally, in section 5 we will discuss the limitations of the model and future extension possibilities.

2 NUMERICAL ASPECTS

All following quantities are functions that map from a position in two-dimensional space to either a scalar value or to a two-dimensional vectorial quantity. Assuming we have a given density distribution p the first step is to calculate the magnitude of the velocity

$$c = \|\vec{v}\|. \qquad (1)$$

This is done by applying the mapping function given in [Weidmann, 1995] $c = h(p)$. This mapping function is displayed in Figure 1. In our program, we use a piecewise linear approximation of that function.

The result gets low pass filtered with a standard gaussian and a kernel width of 5 cells. Doing so is necessary to suppress discretization artifacts. If one cell is occupied by six persons with the neighboring cells being empty, no movement would take place. Having an impact radius of the density on the velocity of one meter also makes sense taking the average diameter of a person into account. Areas that are occupied by obstacles like walls get defined to have zero velocity $c := 0$.

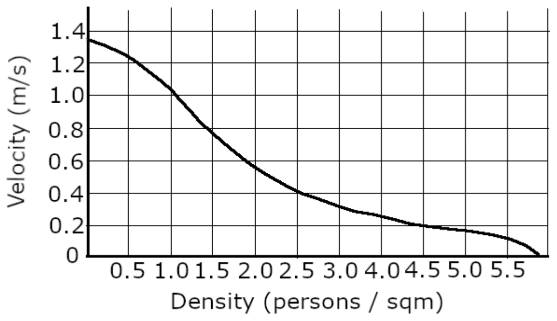

Figure 1. Mapping from density to velocity

Let $u \in (\Omega \rightarrow \mathbb{R})$ with $\Omega \subset \mathbb{R}^2$ be the time to get to the nearest exit. Ω represents the area where people can walk. Let $\Gamma \subset \Omega$ be the exits or destination points. Then $u = 0$ if $\vec{x} \in \Gamma$ as a Dirichlet boundary condition and for the Eikonal equation:

$$\|\nabla u\| = \frac{1}{c} \qquad (2)$$

As we want to keep the number of people constant in the simulation, we also have a Neumann boundary condition

$$\frac{\partial u(\vec{x})}{\partial \vec{n}} = 0; \vec{x} \in \partial\Omega \qquad (3)$$

with $\vec{n}$ being the normal of the boundary.
Equation (2) gets discretized over a uniform grid using the Godunov upwind scheme using the following equation:

$$\left[\left(u_{i,j} - u_{i,j}^{xmin}\right)^+\right]^2 + \left[\left(u_{i,j} - u_{i,j}^{ymin}\right)^+\right]^2 = \frac{1}{c^2} \qquad (4)$$

In equation (4) the following definitions hold:

$$u_{i,j}^{xmin} = \min\,(u_{i-1,j}, u_{i+1,j})$$
$$u_{i,j}^{ymin} = \min\,(u_{i,j-1}, u_{i,j+1})$$
$$(x)^+ = \max\,(x, 0)$$

The approximation of u in equation (2) is made as follows. As start values, we choose $u \coloneqq 0$ for all destination points and $u \coloneqq \infty$ for all others. Afterward, we apply iterations over the whole grid until convergence. In one iteration first, the two upwind values $u_{i,j}^{xmin}$ and $u_{i,j}^{ymin}$ are determined from the previous iteration. Then also with the values from the previous iteration, it is determined whether one or both of the summands of equation (4) become zero. If both become zero, we do nothing. If one becomes zero, the equation becomes resolved by $u_{i,j}$, which is used as the grid point value for the next iteration. If both are none zero $u_{i,j}$ gets computed by resolving the quadratic equation. If this equation has no solution in the domain of real numbers, we take the best partial derivative by updating

$$u_{i,j} \coloneqq \frac{1}{c} + min\left(u_{i,j}^{xmin}, u_{i,j}^{ymin}\right) \qquad (5)$$

Take a look at Figure 2 to get an intuitive understanding of the Eikonal equation. In this example, we assume constant velocity, have some walls inserted, and have defined the bottom pixel line as a destination.

