Contagion Modeling with PLE and UIC

Eva Brayfindley, Lusine Kamikyan, Matt Lam, John Wu, Yabin Zhang

September 10, 2017

1 Introduction

Large crowds reacting to a stimulus is a common occurrence in everyday life. Examples range from crowds flooding stores on Black Friday to a herd escaping from a predator. Various models have been proposed to simulate rudimentary crowd dynamics, only few of which include communication beyond position and velocity. In this report we summarize adaptations of two such models to incorporate the effects and contagion of fear.

There are several levels of models for large crowds: agent-based, kinetic and macroscopic. Roughly speaking, agent-based models simulate individuals, whereas kinetic and macroscopic models simulate the crowd as a single entity. We focused on two variants of agent-based models, referred to as the Aggregate and PLE. In the aggregate model, dense crowds of goal-oriented persons are simulated via individual path planning in conjunction with a density constraint to avoid collisions. Fear levels dictate movement speed and evolve towards a local average, i.e. equilibrate to surrounding levels. In the PLE model, crowds of goal-oriented agents are moving according to their most biomechanically efficient paths. Collisions are avoided using an optimal reciprocal collision avoidance algorithm. Fear levels act as a switch to indicate whether the energy function used is for walking or running, and thus dictate the speed of the agent. Similar to Aggregate, the fear evolves towards a local average.

2 Models

2.1 1D ASCRIBE Model

2.1.1 Agent-based

In the agent-based model, velocity is proportional to fear. The change in velocity then can be represented as a change in fear. From this, the governing system of differential equations can be written[1].

$$\begin{aligned} \dot{x} &= q_i \\ \dot{q}_i &= \gamma(q_i^* - q_i) \end{aligned} \tag{1}$$

where γ is an equilibration rate, and q_i^* is the weighted average of emotion across the set of points that interact with agent i.

$$q_i^* = \frac{\sum_{j \in G_i} w_{ij} q_j}{\sum_{j \in G_i} w_{ij}} \tag{2}$$

2.2 Global Planning Tool

We assumed that each particle has a goal that is reached by using a global path planner. For our global planner, we modeled the possible paths that an agent takes by assuming that there are regions of discomfort, which can be modeled by a function of $F(x)$ and is commonly referred to as the discomfort function of an environment. If $F(\widetilde{x}) \geq F(\widetilde{x_0})$, then a particle would rather move through point x_0. These ideas lead us to the constraints for a shortest path: a particle will choose the path that minimizes the discomfort encountered on the way to the goal position. So an agent can obtain this path by solving the following line integral problem:

$$S = \arg\min_f \int F(f(t)) \| f'(t) \| dt$$

where S is the agent's preferred path, f is the path that a particle takes, and F is the discomfort function of the environment. An example of the discomfort function can be seen in figure 1.

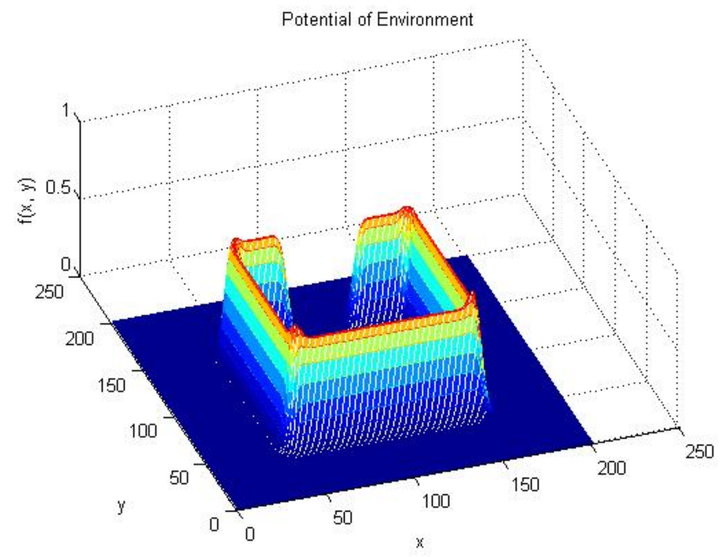


Figure 1: Example of a discomfort function. The high levels of discomfort represents an obstacle for the agent

It turns out that the path of this function is equivalent to the gradient descent path of the solution to the Eikonal equation [7], which is defined as:

$$\begin{aligned} \|\nabla u(x)\| &= C(x) \\ u(X_0) &= 0 \end{aligned}$$

where, X_0 is the set of initial positions of the particles and C is the speed function. We define the speed function as $C(x) = 1 - \frac{F(x)}{\sup(F(x))}$. Intuitively, $\|\nabla u(x)\|$ represents the speed of a particle at x. Therefore, at levels of high discomfort, the particles should move slow and C should have a low value at that point.

To find the shortest path, a shortest path algorithm derived from graph theory can be used. This can be done by fitting the environment into a mesh grid of points, and then at each grid point, calculate the discomfort F at that point. The edges of the grid represent an interpolated value of the function F for the edge. Denote $D_{s_0 j}$ and W_{xy} as the distance between point s_0 and j and W_{xy} as the weighted edge value of node x and y. After obtaining these edges, a shortest path algorithm such as Dijkstra's agorithm can be used to find the path that leads a particle to its goal. The 2013 group [2] demonstrated this technique by simulating the Asiana Plane crash and modeling agents evacuating the plane using the method. According to Sethian[12] and Peyre [8] , Dijkstra's algorithm produces a L1 solution to the problem, which implies skewed computational accuracy, and cannot converge to the solution of a continuous Eikonal problem. Despite this, it can been shown through the 2013 REU report that Dijkstra's algorithm along with a spline interpolation technique to smooth the generated path can produce satisfying results [2]. The following is pseudocode for Dijkstra's Algorithm [13]:

- Intial Step: Set $\begin{cases} D_{i,i} = 0, & \text{starting point label as red} \\ D_{i,j} = \infty, & \text{otherwise label as green} \end{cases}$

- Iterative Step:
  - Green points adjacent to red points become yellow.
  - Yellow points are updated using $D_{i,j} = \min_k \left(D_{ik} + W_{jk} \right)$
  - The Yellow point with the smallest D becomes red.

- Stop when all points are red. All red points become the smallest distance from point i to point j.

In order to improve the accuracy for computing the shortest path, we used another classical technique called Fast Marching. According to Peyre [8], Fast Marching does this by using upwind difference operators to obtain a more accurate approximation of the discomfort function. The solution of the Eikonal problem, u, can be discretized similarly to Dijkstra's problem and each value of $u_{k,l}$ can be approximated by the following [7]:

$$W_{k,l} = ||\nabla u_{k,l}||^2 = \max(u_{k+1,l} - u_{k,l}, u_{k-1,l} - u_{k,l}, 0)^2 + \max(u_{k,l+1} - u_{k,l}, u_{k,l-1} - u_{k,l}, 0)^2 \tag{3}$$

Fast marching for solving $u_{k,l}$ can be computed by the following algorithm:

- Intial Step: Set $\begin{cases} u(x) = 0, x \in X_0 \text{ label as red} \\ u(x) = \infty, x \in \Omega \setminus X_0 \text{ label as green} \end{cases}$

- Iterative Step:
  - Green points adjacent to red points become yellow.
  - Yellow points are updated using 3 by solving for $u_{k,l}$
  - The Yellow point with the smallest u becomes red.

