Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Simulation of Human Trail Systems

Jonas Pfefferle & Nicholas Pleschko

Zürich
December 13, 2010

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Gruppenarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen-Hilsmittel verwenden habe, und alle Stellen, die wörtlich oder sinngemäss aus veröffentlichen Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Gruppenarbeit nicht, auch nicht auszugsweise, bereits für andere Prüfung ausgefertigt wurde.

Jonas Pfefferle　　　　Nicholas Pleschko

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Jonas Pfefferle   Nicholas Pleschko

Contents

1 Individual contributions **5**

2 Introduction and Motivations **5**

3 Description of the Model **5**
- 3.1 Active Walker Model 5
- 3.2 Ground Structure - Comfort of Walking 5
- 3.3 Attractiveness of a Trail Segment 6
- 3.4 Walking Direction 6

4 Implementation **7**
- 4.1 A Discrete Model 7
- 4.2 Class Design 8
  - 4.2.1 Plain 8
  - 4.2.2 Pedestrian 9
  - 4.2.3 State Machine 9
- 4.3 Driver 9

5 Simulation Results and Discussion **10**
- 5.1 Experimental setup 10
- 5.2 Results 10
  - 5.2.1 Triangle Setup 10
  - 5.2.2 Corners Setup 11
  - 5.2.3 Left-to-Right Setup 13
- 5.3 Left-to-Right with Obstacle 16

6 Summary and Outlook **18**

References **19**

7 Appendix: MATLAB code **20**

1 Individual contributions

Our code was mostly developed in team work. Jonas described the model and it's discretisation. Nicholas did run the simulations and visualised the results. The rest of the work was done in a cooperative manner.

2 Introduction and Motivations

Most human interactions can be described by mathematical models that invoke self organization. Surprisingly one can develop simple models for complex systems like human trails which are in high agreement with observations made in reality. Such models can for example be used for urban planning or design and building of path systems. Furthermore one may predict optimal paths for recreational park. Another interest of ours was the investigation of the evolution of nonplanar trail systems. We expect the trails to evolve around elevations.

3 Description of the Model

3.1 Active Walker Model

An active walker model[1] describes a two-component system. Where one component is the walker and the other is the landscape and both component are coupled with each other. The walker can alter the landscape while walking. These effects on the landscape influences the walker's movement respectively his walking direction.

3.2 Ground Structure - Comfort of Walking

The ground is represented by a plain. According to [HKM97] one could define the comfort of walking as a function $G(r,t)$ on the ground which represents the ground structure at place r and time t. Furthermore because they use the active walker model to represent the pedestrians, the environmental change of the ground structure at a place r is determined by all the other pedestrians in the plain and the durability of a evolved trail. Where the durability of a trail $T(r)$ can be used to represent the weathering effect $\frac{1}{T(r)}$, i.e. the effect of restoration of the ground structure. Clearly the ground structure can only be restored to some initial condition G_0. Therefore the comfort of walking of a place r decreases by some value determined by the durability of a trail segment. The higher the durability the less the comfort decreases. This can be expressed by $\frac{1}{T(r)}[G_0(r) - G(r,t)]$. Furthermore one now

[1] [KAPL92, *cf. Introduction*]

has to take in account the other pedestrians. As we described in the section before every walker alters the environment it walks over. One could describe those changes as footprints every walker leaves on a place r. Their intensity can be expressed by $I(r)[1 - \frac{G(r,t)}{G_{max}(r)}]$, where $G_{max}(r)$ is the maximum comfort of walking at some place r (e.g. all the vegetation at place r is trampled down). Finally this leads to the environmental change[2] of our ground structure $G(r,t)$:

$$\frac{dG(r,t)}{dt} = \frac{1}{T(r)}[G_0(r) - G(r,t)] + I(r)[1 - \frac{G(r,t)}{G_{max}(r)}] \sum_{\alpha} \delta(r - r_\alpha(t))$$

Where α is the set of all pedestrians walking on the plain.

3.3 Attractiveness of a Trail Segment

Somehow the movement of a pedestrian must be related to the attractiveness of a trail segment. And this attractiveness must be related to the ground structure, as it represents the comfort. Clearly a place r is less attractive if its distance $|r - r_\alpha(t)|$ to the pedestrian's position r_α is large. The larger the distance the less attractive a place gets. Furthermore also the visibility $\sigma(r_\alpha)$ is deciding. If the visibility is low clearly only places in the close neighbourhood are attractive for the pedestrian's movement. According to [HKM97] this leads to the trail potential:

$$V_{tr}(r_\alpha, t) = \int d^2 r e^{\frac{-|r - r_\alpha|}{\sigma(r_\alpha)}} G(r,t)$$

That is the attractiveness of each place r from the perspective of a pedestrian at place r_α.