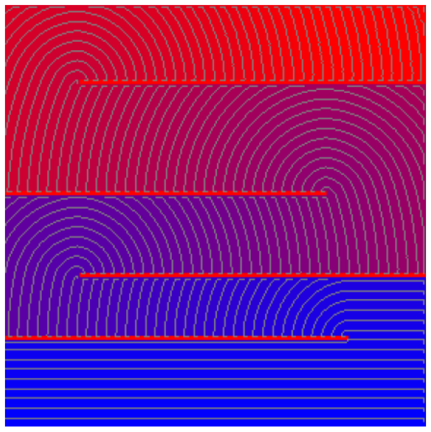

Figure 2. Solution of the Eikonal equation. Destination is the bottom line of the picture.

In the next step, the velocity field $\vec{v}$ can be computed. We walk against the gradient direction of u. Applying equation (1) and (2) we can derive

$$\vec{v} = -\frac{\nabla u}{\|\nabla u\|^2} \qquad (6)$$

To compute equation (6), usually, the central difference quotient is used. The exception is when one of the neighboring cells is a blocked element. In this case, one of the one-sided difference quotients gets applied.

As a next step, the time derivative of density p can be computed using the continuity equation

$$\frac{\partial p}{\partial t} = -\nabla \cdot (p\vec{v}) \qquad (7)$$

with $\nabla \cdot$ being the divergence operator which unfolds to

$$\nabla \cdot (p\vec{v}) = \frac{\partial p\vec{v}_x}{\partial x} + \frac{\partial p\vec{v}_y}{\partial y} \qquad (8)$$

The update of the density field p is finally determined by equation (7) and the Euler integration scheme

$$p(t + \Delta t) \approx p(t) - \Delta t\,(\nabla \cdot (p\vec{v})) \qquad (9)$$

Computing the partial derivatives in equation (8) and, therefore, also equations (7, 9) is numerically tricky. The obvious central difference quotient turns out to generate unstable results in the presence of strong gradients in density. The second problem manifests itself in the presence of walls. At walls, the density does not get updated, but one has to pay attention that we do not loose mass at the neighboring cells.

The problem with the central difference becomes obvious if we assume constant velocity in the field and a square block with a sharp density gradient. This situation is pictured in Figure 3.

At the point a, one would get a positive partial derivative of the density and in point b a negative one. Principally both should be zero as only the points left of them should get a partial derivative. We solve this problem by taking the one-sided difference quotient that is orientated against movement velocity. In the case of the derivative in the x-direction, this means

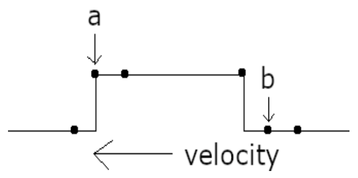

Figure 3. Problem with central difference quotient in the estimation of divergence.

$$\left(\frac{\partial p\vec{v}_x}{\partial x}\right)_{i,j} \approx \begin{cases} (p\vec{v}_x)_{i,j} - (p\vec{v}_x)_{i-1,j}; \vec{v}_x \geq 0 \\ (p\vec{v}_x)_{i+1,j} - (p\vec{v}_x)_{i,j}; \vec{v}_x < 0 \end{cases} \qquad (10)$$

The partial derivative in y direction is analog. The problem remaining is the case that we have a wall in the direction of the velocity vector. Applying equation (10) gives incorrect results because it assumes a drain of people into the direction of the wall. To fix this problem, the negative component of the side differences in equation (10) are left out in this case. Meaning if we have $wall_{i-1,j}$ and $\vec{v}_x < 0$ we assume

$$\left(\frac{\partial p\vec{v}_x}{\partial x}\right)_{i,j} \approx (p\vec{v}_x)_{i+1,j} \qquad (11)$$

On the other hand, if we have $wall_{i+1,j}$ and $\vec{v}_x > 0$ we assume

$$\left(\frac{\partial p\vec{v}_x}{\partial x}\right)_{i,j} \approx -(p\vec{v}_x)_{i-1,j} \qquad (12)$$

Both equations (11, 12) can be formally derived from Gauss integral sentence by regarding the border of the cell, that lies close to the wall.