- Stop when all points are red

After obtaining the numerical solution of the Eikonal equation, gradient descent is used to compute the shortest path [8]. Intuitively, the instantaneous change of path starting from the goal is associated to the gradient of u. Specifically, it is defined as:

$$\psi'(t) = \frac{-\nabla u(\psi(t))}{||\nabla u(\psi(t))||}$$

For simplicity, $\psi'(t)$ is approximated with a first order derivative approximation. Thus, the path can be solved dynamically at different timesteps:

$$\psi_{k+1} = \psi_k - \tau \frac{-\nabla u(\psi_k)}{||\nabla u(\psi_k)||}$$

where k is the time step index. Also, we defined S as the path that gives the mininum weighted distance from the starting point to the goal. This can be seen as $S(t) = \psi(1-t)$. From these data points, the preferred velocity can be easily computed by finding $\frac{dS(x)}{dt}$, which is aproximated by $\frac{S(x(j))-S(x(0))}{\tau}$, where j is the j^{th} value of the path.

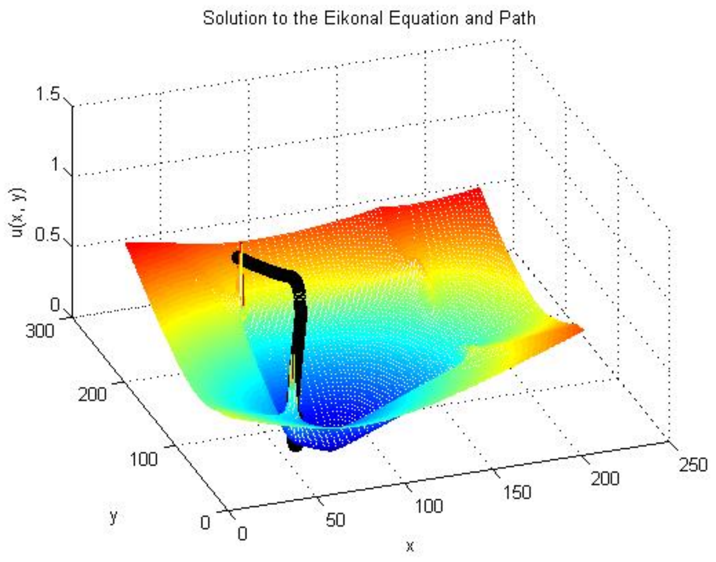


Figure 2: Solution to the Eikonal Problem of Figure 1 with an illustration of the shortest path

2.2.1 Numerical Simulation

In order to use Fast Marching and compute the optimal path, we used a Matlab Fast Marching toolbox created by Gabriel Peyre [9]. This toolbox provided fast computation because the toolbox called C++ functions. Unfortunately, computing the optimal paths cannot be vectorized in Matlab unless all agents have the same starting position, due to the fact that the boundary condition of the Eikonal equation is based on the location of the starting position. Also, if the agents have their goals at the same position, then their paths can be computed in parallel.

Additionally, since solving the Eikonal equation can give us the mininum shortest path, we can model the velocity of each agent as:

$$\dot{x_i} = \dot{S_i}$$

where $\frac{dS(x)}{dt}$ is the preferred velocity over the ith agent path $\dot{S}$. Thus, the movement can be defined as:

$$x_i^{(k+1)} = x_i^{(k)} + \dot{S_i}^{(k)} \triangle t$$

With this equation, we modeled agents trying to escape from a box obstacle with an opening. To create the discomfort function of this model, we used a binary image of a box, where the black values represents high levels of discomfort and smoothed the edges with a smoothing technique from the Image Processing Toolbox. Also, for some cases, we wanted particles to avoid colision with each other. These results can be shown in a video. To do this, the location of each particle contains high levels of discomfort and this function forms a Gaussian function.

2.2.2 Numerical Results

We tested the speed of the Fast Marching toolbox. It turns out that the computation speed depends on the number of grid cells, complexity of environment and the distance between an agent and their goal. With a blank environment in a 500-by-500 grid, path planning took on average 0.6 seconds per agent at the first time step. However, at the 50th time step when agents are closer to their goals, the average time was 0.4 seconds. This simulation is shown in figure 2.2.2, along with a simulation of particles exiting a box containing an obstacle in figure 2.2.2.

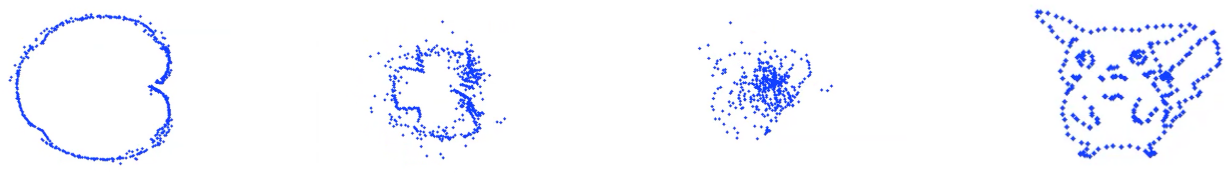


Figure 3: Snapshots of a simulation with starting positions forming a cardiod and goal positions forming a Pikachu.

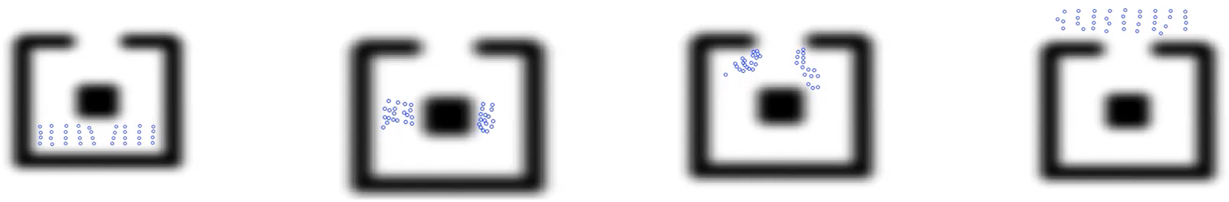


Figure 4: Shapshots of a simulation of particles trying to escape from a box

2.3 The Aggregate Model

The Aggregate Model[10] focuses on the simulation of inter-agent dynamics of large, dense crowds, which exhibit a low interpersonal distance and a corresponding loss of individual freedom of motion. It uses properties of the macroscopic flow of the crowd to correct individual velocities for collision avoidance.

2.3.1 Assumptions

Assumption 1. Each agent has a goal and determines the optimal path to its goal based on the static obstacles.
Assumption 2. Each agent adjust its velocity according to the crowd density to avoid collision.

2.3.2 Unilateral Incompressibility Constraint

The ASCRIBE model discussed above allows agent densities to become arbitrarily large - infinite, in fact, in the case of crossing trajectories. The unilateral incompressibility contraint (UIC) offers a solution to this density blow-up in both the ASCRIBE model and its continuum limit. It is based on the concept of modeling the crowd as an incompressible fluid above a critical density, effectively maintaining an upper bound on agent density.

A common feature of models with interacting agents is $O(n^2)$ running time in the number of agents, due to the quadratic growth of their number of pairs. The UIC stands out in this regard, having linear complexity in the number of agents. This asymptotic bound also extends to the number of static and moving obstacles as well, and in practice has the implied constant dominated by path planning.