3.4 Walking Direction

The walking direction of a pedestrian is determined by his destination d_α and the attractiveness of the trail segments. The direction can be expressed by the unit vector $e_\alpha(r_\alpha) = \frac{d_\alpha - r_\alpha}{|d_\alpha - r_\alpha|}$. Furthermore considering only the attractiveness of a trail segment the pedestrian should move into the direction given by the highest slope at the place r_α in V_{tr}, i.e. the gradient $\nabla_{r_\alpha} V_{tr}(r_\alpha, t)$. Finally combining those to aspects this leads to the walking direction:

$$e_\alpha(r_\alpha) = \frac{d_\alpha - r_\alpha + \nabla_{r_\alpha} V_{tr}(r_\alpha, t)}{|d_\alpha - r_\alpha + \nabla_{r_\alpha} V_{tr}(r_\alpha, t)|}$$

[2] [HKM97, *c.f. page 48 - (1)*]

4 Implementation

4.1 A Discrete Model

To implement a simulation we had to discretise the in section 3 described model. Therefore we partitioned the plain into a mesh, i.e. small squares of equal sizes. Furthermore we had to discretise the time such that a time step goes from time t to $t+1$. Finally this leads to the equation of environmental change with direct update of the ground structure:

$$G(r,t+1) = G(r,t) + \tfrac{1}{T(r)}[G_0(r) - G(r,t)] + I(r)[1 - \tfrac{G(r,t)}{G_{max}(r)}] \sum_{\alpha} \delta(r - r_\alpha(t))$$

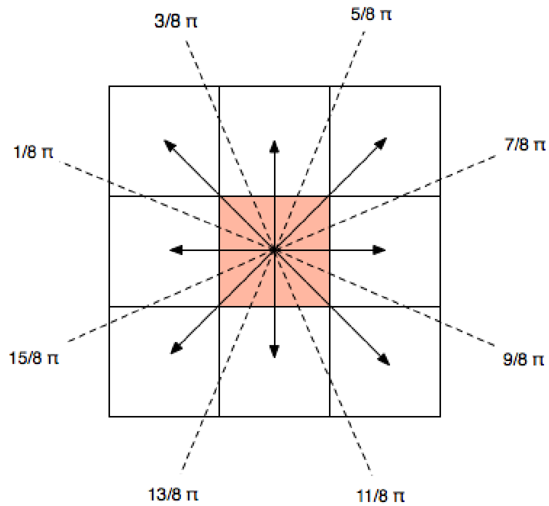
Let Ω be the set of all places r in the plain than the attractiveness at a place r_α is approximated by an arithmetic average over this set:

$$V_{tr}(r_\alpha, t) = \left[\sum_{r \in \Omega} e^{\frac{-|r-r_\alpha|}{\sigma(r_\alpha)}} G(r,t) \right] / |\Omega|$$

Where the direction of the gradient becomes the direction to the maximum value of V_{tr} of all direct neighbors surrounding r_α, i.e. one of the nine squares adjacent to r_α. The movement of the pedestrian is then determined by the direction of the next destination where we can use the described formula of section 3 and the direction given by the maximum value of the surrounding squares. Let $\Lambda \subseteq \Omega$ be the set of surrounding squares:

$$e_\alpha(r_\alpha, t) = \tfrac{d_\alpha - r_\alpha}{|d_\alpha - r_\alpha|} + \tfrac{\arg\max\limits_{r \in \Lambda} V_{tr}(r,t)}{|\arg\max\limits_{r \in \Lambda} V_{tr}(r,t)|}$$

As we have only eight directions the pedestrian can move to, we had to divide the directions into eight equally sized segments of the circle. Where each is $2\pi/8$ wide:



To change the behaviour whether the next destination or the attractiveness for deciding the direction to go to is more important we introduced a new variable ρ. If ρ is larger than 1 the distance gets more important elsewise the attractiveness gets more important.

$$e_\alpha(r_\alpha, t) = \rho \cdot \frac{d_\alpha - r_\alpha}{|d_\alpha - r_\alpha|} + \frac{\arg\max_{r\in\Lambda} V_{tr}(r,t)}{|\arg\max_{r\in\Lambda} V_{tr}(r,t)|}$$

4.2 Class Design

In this section we describe our class design.

4.2.1 Plain

The Plain class represents the ground structure, i.e. a matrix representing the equally sized squares. Where each value at a coordinate (i, j) is related to the comfort of walking as described in previous sections. Furthermore we store a matrix (cf. G_{max}) holding the maximum comfort value at a given coordinate. When instantiating a Plain object one has to specify a initial ground structure representing G_0. Also the intensity of a footprint and the durability at coordinate (i, j) are stored as a matrix.

Finally all to the plain relevant information is stored in this Plain class, the comfort of walking, the maximum comfort of walking, the intensity and the durability.

4.2.2 Pedestrian

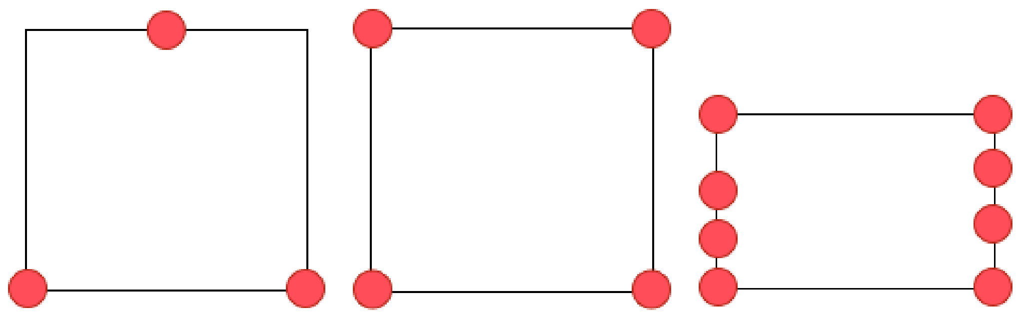
The Pedestrian class represents a pedestrian α at a position r_α. As each pedestrian has to have a destination d_α in the plain this coordinate is also stored. If one would like to check if a pedestrian is at its given destination he easily can call the helper function isAtDestination returning a boolean value.