The last point missing is the quantity named crowd pressure. Crowd pressure as defined in the work of [Helbing et al., 2007] as the product of crowd density and variance of velocity. As variance is the sum of squared distances of the sample point from their average, we use the squared magnitude of the gradient of the velocity magnitude field c here as an estimator. We use the definition of crowd pressure cp as follows:

$$cp \coloneqq p\,\|\nabla c\|^2 \qquad (13)$$

As we have discussed the mathematical and numerical aspects of the simulation, the following section 3 will discuss the overall data flow of the program and take a closer look at some aspects of the CUDA implementation.

3 FLOW AND IMPLEMENTATION IN CUDA

The overall process flow of the simulation core is as follows. First, we initialize the crowd density to the starting conditions. Then iterations are performed for every simulation step. The first step in the simulation handles the spawning and despawning of people. In target areas, people get despawned, meaning the density of the people gets set to zero. Spawning means that the density of the people gets increased by a certain amount in designated spawn areas. These are areas where people are continuously entering the simulation. In the second step, the velocity magnitude gets computed. We do this calculation with the mapping function from density to the magnitude of velocity. Afterward, the velocity gets low pass filtered. At this point, also, the crowd pressure gets computed.

The resulting velocity field gets fed into the module that uses the Eikonal equation solver to compute the time needed to get to the nearest exit. The resulting time field also gets low pass filtered with one pass reaching out *0.5m* in every direction. Low pass filtering has proven to help suppress numerical artifacts that may occur at discretization boundaries that may happen during the computation of the solution of the Eikonal equation, as explained in the following paragraph. The velocity vector field gets computed from the gradient of this time field. This one gets fed into the integrator of the Euler scheme, which also computes the time derivative of the density using the continuity equation. After this last processing step, the simulation is advanced one time step into the future.

We do all computations on the GPU in CUDA. We use the CPU only to issue the kernel invocations. In the demo program, even memory transfer is not required anymore after the initial setup. We visualize the results also in CUDA. The overall processing strategy in all steps is pretty similar. All steps usually update one grid point in the numerical discretization and require the information of all neighbor points. We do the processing by CUDA kernels in blocks of 32 * 32 threads that copy their required processing values in tiles of 34 * 34 elements into shared memory. Two specialties are the solver of the Eikonal equation and the integrator of the Euler scheme. For convergence, the solver of the Eikonal equation needs many iterations ranging from 150 to 5000, depending on the situation. To make things more efficient, every block performs 32 iterations before writing shared memory back to the device memory. This way, we increase the arithmetic intensity of the computation. On the flip side, we get slight numerical artifacts at the border of the tiles mentioned in the previous paragraph. During iteration, the maximum change at a single grid point over the overall domain gets computed to check whether further iterations are needed. If no more iterations are needed, further kernel invocations in CUDA get an early out. As for solving the continuity equation and updating the density field, we use the wall clock time passed since the last frame. Should that time get above *10ms* sequential integration steps for the crowd density are done with every integration step not using more than *10ms*. This way, we avoid accuracy problems if the frame rate should drop too low.

In the following section 4, we will take a look at some typical examples for known crowd effects and also take a look at some performance figures.

4 RESULTS

In this section, we will discuss some typical examples of crowd situations and analyze some performance figures. We did all computations on an NVIDIA GTX 1650.

The first typical test case is displayed in Figure 4. People are walking around a corner from the upper left to the lower right. The corridor width is *5m*; the two segments are *30m* each. People enter at *0.875 persons / second*. The image contains the visualization of the crowd density. The phenomenon that is of relevance here and which is often used as a test case for crowd simulations is the clogging effect at the corner. On the one hand, people try to take the shortcut around the corner. On the other hand, that slows down the walking velocity so that way, people also start taking the longer but potentially faster way on the outside of the corner.