2.3.3 Continuous Representation

We first transfer information from discrete agents to the grid by assigning each agent a Gaussian function of the form:

$$f(x) = e^{-x^2} \tag{4}$$

Then, we evaluate the following function at each grid point:

$$F_\xi(y) = \sum_i \alpha_\xi(x_i) f\left(\frac{\text{dis}(x_i, y)}{\beta}\right) \tag{5}$$

- ξ: Indicate the macroscopic parameter that is calculated. In our model, ξ is density, velocity field, or fear field;
- β: Determine how localized the Gaussian is;
- $\alpha_\xi(x_i)$: Scale the Gaussian function according to the following:

$$\alpha_\xi(x_i) = \begin{cases} 1, & \text{if } \xi = \rho, \text{density, or } \xi = q, \text{fear field;} \\ \mathbf{v}_{i,1}, & \text{if } \xi = \mathbf{v}_x, \text{velocity on the } x-\text{direction;} \\ \mathbf{v}_{i,2}, & \text{if } \xi = \mathbf{v}_x, \text{velocity on the } y-\text{direction;} \end{cases}$$

- $\text{dis}(x, y)$: Calculate the Euclidean distance between two points;

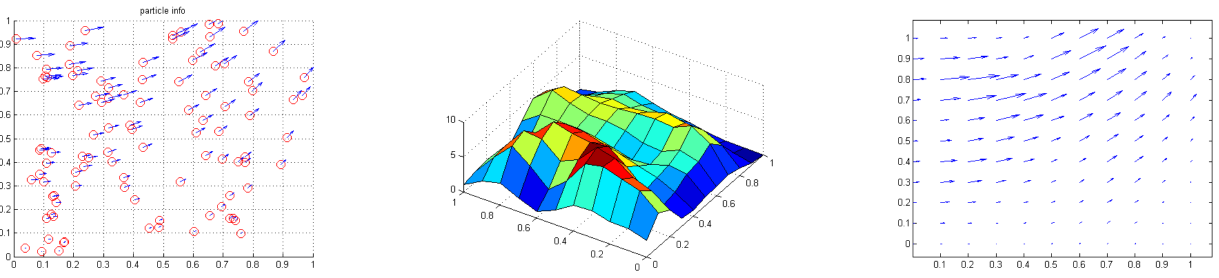


Figure 5: Left to Right: particle distributions, density plot, velecity field

2.3.4 UIC Computation

We wish to calculate adjusted velocities that are close to the agents' preferred velocities, but such that the density constraint is upheld. It may be assumed that agents will attempt to maximize progress towards their goals, i.e. in their preferred directions, thus the adjusted velocity will maximize $\int \rho \mathbf{v} \cdot \tilde{\mathbf{v}}$ where $\tilde{\mathbf{v}}$ and $\mathbf{v}$ are the preferred and adjusted velocities respectively. According to Narain [10], it then follows that

$$\mathbf{v} = v_{\max} \frac{\tilde{\mathbf{v}} - \nabla p}{\|\tilde{\mathbf{v}} - \nabla p\|},$$

where p is a pressure-like quantity with the property

$$\begin{cases} \rho < \rho_{max} & \Rightarrow \quad p = 0 \\ p > 0 & \Rightarrow \quad \nabla \cdot \mathbf{v} = 0 \end{cases}. \tag{6}$$

In words, the "pressure" is zero when the density is below a predetermined critical value, beyond which the pressure acts to make the velocity field divergenceless and the crowd, therefore, incompressible.

This velocity adjustment is handled in two steps: 1) computation of the pressure gradient ∇p followed by 2) renormalization to the maximum speed. Clearly, the bulk of the work lies in step 1. To begin, the velocity $\tilde{\mathbf{v}} - \nabla p$ is substituted into the continuity equation. This yields

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \tilde{\mathbf{v}}) + \nabla \cdot (\rho \cdot \nabla p),$$

where time discretization leads to solving

$$\rho^{n+1} = \rho^n - \nabla \cdot (\rho^n \tilde{\mathbf{v}}^n)\Delta t + \nabla \cdot (\rho^n \nabla p^n)\Delta t \tag{7}$$

at each timestep. Here, the superscripts index the timestep and the unknowns are ρ^{n+1} and p^n. Using the property (6), equation (7) may be recast as a linear complementary problem. In particular, setting

$$\begin{aligned}\sigma^n &= \rho_{\max} - \rho^n,\\ \mathbf{A} &= -\Delta t \nabla\cdot\rho^n\nabla,\\ \mathbf{x} &= p^n,\\ \mathbf{b} &= \sigma^n + \nabla\cdot(\rho^n\tilde{\mathbf{v}}^n)\Delta t\end{aligned}$$

transforms (7) into

$$\begin{gathered}\mathbf{Ax}+\mathbf{b}\geq 0;\\ \mathbf{x}\geq 0,\ \mathbf{x}^T(\mathbf{Ax}+\mathbf{b}) = 0.\end{gathered} \tag{8}$$

Note, however, that this transformation involves writing as vectors the pressure and density profiles, as well as the velocity field. That is, the two dimensional arrays containing scalar pressures and densities, and the two dimensional array containing vector valued velocities, must be in their entireties converted into vectors. It also requires us to encode the linear operator $\nabla\cdot\rho^n\nabla$ as a matrix to act on said vector forms.

Assume, for notational simplicity, that the grid imposed on our domain contains n cells along both x and y directions. It is then natural to reshape the pressure and density profiles each into n^2 dimensional vectors - however, the velocity field contains $2n^2$ elements and its vector representation must be compatible with those of the pressure and density profiles. Thus we duplicate the pressure and density profiles so that from a physical standpoint there exist (equal) x and y directional pressures and densities. These reshapings are more explicitly written below.

$$v_{\text{profile}} = \begin{bmatrix}(x_{11},y_{11}) & \cdots & (x_{1n},y_{1n})\\ \vdots & \ddots & \vdots\\ (x_{n1},y_{n1}) & \cdots & (x_{nn},y_{nn})\end{bmatrix} \mapsto \begin{bmatrix}x_{11}\\ x_{21}\\ \vdots\\ x_{n1}\\ x_{12}\\ \vdots\\ x_{nn}\\ y_{11}\\ \vdots\\ y_{nn}\end{bmatrix}, \quad \rho_{\text{profile}} = \begin{bmatrix}\rho_{11} & \cdots & \rho_{1n}\\ \vdots & \ddots & \vdots\\ \rho_{n1} & \cdots & \rho_{nn}\end{bmatrix} \mapsto \begin{bmatrix}\rho_{11}\\ \rho_{21}\\ \vdots\\ \rho_{n1}\\ \rho_{12}\\ \vdots\\ \rho_{nn}\\ \rho_{11}\\ \vdots\\ \rho_{nn}\end{bmatrix}$$