4.2.3 State Machine

The State Machine class is the heart of our simulation. In this class the transitions are performed. This could be seen as a finite state machine as each entry in the matrix is bounded by the precision of the floating point value. Our state machine simply uses a state at time t given by a plain object and the pedestrians on the plain stored in this class and transforms it to a state of time $t + 1$. This is done in the transition function. When calling transition one can specify a function handle representing a function which generates new pedestrian at some point in the plain. So first the new pedestrians are computed by calling the function handle and adding them to the list of pedestrians in the plain. Then the comfort of walking is computed with this list of pedestrians and the plain's ground structure, initial ground structure and maximum comfort of walking as described in environmental change to the plain. Now the pedestrians can be moved according to the new ground structure by computing the attractiveness V_{tr} and moving the pedestrian as described in the previous subsection.

4.3 Driver

The driver is responsible for actually running the state machine. It instantiates the plain, i.e. the initial ground structure, maximum comfort, intensity and durability. Furthermore it creates the a state machine with the given plain. Then it calls the transition function with a function handle for a function placing new pedestrians. This call is repeated in a loop running until a predefined time limited is exceeded. The driver is also responsible for visualising the output where both the comfort of walking and the attractiveness are shown. Furthermore we included the pedestrians in the visualisation as small white circles.



(a) Triangle setup (b) Corners setup (c) Left-to-right setup

Figure 1: Entry-points and destinations (red) of our simulations

5 Simulation Results and Discussion

5.1 Experimental setup

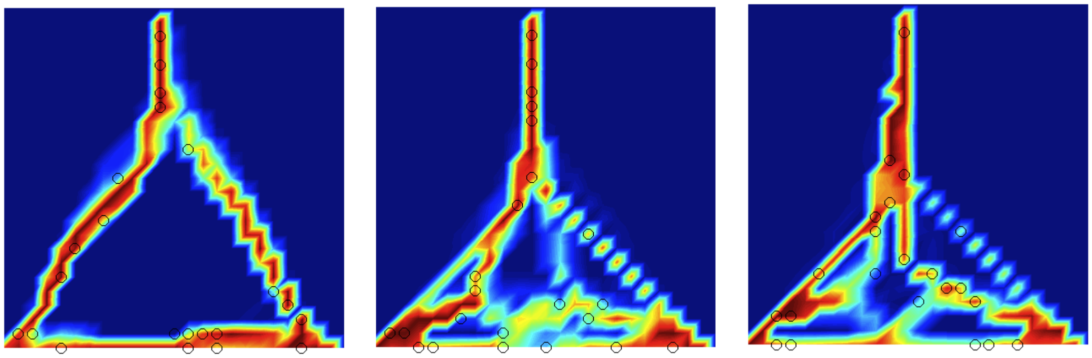
We defined 4 basic setups for visualizing our results. They are characterized by their starting points and destinations of the walkers around the grass area. The first experiment arranges the entry and destination points as a triangle as shown in figure 2a, which corresponds to an fork in a path in the real world. Secondly we simulated a system with four entry points at the corners of the grass area (see figure 2b). This could be the case in a park or the like. The third setup shows the walking of humans which all have the same destination (the grass areas right border) and similar entry points (left border), see figure 2c. Furthermore we decided to generate one new pedestrian "randomly" every time-step.

5.2 Results

5.2.1 Triangle Setup

From this setup we expected simulations which show trails like those at the fork of a path. Increasing the visibility parameter while fixing the other parameters should lead to minimal way system. The following results show the our results on a 25x25 squares grid, increasing the visibility from left to right.

These visualizations of the ground structure show a clear transformation from a direct way system into a minimal way system. Although there are some effects which we didn't expect. For example increasing the visibility to an certain extend yields a more random behavior of the pedestrians.



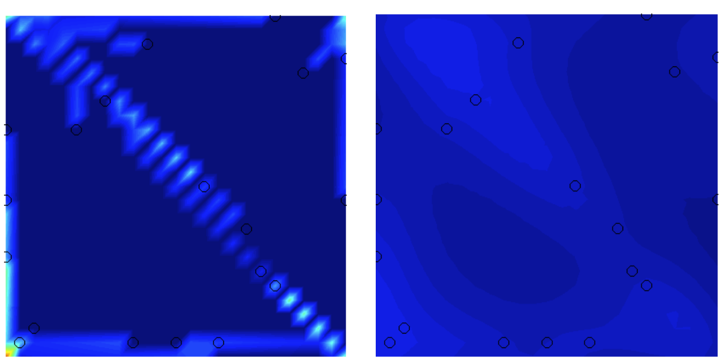
(a) Visibility = 2 (b) Visibility = 4 (c) Visibility = 6