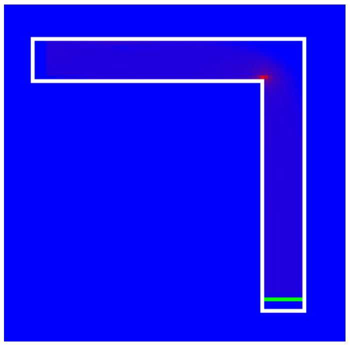
Figure 4. Demonstrating the clogging effect at the corner

In Figure 5 three different evacuation scenarios are compared. All scenarios contained the same amount of people. Figure 5a shows the crowd density and Figure 5b shows a visualization of the crowd pressure. Crowd density is visualized on grayscale here for orientation. One can see that the corridor of constant width gives the smoothest scenario as one would expect. The abruptly narrowing corridor induces strong congestion just at the entry point. Here one can also see a dangerously high crowd pressure making accidents at the point likely. The slowly narrowing corridor is better than the suddenly narrowing one. We can also observe this effect at successively decreasing speed limits ahead of road constructions. On the narrowing corridor, one can also see the formation of the typical shock waves once the density goes into the region of *5 - 6* persons per square meter.

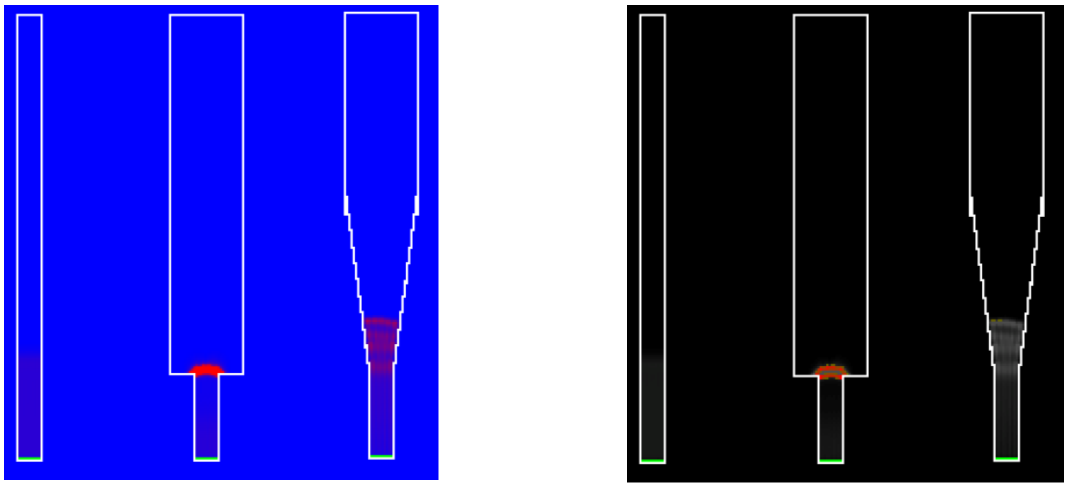
(a) Crowd density **(b) Crowd pressure overlaid with density**

Figure 5. Comparison of three different evacuation scenarios. All contain the same amount of people.

Figure 6 shows a stream of people coming from the top part of the area walking through the two gates to get to the bottom part of the area. The simulation area spans *144m * 144m*. The gates have a width of *17m*. *5.5* persons per second enter the scenario. The two images show that there is a time alternating congestion at the two gates. An effect one also knows from lane hopping cars in traffic jams or different ques at supermarket cashiers. The images also contain the visualization of the solution of the Eikonal equation in the form of iso-lines. Here one can see the effect of the assumption that people do not anticipate the future in decision making for the way through the gates. When one gate is free, most people will run in that direction, making it clogged once they arrived there.