Observe that each contiguous block of n entries in the vectors correspond to a vertical strip in the physical domain. The differential operators can then be neatly encoded by using off-diagonal matrices and simple Kronecker tensor products. For simplicity and sufficient smoothness, we have chosen to approximate derivatives to first order by finite central differences. For example, defining $D_{d,n}$ as the $n\times n$ matrix with 1's along the d diagonal and zeros everwhere else, and with h as the mesh size,

$$G_x = \frac{1}{2h}(D_{n,n^2} - D_{-n,n^2}) \quad\text{and}\quad G_y = \frac{1}{2h} I_n \otimes (D_{1,n} - D_{-1,n}), \quad\text{then}\quad [\text{grad}] = \begin{bmatrix}G_x & 0\\ 0 & G_y\end{bmatrix}$$

computes the gradient of a scalar field in the above described vector form. This can be understood by noting that horizontally adjacent entries in the grid differ by precisely n indices in the reshaped vector, hence to compute x derivatives we consider the n and $-n$ diagonals. The tensor product computing y derivatives may be understood by simply applying a a difference matrix for one dimensional data (1 and -1 diagonals) to each column. Similarly, putting

$$F_x = 2h\,(D_{n,n^2} - D_{-n,n^2}) \quad\text{and}\quad F_y = 2h\, I_n \otimes (D_{1,n} - D_{-1,n}), \quad\text{then}$$

$$\text{then}\quad [\text{div}] = \frac{1}{4h^2}\left(I_{2n^2} + D_{n^2,2n^2} + D_{-n^2,2n^2}\right)\begin{bmatrix}F_x & 0\\ 0 & F_y\end{bmatrix}$$

computes the divergence of a vector field in the above vector form. The sum of 3 matrices in the final line serve to sum the x and y components of flux, which approximate the divergence via $\nabla \cdot \mathbf{u}\Delta x^2 \approx \sum \mathbf{u} \cdot \mathbf{n}\Delta x$ where the summation is taken over cell walls. Note that the resulting vector will be redundant in that the first and second halves will be equivalent - this is because the divergence is a scalar quantity and so is duplicated for size compatibility.

The underlying reason for this seemingly unnecessary transformation is that system (8) is a well-studied type of problem with efficient pre-existing algorithms. Once the vector $\mathbf{x} = p^n$ is computed from system (8), we apply G to obtain the desired velocity field corrections.

The major advantage of using the UIC to enforce a density capacity is that we may effectively perform collision avoidance in linear time with the number of agents. This is because computing the density profile is linear in the number of agents, after which the UIC performs a constant amount of work to determine velocity field adjustments. This is shown in figure 6, comparing the computation time with linear and $n \log n$ plots. The UIC computation in fact appears to be sub-linear, but this is due to the costly LCP solve that dominates the calculation involving fewer agents.

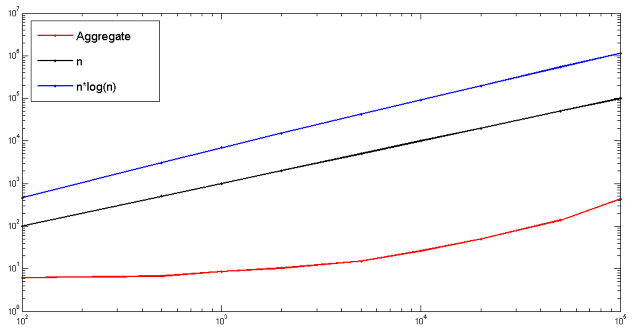
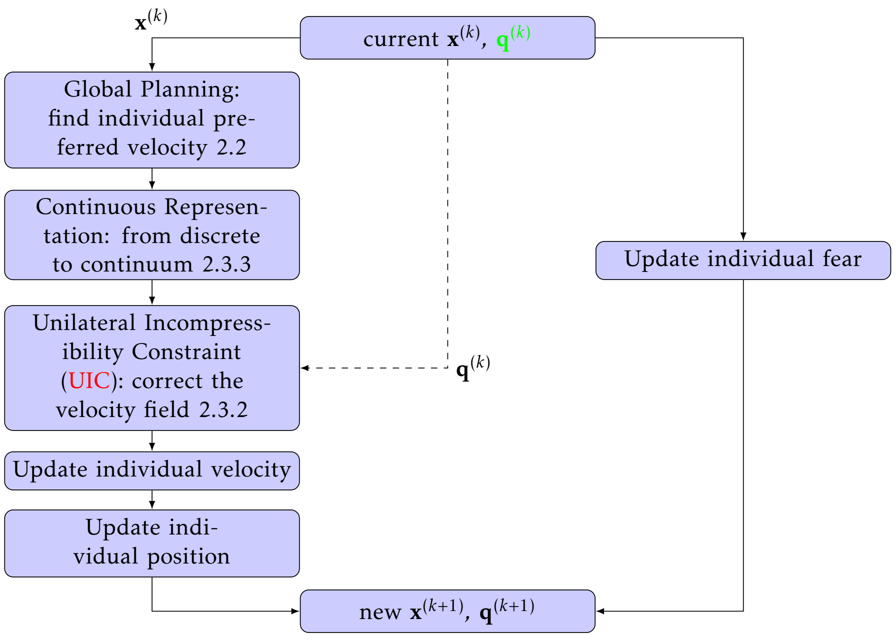


Figure 6: Log-log plot of computation time vs. number of agents for a single timestep.

2.3.5 Algorithm Summary



$\mathbf{x}^{(k)}$
current $\mathbf{x}^{(k)}$, $\mathbf{q}^{(k)}$
Global Planning: find individual preferred velocity 2.2
Continuous Representation: from discrete to continuum 2.3.3
Update individual fear
Unilateral Incompressibility Constraint (UIC): correct the velocity field 2.3.2
$\mathbf{q}^{(k)}$
Update individual velocity
Update individual position
new $\mathbf{x}^{(k+1)}$, $\mathbf{q}^{(k+1)}$

2.3.6 Aggregate Model with Fear

$$\begin{aligned} \dot{\vec{x}}_i &= \vec{v}_i \\ \dot{q}_i &= \gamma(q_i^* - q_i) \end{aligned} \tag{9}$$

where γ and q_i^* is defined as in equation 1, and $\vec{v}_i$ is calculated as following:

1. $\tilde{\mathbf{v}}_i, q_i \xrightarrow{\textit{Continuous Representation}} \tilde{\mathbf{v}}, q$

2. $\tilde{\mathbf{v}} \xrightarrow{UIC} \mathbf{v} = q\frac{\tilde{\mathbf{v}}-\nabla p}{\|\tilde{\mathbf{v}}-\nabla p\|} \xrightarrow{\textit{Interpolation}} \mathbf{v}(\mathbf{x}_i)$

3. $\mathbf{v}_i = \tilde{\mathbf{v}}_i + \frac{\rho(\mathbf{x}_i)}{\rho_{\max}}(\mathbf{v}(\mathbf{x}_i) - \tilde{\mathbf{v}}_i)$

4. $\mathbf{v}_i \xrightarrow{\textit{Velocity Equilibration}} (1-\alpha)\gamma(\mathbf{v}_i^* - \mathbf{v}_i) + \alpha\mathbf{v}_i$, where α determines how much the particle is influenced by its neighbors, and $\mathbf{v}_i = \left(\vec{v}_1, \cdots, \vec{v}_n\right)$.

2.3.7 Numerical Results

I. Pseudo-2D with fear

The simulation is named "Pseudo-2D" because the initial velocities has zero y-component, i.e.

$$\text{Initial condition: } \begin{cases} \vec{v}_{\text{left}}(t_0) = \begin{pmatrix} q_1 \\ 0 \end{pmatrix} \\ \vec{v}_{\text{right}}(t_0) = \begin{pmatrix} -q_2 \\ 0 \end{pmatrix} \end{cases}$$

where $q_1 > q_2 > 0$ indicates the initial fear level of the particles.