Figure 2: Visualization of ground structure after 140 time steps with varying visibility

5.2.2 Corners Setup

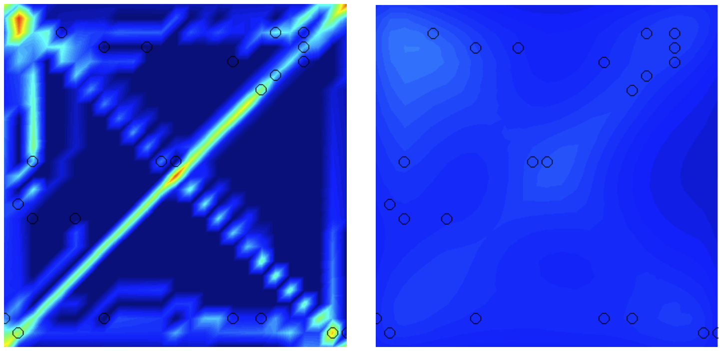
In this simulation we let the pedestrians walk from all the four corners of the grass area to all the other corners. In the early evolution of this system trails should grow in a clear cross and the border lines. But after a while we would like to have more of a minimal way system, such that the outer border lines grow towards the center of the grass area. This would reflect a common behavior on campus grass areas (cf. [HKM97]) and the like.

The following graphics show how the trails evolve in this system. There is an obvious tendency for the borderline paths to move to the center, which is what we would expect. We explain this by the attractiveness maps on the right hand side. They gradually grow into a clear cross. So it makes sense for the walkers to walk towards those highlighted areas.