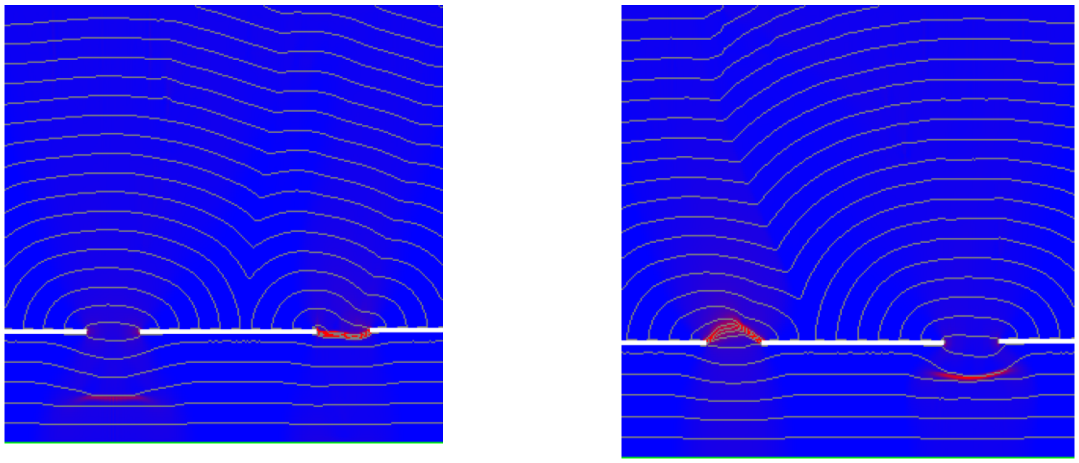
Figure 6. Oscillation effect at two different exit gates.

As a more practical example, we have also remodeled the Love Parade disaster. That was an electronic dance music festival that took place on 24 July 2010 in Duisburg, Germany. During the disaster, 21 people died because of suffocation diagnosed afterward by crushed rib cages, and 500 people were injured. We remodeled the scene according to the map shown by [Helbing and Mukerji, 2012] and by aerial pictures from Google Maps. A side stair just behind the T junction in the entrance area played a crucial role in the disaster. As [Helbing and Mukerji, 2012] have explained, the deadly accidents did not happen because of people falling off the stairs but by a phenomenon called crowd turbulence. Running this scenario through the simulator, the region in front of the stairs, where the deadly accidents happened, gets flagged by high crowd pressure, as shown in Figure 7. Figure 8 shows the density visualization near the stairs during various times and demonstrates the turbulence effect. These turbulences are similar to the effect shown in Figure 6 by a conflict between leaving the premise over the stairs and getting towards the end of the ramp.

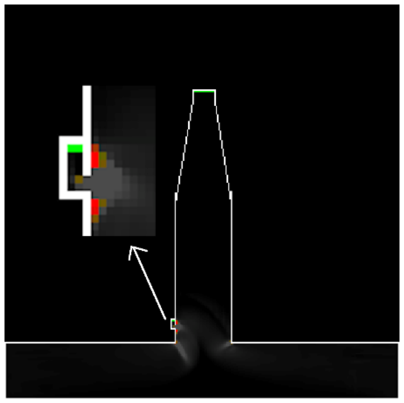
Figure 7. Crowd pressure in front of the stairs at the Love Parade disaster.

We constructed a stress test example with a relatively long corridor in a *144m * 144m* area to measure the performance of the system. We show this scenario in Figure 9. People are entering the scenario from the top and leaving on the bottom. On the implementation computer with an NVIDIA GTX 1650 card, the program runs at *108 fps*, meaning we have a frame time of *9.25 ms*. Because of the structure, the program is entirely GPU bound.

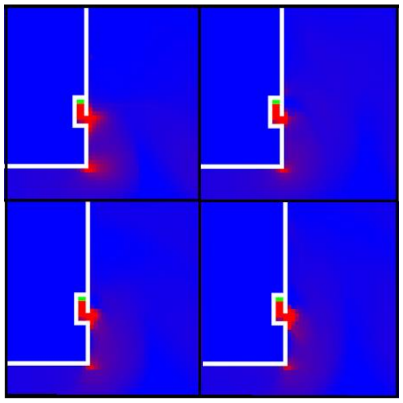
Figure 8. Turbulences close to the stairs at the Love Parade disaster

Running the program through the profiler "nsight compute" the time consumption of the different steps listed in table 1 shows up. Summing all those values up in that table gives a total time of *5.7 ms*, where the 46 iterations of the Godunov upwind scheme take with *5.1 ms* the largest share. As explained in section 3 explained every iteration here results in 32 iterations on the grid internally. The remaining *3.55 ms* to the total frame time of *9.25 ms* max comes from the three potential effects. The first effect is that we do not include visualization in table 1. As a secondary effect, the time OpenGL uses for rendering comes on top. As a third effect, an additional slowdown may be caused by the frame profiler used to measure the total frame time.