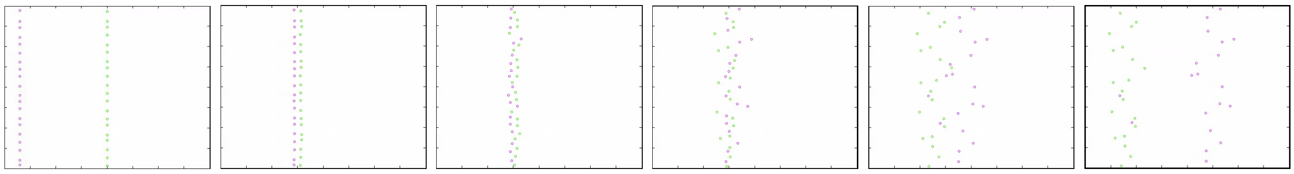


Figure 7: Snapshots from simulation of 40 particles moving in opposite direction. They two groups form a temporal static shock in the middle.

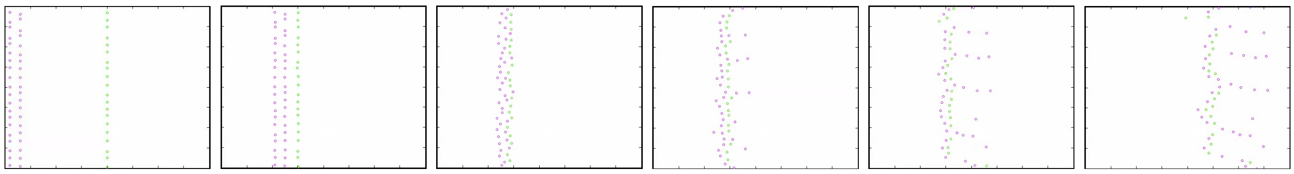


Figure 8: Snapshots from simulation of 60 particles(40 on the left and 20 on the right) moving in opposite direction. They two groups form a traveling shock in the middle.

II. Maze-Escaping Eikonal Solver enable us to simulate emotion contagion of crowds dynamics on more complicated geometry.

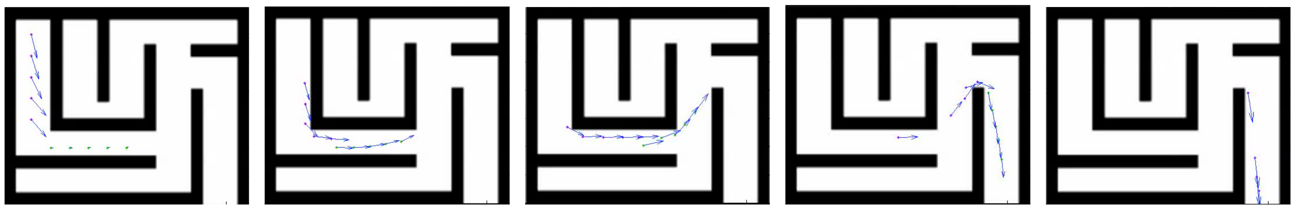


Figure 9: Snapshots from simulation of 10 particles evacuating a maze. Lenghth of the arrows represent the agents' fear level

We also attempted to create a 3D simulation for Maze-Escaping. This simulation tries to replicate the UCLA flood that occurred in the Parking Lot 7, where the x-values between .4 and .6 represents the staircase that connects the basement floor to the ground floor. To create the 3D model, we projected x-values in the plane to the z-values by a piecewise function that forms a sloping level. The particles act as in the 2D aggregate model. The particles that are black have high levels of fear, while the particles that are white have low levels of fear. Eventually all the particles will move at the same speed and have the same fear. Additionally, there is no collision in this model.

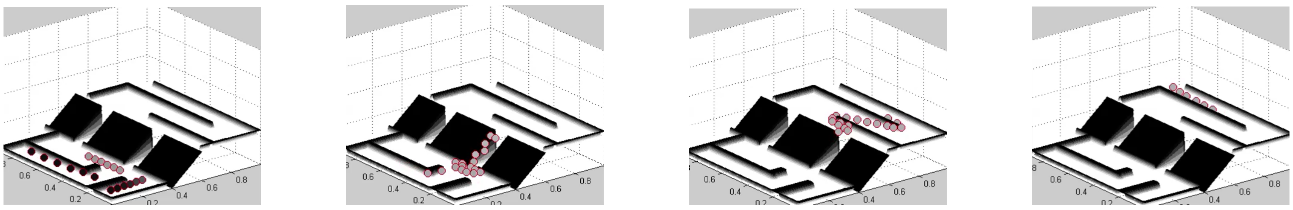


Figure 10: Shapshots of 3D simulation with fear.

2.4 Principal of Least Effort (PLE)

2.4.1 Assumptions of the PLE model

Assumption 1. Each agent has a goal it wants to reach.

Assumption 2. Two agents will each adjust at least half the amount required to avoid collision.

A person will act to avoid someone coming directly at it. That means that each agent is attempting to avoid collision with the other, but will assume that the other agent is also adjusting to avoid collision. Thus an agent can move half the amount required to avoid collision, because the other agent will also be doing the same.

Assumption 3. An agent wants to avoid regions of high density as well as static objects.

Regions of high density will increase the energy expenditure of the agent, as it cannot move along the shortest path at close to optimum speed. Similarly, running into an object stops the agent and keeps them from reaching their goal.

2.4.2 Setting up the PLE Model

The Principle of Least Effort (PLE) states that a person will go towards their goal along the path that they perceive will cost the least effort[11]. Mathematically, this means minimizing an energy function. The energy functions used in our model come from basic gait analysis and biomechanics. A graph of oxygen uptake versus velocity in km/hr is shown in figure 11. Oxygen uptake correlates directly to energy expenditure [15]. The curve for walking energy appears more quadratic, while the energy curve of running is linear [6]. This is reflected in the energy functions chosen for our model.

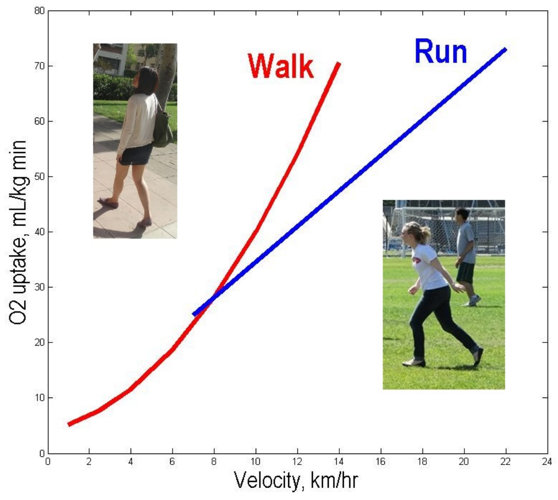


Figure 11: Oxygen uptake vs. agent velocity of walking and running

Using basic gait analysis, the instantaneous power of walking is

$$P = e_s + e_w |\mathbf{v}|^2 \tag{10}$$

where e_s and e_w are empirically derived constants that depend on the person and $\mathbf{v}$ is the instantaneous velocity. In our model, the average values for an adult male were used, 2.23 $\mathrm{J\,kg^{-1}s^{-1}}$ and 1.26 $\mathrm{J\,s\,kg^{-1}m^{-2}}$ respectively.