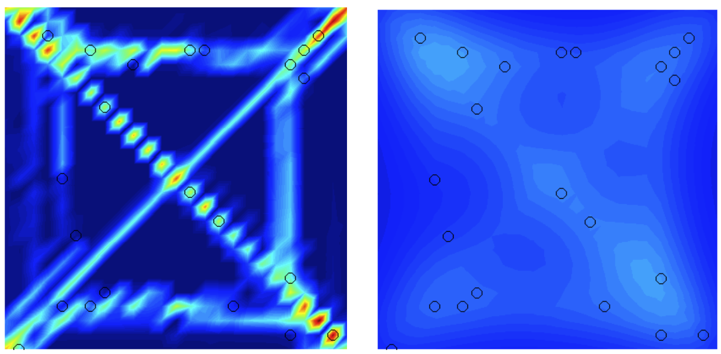
(a) Ground structure (b) Attractiveness

Figure 3: After 20 time steps



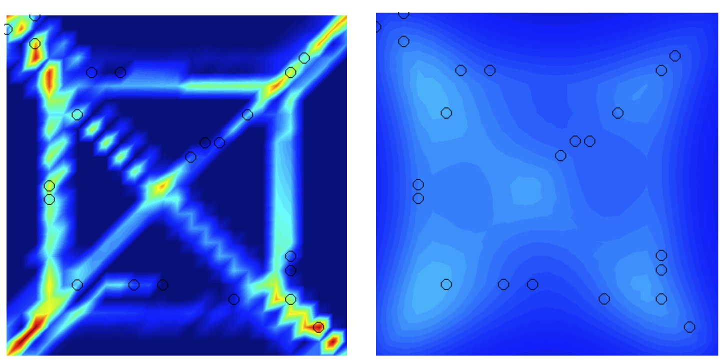
(a) Ground structure (b) Attractiveness

Figure 4: After 40 time steps



(a) Ground structure (b) Attractiveness

Figure 5: After 80 time steps

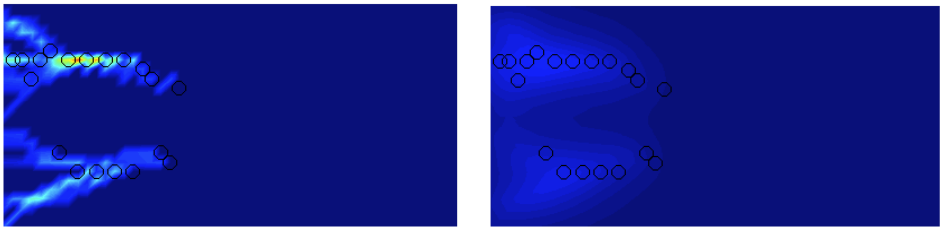


(a) Ground structure (b) Attractiveness

Figure 6: After 80 time steps

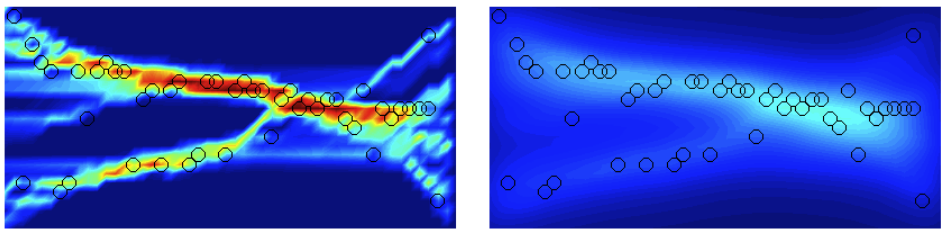
5.2.3 Left-to-Right Setup

When the visibility is low (figure 7a-10b) in this setup there are many trails evolving. But somehow a isle develops were almost all pedestrians walk through. This is very similar to the result shown by [HKM97] figure 4 on page 49. Where they compare this to a real world example of the campus of Brasilia. Note that even such a isle is develop there are always pedestrian walking beside this isle. Therefore their destination gets more important as the attractiveness to this isle. We expected this behavior because of the low visibility the pedestrians are more guided by their destination than the most attractive places in the plain. One can easily see how this behavior changes if we increases the visibility (figure 11a-14b). There a almost unique path is created were all pedestrians walk over. only a few at the border of the plain are developing their own trails. But this trails are likely to get restorated to the initial ground conditions.