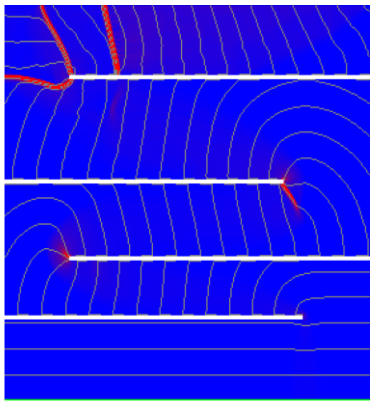

Figure 9. Stress test example for crowd simulator.

In this section, we have validated the model for crowd behavior by showing that it can replicate a couple of known effects. We have also shown that the parallelization on a GPU is very efficient. As the program could use even more streaming processors, time consumption on more powerful graphics cards gets even lower. In the next and last section 5 we will discuss some limitations and extension possibilities of the model.

Table 1. The runtime of different steps

Processing step	Time (μs)
spawning and despawning	17.9
mapping to the magnitude of the velocity	13.6
2 low pass filter of velocity	2 * 25.0
compute crowd pressure	21.1
preparation of Eikonal solver	22.0
46 iterations of upwind scheme	46* 111
154 empty iterations on the upwind scheme	154 * 3.0
1 low pass filter time	24.2
computing velocity (vector field)	27.0
integrate the equation of motion	30.0

5 DISCUSSION AND FUTURE WORK

In this paper, we have explained an efficient crowd simulator that makes minimal assumptions on the behavior of crowds and essentially formulates the problem as a nonlinear partial differential equation. We have shown its validity by its capability to replicate known effects of crowd behavior.

As with every model, however, there are limitations. The most obvious one is that in this model, everybody would have the same objective. Everybody having the same objective is feasible for evacuation situations but would run into severe limitations if one would like to simulate the usual pedestrian traffic in a subway station where every pedestrian might have a different objective. Simulating people with different objectives is where particle-based simulations shine because modeling an individual objective comes here naturally. To extend our approach for people with different objectives, one could run independent simulations for every interest group with their respective objectives. The only difference is the point where we determine the magnitude of the moving velocity of the people. Here we take the sum of the density of all simulations into account.

Into the same direction goes the separation of people being in a panic state and not being in a panic state. People in panic will probably choose a higher desired walking velocity than *1.34m/s*. One would have to determine what the correct density to walking velocity mapping for that group of persons would be. Once we have found a correct mapping, two separate but linked simulations like in the case with different objectives could be run that simulate the group of people being in a panic and not being in a panic.

As for practical applications of the system, three fields look promising. First, there is the possibility to turn it into a serious game to teach about the dynamics of crowd disasters. Second, we can use this as a planning tool for evacuation situations. Finally, it may be used in combination with surveillance cameras to predict potential critical situations in the next couple of minutes.

The source code of the simulator discussed in this paper is available under MIT license on GitHub (https://github.com/Carbonfreezer/PanicSimulator).

ACKNOWLEDGEMENTS

I would like to thank Hans-Peter Beise for reading the initial draft of the paper and giving advice.

REFERENCES

[Batchelor, 2000] Batchelor, G. K. (2000). An Introduction to Fluid Dynamics. Cambridge Mathematical Library. Cambridge University Press.

[Bruneau and Pettre, 2015] Bruneau, J. and Pettre, J. (2015). Energy-efficient mid-term strategies for collision avoidance in crowd simulation.

[Chenney, 2004] Chenney, S. (2004). Flow tiles. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Goslar, DEU. Eurographics Association.

[Dang and Emad, 2014] Dang, F. and Emad, N. (2014). Fast iterative method in solving eikonal equations: A multi-level parallel approach. Procedia Computer Science, 29:1859 – 1869. 2014 International Conference on Computational Science.

[Drury et al., 2009] Drury, J., Cocking, C., and Reicher, S. (2009). The nature of collective resilience: Survivor reactions to the 2005 london bombings. International Journal of Mass Emergencies and Disasters, 27(1):66–95.

[Drury and Reicher, 2000] Drury, J. and Reicher, S. (2000). Collective action and psychological change: The emergence of new social identities. British Journal of Social Psychology, 39(4):579–604.