By integrating over the path, Π, we can find the total energy expended over that path:

$$E_1 = m \int_{\Pi} (2.23 + 1.36|\mathbf{v}|^2) dt. \tag{11}$$

However, people prefer to run when scared, so we used an equation based off oxygen uptake to find the energy of running over path Π. This equation is called the Léger equation [4]. After conversion to the same units as the walking equations, the instantaneous power equation was found to be

$$P = 0.771 + 3.974|\mathbf{v}|, \tag{12}$$

where 0.771 and 3.974 are constants with units $\mathrm{J\,kg^{-1}s^{-1}}$ and $\mathrm{J\,kg^{-1}m^{-1}}$ respectively. Again, we can integrate over the path Π to find the total energy expenditure.

$$E_2 = m \int_{\Pi} 0.771 + 3.974|\mathbf{v}| dt \tag{13}$$

Because this function is linear, the minimum energy expenditure occurs at a person's minimum running speed before converting to walking. The range of the transition between walking and running is at approximately 2.4 m/s.

The fear acts as a sort of switch to indicate preference towards walking or running. The higher the level of fear, the more the person will favor running. To reflect this phenomena, we combine the energy equations and multiply each by a fear factor as follows

$$E_{total} = \left(1 - \frac{q}{q_{max}}\right) E_1 + \left(\frac{q}{q_{max}}\right) E_2 \tag{14}$$

where q is the particle's fear at time t and q_{max} is the maximum fear of an agent.

Because the PLE model is agent based, we chose to update the fear in a way similar to the agent-based ASCRIBE model previously used by our group.

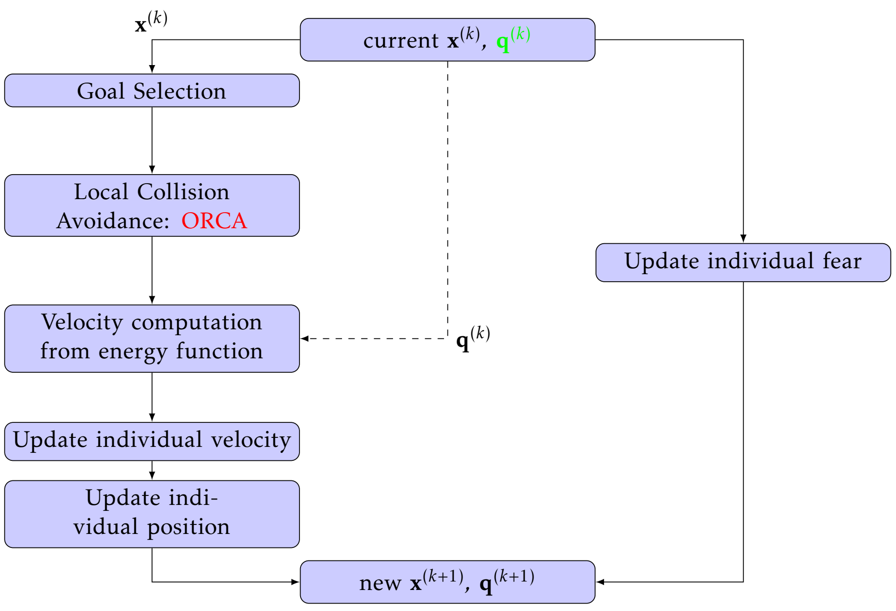
$$\dot{q} = \gamma(q_i^* - q_i) \tag{15}$$

where q_i^* is the weighted average fear of all the surrounding particles:

$$q_i^* = \frac{\sum_{j \in G_i} w_{ij} q_j}{\sum_{j \in G_i} w_{ij}} \tag{16}$$

G_i is the set of points that interact with agent i and the weights, w_{ij}, are determined by their distance from point i. This is a slight simplification of the model used previously in the group that considered more factors in the interaction.

2.4.3 Algorithm Summary



$\mathbf{x}^{(k)}$
current $\mathbf{x}^{(k)}$, $\mathbf{q}^{(k)}$
Goal Selection
Local Collision Avoidance: ORCA
Update individual fear
Velocity computation from energy function
$\mathbf{q}^{(k)}$
Update individual velocity
Update individual position
new $\mathbf{x}^{(k+1)}$, $\mathbf{q}^{(k+1)}$

1. Goal selection updates the goal of the agent. This is fixed in the first step in current simulations. See section 2.4.5
2. The agent positions $\mathbf{x}_i$ and current velocities $\mathbf{v}_i$ are used to determine the ORCA region that will avoid collisions with other agents and obstacles as in section 2.4.4.
3. The new veloicty is determined byminimizing the energy functions inside the ORCA region.
4. Each agent updates its position as $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1}\Delta t$

2.4.4 ORCA [14]

The velocity obstacle of each pair of points is defined as the set of velocities that will result in pairwise collision. The region is represented graphically in figure 12. Mathematically, the velocity obstacle is defined as follows:

$$VO_{A|B}^{\tau} = \{\mathbf{v} | \exists t \in [0, \tau] :: t\mathbf{v} \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\} \tag{17}$$

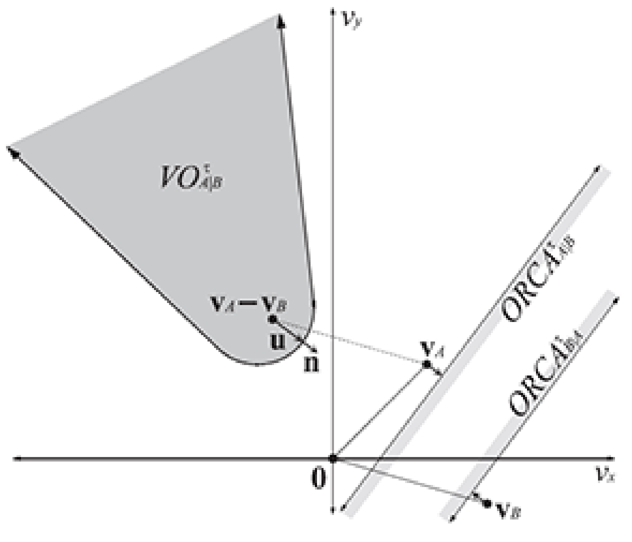


Figure 12: Velocity obstacle and ORCA lines for two particles A and B

If the relative velocity is in the velocity obstacle, i.e. $\mathbf{v}_A - \mathbf{v}_B \in VO^{\tau}_{A|B}$, then agents A and B will collide within τ seconds. The set of collision free velocities is given by

$$CA^{\tau}_{A|B}(V_B) = \{\mathbf{v} | \mathbf{v} \notin VO^{\tau}_{A|B} \oplus V_B\}. \tag{18}$$

If $V_A \subseteq CA^{\tau}_{A|B}(V_A)$ and $V_B \subseteq CA^{\tau}_{B|A}(V_B)$ then V_A and V_B are called reciprocally collision avoiding and if $V_A = CA^{\tau}_{A|B}(V_A)$ and $V_B = CA^{\tau}_{B|A}(V_B)$ then they are called reciprocally maximal.The goal is to choose velocities for A and B such that they are reciprocally collision-avoiding and maximal. There are infinitely many choices however we choose velocities that are closer to the optimal velocity and call the set $ORCA^{\tau}_{A|B}$ for A and $ORCA^{\tau}_{A|B}$ for B. Formally, the $ORCA$ region is defined as:

$$ORCA^{\tau}_{A|B} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}^{opt}_A + \frac{1}{2}\mathbf{u})) \cdot \mathbf{n} \geq 0\} \tag{19}$$

where u is the smallest change of the relative velocity to avoid collision defined as follows:

$$\mathbf{u} = (\arg \min_{\mathbf{v} \in \partial VO^{tau}_{A|B}} ||\mathbf{v} - (\mathbf{v}^{opt}_A - \mathbf{v}^{opt}_B)||) - (\mathbf{v}^{opt}_A - \mathbf{v}^{opt}_B) \tag{20}$$

This change will define a half-plane of permissible new velocities for any pairwise interaction. The intersection of all the half-planes defined by the pairwise interactions of agent i with all other agents will define a convex region. Next each agent picks a velocity from the convex region:

$$\mathbf{v}^{new} = \arg \min_{\mathbf{v} \in ORCA^{\tau}_A} ||\mathbf{v} - \mathbf{v}^{pref}_A||. \tag{21}$$

Then the agents update their position by the adding the velocity to the current position according to step 4 (section 2.4.3).