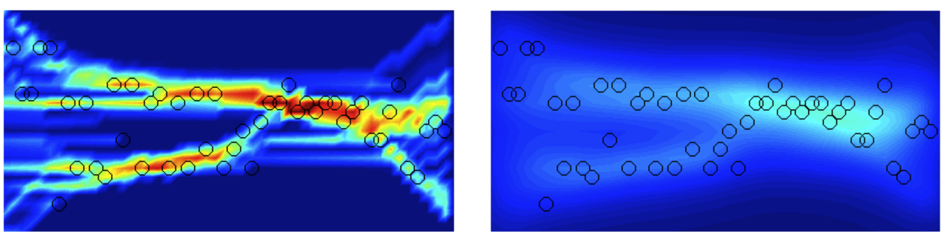
(a) Ground structure (b) Attractiveness

Figure 7: After 20 time steps



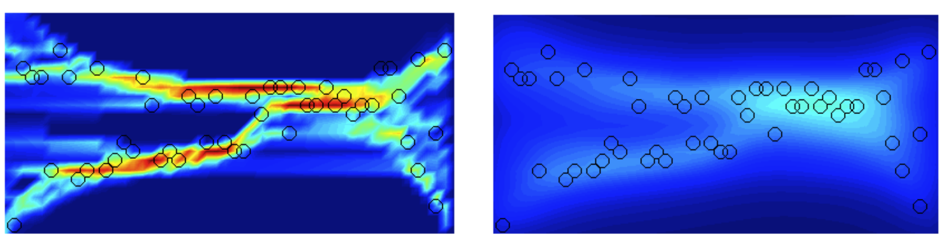
(a) Ground structure (b) Attractiveness

Figure 8: After 80 time steps



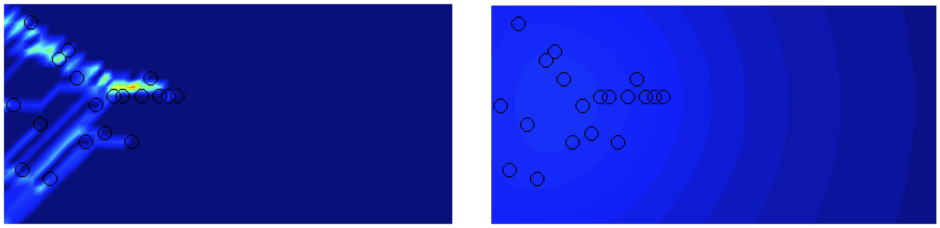
(a) Ground structure (b) Attractiveness

Figure 9: After 140 time steps



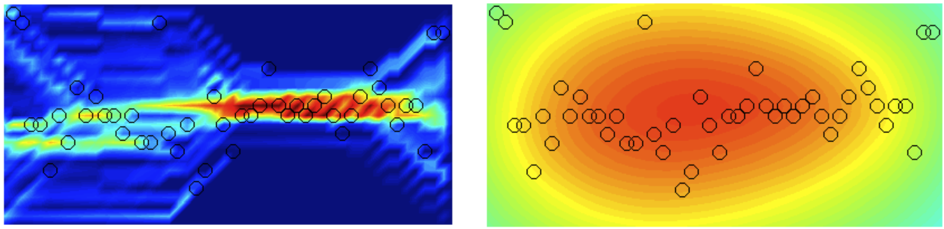
(a) Ground structure (b) Attractiveness

Figure 10: After 200 time steps



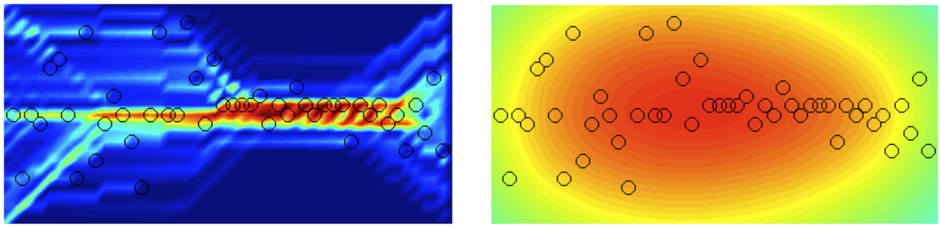
(a) Ground structure (b) Attractiveness

Figure 11: After 20 time steps



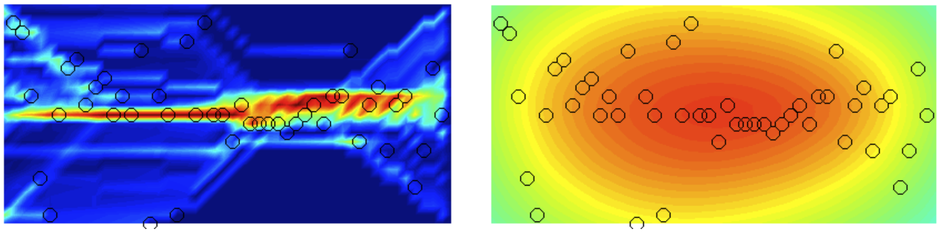
(a) Ground structure (b) Attractiveness

Figure 12: After 80 time steps



(a) Ground structure (b) Attractiveness

Figure 13: After 140 time steps



(a) Ground structure (b) Attractiveness

Figure 14: After 200 time steps

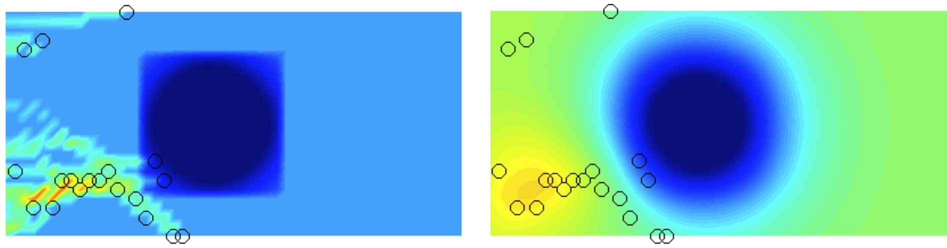
5.3 Left-to-Right with Obstacle

In this setup we tried to modify the given model to simulate trails evolving where obstacles block the pedestrians' way. As we described in previous sections we change the initial ground structure (i.e comfort of walking) such that the slope of a hill is related to a less comfortable ground. We expected the trails to evolve around the obstacle.

As shown in figure 15a - 18b the expectations where met. One can see that in the early evolution there are two main paths above and below the obstacle. In the later process they merge to one more attractive path above. We explain this behavior by the random placement of the pedestrians. Probably in the previous steps there were more pedestrians spawned in the upper half than in the lower half of the plain's border.

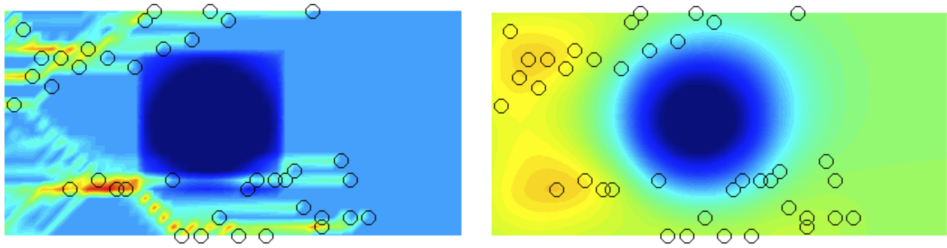
We observed a problem with the intensity at our obstacle. It sometimes happens that a walker walks through the obstacle area which mean that he leaves footprints there. So if there is one pedestrian who chooses the path over the obstacle there will be many to follow. This doesn't represent what we want as the pedestrians trample down our hill.

We therefore decreased the intensity of the footprints in the obstacle area such that pedestrians don't leave any footprints on the obstacle. This models a immutable hill.