[Elliott and Smith, 1993] Elliott, D. and Smith, D. (1993). Football stadia disasters in the united kingdom: learning from tragedy? Industrial and Environmental Crisis Quarterly, 7(3):205–229.

[Hartmann et al., 2012] Hartmann, D., Mille, J., Pfaffinger, A., and Royer, C. (2012). Dynamic medium scale navigation using dynamic floor fields.

[Helbing et al., 2007] Helbing, D., Johansson, A., and Al-Abideen, H. Z. (2007). Dynamics of crowd disasters: An empirical study. Phys. Rev. E, 75:046109.

[Helbing and Molnár, 1995] Helbing, D. and Molnár, P. (1995). Social force model for pedestrian dynamics. Phys. Rev. E, 51:4282–4286.

[Helbing and Mukerji, 2012] Helbing D, Mukerji P. (2012) Crowd disasters as systemic failures: Analysis of the Love Parade disaster. EPJ Data Science, 2012, 1(1): 1−40.

[Hirai and Tarui, 1975] Hirai, K. and Tarui, K. (1975). A simulation of the behavior of a crowd in panic. Proc. of the 1975 International Conference on Cybernetics and Society.

[Jacobs and 't Hart, 1992] Jacobs, B. and 't Hart, P. (1992). Hazard Management and Emergency Planning, chapter 10. James and James Science.

[Kenjereˇ s and Zwinkels, 2019] Kenjereˇ s, S. and Zwinkels, S. (2019). Numerical modeling of the macroscopic behavior of a crowd of people under emergency conditions triggered by an incidental release of a heavy gas: an integrated approach. Flow, Turbulence and Combustion, 103(4):1081–1107.

[Kleinmeier et al., 2019] Kleinmeier, B., Zönnchen, B., Goedel, M., and Koeester, G. (2019). Vadere: An open-source simulation framework to promote interdisciplinary understanding. Collective Dynamics, 4:1–34.

[Köster and Zönnchen, 2014] Köster, G. and Zönnchen, B. (2014). Queuing at bottlenecks using a dynamic floor field for navigation. Transportation Research Procedia, 2:344 – 352. The Conference on Pedestrian and Evacuation Dynamics 2014 (PED 2014), 22-24 October 2014, Delft, The Netherlands.

[Luo et al., 2017] Luo, L., Chai, C., Ma, J., Zhou, S., and Cai, W. (2017). Proactivecrowd: Modelling proactive steering behaviours for agent-based crowd simulation: Modelling proactive steering for agent-based crowd simulation. Computer Graphics Forum, 37.

[McCool et al., 2012] McCool, M., Reinders, J., and Robison, A. (2012). Structured Parallel Programming: Patterns for Efficient Computation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

[Nishinari et al., 2003] Nishinari, K., Kirchner, A., Namazi, A., and Schadschneider, A. (2003). Extended floor field ca model for evacuation dynamics. IEICE Transactions on Information and Systems, E87-D.

[Pretorius et al., 2015] Pretorius, M., Gwynne, S., and Galea, E. R. (2015). Large crowd modelling: an analysis of the Duisburg love parade disaster. Fire and Materials, 39(4):301–322.

[Sanders and Kandrot, 2010] Sanders, J. and Kandrot, E. (2010). CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 1st edition.

[Schwarz and Waldvogel, 1989] Schwarz, H.-R. and Waldvogel, J. (1989). Numerical Analysis: A Comprehensive Introduction. John Wiley & Sons Ltd.

[Seitz, 2016] Seitz, M. J. (2016). Simulating pedestrian dynamics. Dissertation, Technische Universität München, München.

[Weidmann, 1995] Weidmann, U. (1995). Der Fahrgastwechsel im öffentlichen Personenverkehr. PhD thesis, Zürich.

[Wockenfuss and Lürig, 2011] Wockenfuss, F. and Lürig, C. (2011). Introducing Congestion Avoidance into CUDA Based Crowd Simulation. In Bender, J., Erleben, K., and Galin, E., editors, Workshop in Virtual Reality Interactions and Physical Simulation ”VRIPHYS” (2011). The Eurographics Association.