2.4.5 Numerical Simulation [14][3]

Our simulation was done entirely in Matlab. Our approach was based on the n-particle optimal reciprocal avoidance (ORCA) model developed by Guy et al [11].

The initial positions, velocities, goals, fear levels and radii of each agent are preassigned. At each step in the simulation, we determine whether each pair of points will collide by using the ORCA algorithm(section 2.4.4). If the relative velocity is in the VO region, then we find the vector **u** defined above. Otherwise, we set **u** to be the **0** vector since there is no collision. Once we find **u**, we find the ORCA halfplane for each agent, which has the direction of **u** and passes through $v_A^{opt} + \frac{1}{2}\mathbf{u}$. Intersecting all the ORCA halfplanes results in the convex region which represent the set of velocities that will avoid collisions between the i^{th} agent and all other agents. Then we find the velocity on each segment of the convex region that minimizes the energy. The velocity in this region that minimizes the energy function is the new velocity for the agent i. After this new velocity is calculated for each agent, the position and velocity vectors are updated.

2.4.6 Numerical Results

To test our initial model, we started off with the energy of walking. We can compare our results with the results obtained by the group at UNC Chapel Hill [14] to make sure our code works correctly. On the left in figure 13 shows the paths of two particles switching places. On the right is the results from Guy et al's ORCA model. In their image, the darker color indicates closeness to the goal. In both, each agent moves equally to avoid collision, as expected.

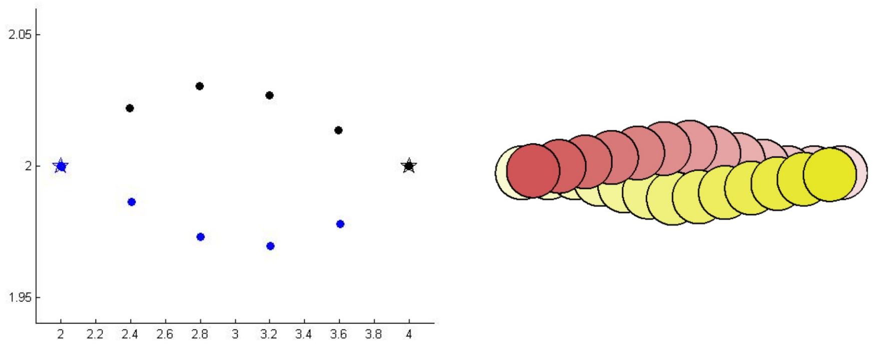


Figure 13: Two agents switching places: our initial results vs previous ORCA work

After this, fear was added to the model. Initially, γ was set to be 1 so that the agents would immediately equilibrate instantly with the fear of the surrounding particles. With an interaction radius of 2, the two agent simulation shown above was repeated, but with the fear update. A comparison of the two simulations with and without fear are shown below in figure 14. The agent on the left starts out with maximum fear (red), and the agent on the right starts out with no fear (yellow). The first particles original velocity is much larger than the original velocity of the particle with no fear. As the agents pass each other, their respective fears spread to the other person, and they both equilibrate to an intermediate fear halfway between their two initial fears (orange).

With the smaller timestep in figure 14, a small amount of oscillation is evident at the beginning in the paths of the particle without fear. Some oscillation is also seen in the Aggregate model when the particles are finding the path without collision. The PLE model does, however, seem to have fewer oscillations when choosing new paths.

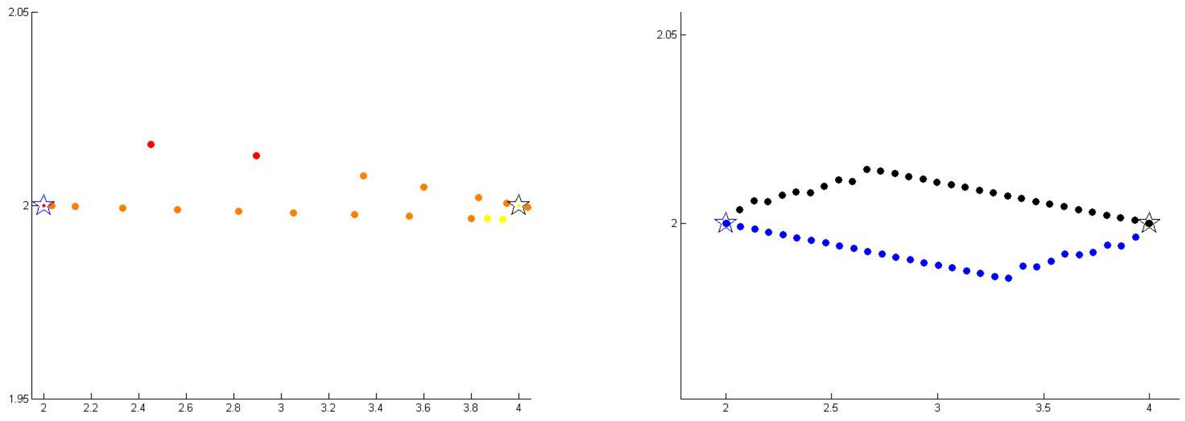


Figure 14: Two agents switching places: Fear with $r_i = 2$ and $\gamma = 1$ vs. no fear

After adding fear, we added a fear source. In our case, the fear source was a predator. An agent's fear was then updated not only by the fear of the surrounding particles, but also by its distance to the predator. By placing the fear sources in different places, we can watch a wave of fear affect other particles. The predator was not included in the local collision avoidance for the agents. In figure 15, blue indicates no fear, green indicates half fear, and dark red indicates maximum fear. For a simple run, the first predator was placed near the bottom corner of two rows of agents.