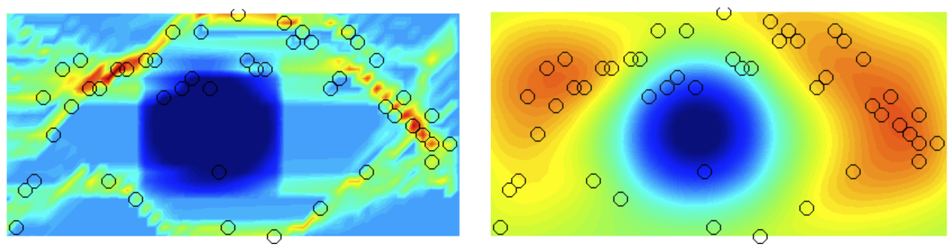
(a) Ground structure (b) Attractiveness

Figure 15: After 20 steps



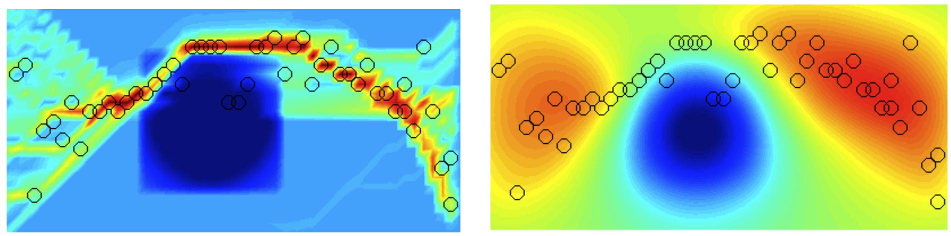
(a) Ground structure (b) Attractiveness

Figure 16: After 40 steps



(a) Ground structure (b) Attractiveness

Figure 17: After 80 steps



(a) Ground structure (b) Attractiveness

Figure 18: After 160 steps

6 Summary and Outlook

The model seems to fit well compared to real world trail systems. But it is very hard to find satisfying values for all the parameters. The paper [HKM97] does not give any hint of how the values should be chosen to gain good results. So we had to somehow guess the values and test if they fulfil our requirements.
Furthermore our idea to model elevation with comfort of walking is as we saw a too simple description. Consider a plain ground where there is some kind of trench from left to right with decreasing height such that on the left side it has equal height to the rest of the plain. Furthermore the pedestrians are walking from left to right. Because we described slope as unattractive this would be the last path taken. But considering the slope is not too high probably a human would choose such paths. Therefore one better changes the behaviour of the attractiveness function V_{tr} such that small decreasing slopes are more attractive and a increasing slope is always unattractive. Note that large decreasing slopes should be considered to be very unattractive as walking downwards on such a path is very exhausting. One might think of trails evolving on precipitous hillsides. There more often than not zig-zag paths develop. It would be interesting if such paths would develop in a modified simulated environment.

References

[HKM97] *Modelling the evolution of human trail systems*, Nature - Volume 388, Dirk Helbing, Joachim Keltsch and Pter Molnr July 1997

[KAPL92] *Active walker models: tracks and landscapes*, Physica A 191, D.R. Kayser, L.K. Aberle ,R.D. Pochy and L. Lam 1992

7 Appendix: MATLAB code

```
classdef Plain < handle
    %PLAIN Saves state of the plain

    properties(SetAccess = public)
        ground;         % The current ground structure
        groundMax;      % The maximum values of the walking comfort
        intensity;      % The footprint intensity
        durability;     % The durability of trails
        visibility;     % The visibility at each point
    end

    properties(SetAccess = private, GetAccess = private)
        initialGround;
    end

    methods
        function obj=Plain(initialGround,aGroundMax,aIntensity,aDurability,aVisibility)
            initSize = size(initialGround);

            if((nnz(initSize == size(aIntensity)) == 2) &&...
                    (nnz(initSize == size(aDurability))==2) &&...
                    (nnz(initSize == size(aGroundMax))==2) &&...
                    (nnz(initSize == size(aVisibility)))==2)

                obj.ground = initialGround;
                obj.groundMax = aGroundMax;
                obj.initialGround = initialGround;
                obj.intensity = aIntensity;
                obj.durability = aDurability;
                obj.visibility = aVisibility;
            else
                error('PLAIN(): initialGround must be same size as intensity and durability');
            end
        end

        function changeEnvironment(obj,pedestrians)
            % Changes the environment according to the positions of the
            % pedestrians
            [n m] = size(obj.ground);
            pedAt = sparse(n,m);

            for i=1:length(pedestrians)
                ped = pedestrians(i);
                pedAt(ped.position(1),ped.position(2)) = ...
```

```
                    pedAt(ped.position(1),ped.position(2)) + 1;
            end

            % Change the environment on each square of the plain
            for i=1:n
                for j=1:m
                    % Change the ground according to the formula
                    obj.ground(i,j) = obj.ground(i,j) + ...
                        1/obj.durability(i,j) * (obj.initialGround(i,j)-...
                        obj.ground(i,j)) + obj.intensity(i,j) * ...
                        (1-(obj.ground(i,j)/obj.groundMax(i,j))) * ...
                        pedAt(i,j);

                    % Check for the boundaries of the ground values
                    if(obj.ground(i,j) > obj.groundMax(i,j))
                        obj.ground(i,j) = obj.groundMax(i,j);
                    elseif(obj.ground(i,j) < obj.initialGround(i,j))
                        obj.ground(i,j) = obj.initialGround(i,j);
                    end
                end
            end

        end

        function val = isPointInPlain(obj,y,x)
            % Returns wheter or not a point (x,y) is in this plain
            val = (y>0 && x>0);
            val = val && y≤size(obj.ground,1) && x≤size(obj.ground,2);

        end
    end

end
```

```
classdef Pedestrian < handle
    %PEDESTRIAN our pedestrian class

     properties(SetAccess = private )
        destination;
     end

    properties
        position;
    end

    methods
        function obj = Pedestrian(dest)
            obj.destination = dest;
```

```
        end

        function set.position(obj,x)
            obj.position = x;
        end

        function val = isAtDestination(obj)
            val = (norm(obj.position - obj.destination)<2);
        end
    end

end
```

```
classdef StateMachine < handle
    %STATEMACHINE Handles the state changes in the simulation
    %   Computes the change of the environment and moves all the pedestrians

    properties(SetAccess = public)
        plain;          % G ... the current plain
        pedestrians;    % Array of pedestrians which are currently walking
        importance;     % How to weight the vector to the destination
    end

    methods
        function obj = StateMachine(aPlain)
            % Constructor: set the plain
            obj.plain = aPlain;
        end

        function [Vtr] = transition(obj, newPedsFun)
            % Does a transition in the state machine according to the plain
            % and the pedestrians.
            % newPeds ... function handle returns pedestrian vector
            [n m] = size(obj.plain.ground);
            Vtr = zeros(n,m);

            % Generate new pedestrians
            newPeds = newPedsFun(size(obj.plain.ground));
            obj.pedestrians = [obj.pedestrians,newPeds];

            % Change the environment according to the pedestrian positions
            obj.plain.changeEnvironment(obj.pedestrians);

            % Compute the attractiveness for each point in the plain
            for i=1:n
                for j=1:m
                    Vtr(i,j) = obj.computeAttractiveness([i;j]);
                end
```

```
            end

            % Delete pedestrians which are at their destination (or close
            % to it)
            deletePeds = [];
            for i=1:length(obj.pedestrians)
                if(obj.pedestrians(i).isAtDestination())
                    deletePeds = [deletePeds,i];
                else
                    obj.movePedestrian(i,Vtr);
                end
            end

            obj.pedestrians(deletePeds) = [];


        end

        function movePedestrian(obj,pedestNum,vtr)
            % Moves a pedestrian according to the attractiveness of the
            % neighbourhood and its destination

            pedest = obj.pedestrians(pedestNum);

            maxvtr = -inf;
            maxcoords = [0;0];

            % compute the maximum value of vtr in the neighbourhood and
            % save the direction to it
            for i = -1:1
                for j = -1:1
                    y = pedest.position(1)+i;
                    x = pedest.position(2)+j;
                    if(obj.plain.isPointInPlain(y,x))
                        if maxvtr < vtr(y,x)
                            maxvtr = vtr(y,x);
                            maxcoords = [i j];
                        end
                    end

                end
            end

            % normalize the gradient vector (but check for zero division)
            if(norm(maxcoords)>0)
                maxcoords = maxcoords / norm(maxcoords);
            end

            % compute the vector to the destination and normalize it
            toDest = pedest.destination - pedest.position;
```

```
            toDest = toDest ./ norm(toDest);

            % add both vectors, but multiply the toDest vector with
            % importance to get better results
            moveDir = obj.importance * toDest + maxcoords;

            % compute the angle of the directional vector
            alpha = atan(moveDir(1)/moveDir(2));

            % Because tan is pi periodic we have to add pi to the angle
            % if x is less than zero
            if moveDir(2) < 0
                alpha = alpha + pi;
            end

            % Define the direction vectors
            up = [-1 0];
            down = [1 0];
            left = [0 -1];
            right = [0 1];

            % Initialize the move vector
            move = [0 0];

            % Shortcut for pi/8
            piEi = pi/8;

            % Check the angle of the resulting vector and choose
            % the moving direction accordingly

            if (alpha < -3*piEi) || (alpha >  11*piEi)
                % move up
                move = up;

            elseif (alpha ≥ -3*piEi) && (alpha < -piEi)
                % move right up
                move = up + right;

            elseif (alpha ≥ -piEi) && (alpha < piEi)
                % move right
                move = right;

            elseif (alpha ≥ piEi) && (alpha < 3*piEi)
                % move down right
                move = down + right;

            elseif (alpha ≥ 3*piEi) && (alpha < 5*piEi)
                % move down
                move = down;