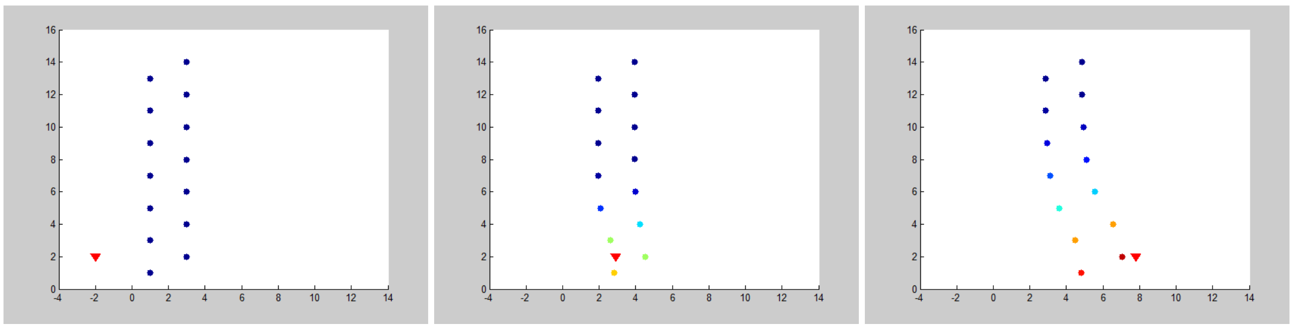


Figure 15: 14 agents in 2 lines with 2 predators,$r_i = 2$, $\gamma = 1$, fixed goals

2.4.7 Final model for PLE

For the PLE model, we recreated a simplified version of the Running of the Bulls. In this model, we made several assumptions about both the bulls and the people.

Assumption 1. The bulls run a straight line to their goal, unaffected by the surrounding people.

This is a fair assumption because when the bulls run together, they are mostly unaffected by the surrounding people, preferring to run together at a near constant speed towards the goal.

Assumption 2. The course the bulls and agents run is straight.

Because much of the course is on straight streets, this is a fair assumption in this model.

Assumption 3. The human agents do not tire from running.

The average speed of the bulls, 6.0 m/s, in this model is slower than the average maximum sprint speed for humans, 9.0 m/s [5]. However, most humans cannot sprint for long periods of time. This would involve another set of equations based on how long a person has been running, but to simplify this simulation, we assumed the agents could sprint for extended periods of time.

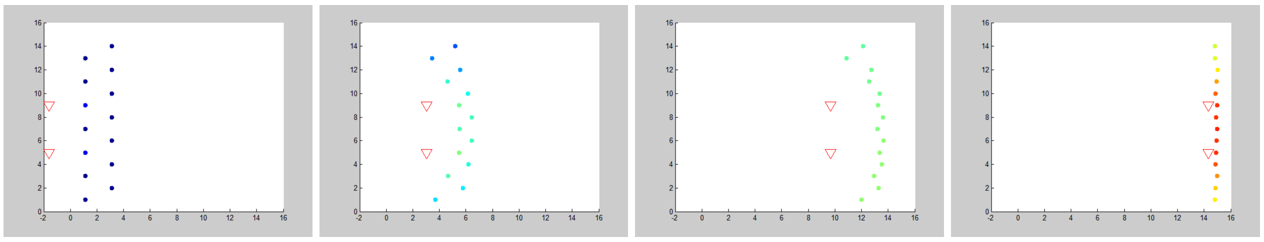


Figure 16: Running of the Bulls simulation: 14 agents in 2 lines with 2 predators,$r_i = 2$, $\gamma = 1$, fixed goals

Our simulation only has two bulls and 14 people, but the sort of wave effect of the fear pushing the agents forwards is evident. This can also be seen in the actual running of the bulls, on the people on the edge of the street do not move as quickly as those directly in front of the bulls, shown in figure 17. This is a good result given the assumptions that greatly simplified our simulation.

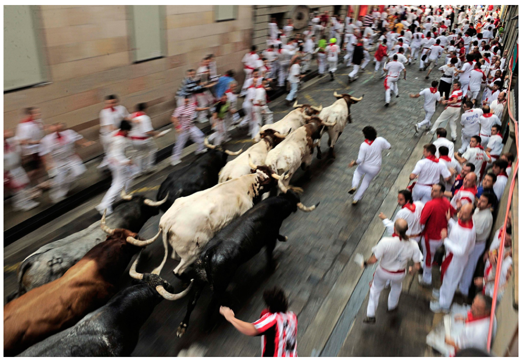


Figure 17: Running of the Bulls: Wave effect

3 Conclusion

Our objective was to model pedestrian crowd flow and add fear to these models based on the work of past UCLA researchers. We first modeled the movement of each particle based on the Eikonal equation, where the particle trajectories are determined by minimizing static environmental discomfort. To extend upon that, we used the strategy in Narain's aggregate model to prevent pedestrian collisions by imposing a maximum allowed density. The advantage of this model is that it performs collision avoidance in linear complexity with the number of agents. Lastly, fear and velocity equilibration were incorporated to model basic emotional communication.

In the PLE model, we first recreated the PLE model developed at UNC, where the velocity is that which minimizes the biomechanical energy expenditure equation for walking. To add the fear, we added a running equation with fear acting as a switch between walking and running. An optimal reciprocal collision avoidance algorithm that uses the change required of each agent to avoid collision was used to update pedestrian paths. The advantage of this model is fewer oscillations when particles are on a collision course, and also its ability to capture emergent phenomena similar to those seen in real situations. Additionally, this model can be combined with other models that take into account psychological and sociological effects.

For future work, we would like to create more sophisticated environments for both of our models and see how the particles will behave accordingly. In the aggregate model, a great deal of unnecessary computation occurs where agent density is very low; this can be avoided by using an adaptive grid size so that high resolution simulation is reserved only where it is needed. In the PLE model, calculation of the ORCA region is costly, and can be minimized by restricting the number of agents and objects considered to neighboring agents.

References

[1] Andrea L Bertozzi, Jesus Rosado, Martin B Short, and Li Wang.
Contagion shocks in one dimension.
Journal of Statistical Physics, pages 1–18, 2014.

[2] Douglas de Jesus, Lingge Li, Daniel Moyer, and Junyuan Lin.
Metaheuristics using agent-based models for swarms and contagion.
2013.

[3] Stephen Guy, Sean Curtis, Ming Lin, and Dinesh Manocha.
Least-effort trajectories lead to emergent crowd behaviors.
American Physical Society, Physical Review E, 85, 2012.

[4] Cameron Hall, Arturo Figueroa, Bo Fernhall, and Jill Kanaley.
Energy expenditure of walking and running: Comparison with prediction equations.
Medicine and Science in Sports and Exercise, 36(12):2128–2134, 2004.

[5] C.M. Wall-Scheffler K.L. Steudel-Numbers.
Optimal running speed and the evolution of hominin hunting strategies.
Journal of Human Evolution, 2009.

[6] W. McArdle, V. Katch, and F. Katch.
Essentials of Exercise Physiology.
2010.

[7] Régis Monneau.
Introduction to the fast marching method.
2010.

[8] Gabriel Peyre.
Dijkstra and fast marching algorithms, 2009.

[9] Gabriel Peyre.
Fast marching code, 2009.

[10] S. Curtis M. Lin R. Narain, A. Golas.
Aggregate dynamics for dense crowd simulation.
ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia, 28(5), 2009.

[11] S. Curtis P. Dubey M. Lin D. Manocha S. Guy, J. Chhugani.
Pledestrians: A least-effort approach to crowd simulation.
Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2010.

[12] J.A. Sethian.
Fast marching methods: A boundary value formulation, 2010.

[13] S Skiena.
Dijkstra's algorithm.
Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley, pages 225–227, 1990.

[14] Jur van den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha.
Reciprocal n-body collision avoidance.
International Symposium on Robot Research, 2009.

[15] Michael Whittle, David Levine, and Jim Richards.
Whittle's Gait Analysis.
2012.