```

```
            elseif (alpha >= 5*piEi) && (alpha < 7*piEi)
                % move down left
                move = down + left;

            elseif (alpha >= 7*piEi) && (alpha < 9*piEi)
                % move left
                move = left;

            elseif (alpha >= 9*piEi) && (alpha < 11*piEi)
                % move up left
                move = up + left;
            end

            % Actually move the pedestrian
            pedest.position = pedest.position + move;
        end

        function [Vtr] = computeAttractiveness(obj,coords)
            % This function computes the sum of all attracivenesses
            % of the whole area from the viewpoint of coords

            Vtr = 0;

            % Get the visibility at point coords
            visibility = obj.plain.visibility(coords(1),coords(2));

            % Get the current ground structure
            G = obj.plain.ground;
            [n m] = size(G);

            % Efficient implementation for the sum
            S = zeros(size(G));
            [A,B]=meshgrid(([1:m]-coords(2)).^2,([1:n]-coords(1)).^2);
            S=-sqrt(A+B);
            S = exp(S/visibility);
            S = S.*G;
            Vtr = sum(sum(S));

            % Average the sum over the number of squares in the plain
            Vtr = Vtr/(m*n);

        end
    end
end
```

```
function smDriver( )
%SMDRIVER Sets up a simulation

```

```matlab
f1 = figure('OuterPosition',[0 0 700 600]);

% Set the grid size
m = 25;
n = 50;


% Set the parameters
gauss = fspecial('gaussian',15,5);

initialGround = zeros(m,n); % Modify this to get objects or slopes into
%initialGround(6:20,16:30)= - (gauss * 5000);
% the simulation.
% Example: Box in the middle
% initialGround(9:12,20:40) = -1000;
dur = 25;           % Durability
inten = 10;         % Intensity
vis = 2;            % Visability
importance = 1.6;   % Weight of the destination vector

groundMax = ones(m,n) * 100;
intensity = ones(m,n) * inten;
%intensity(6:20,16:30) = inten - gauss*1000 ;
durability = ones(m,n) * dur;
visibility = ones(m,n) * vis;


% create new plain with the specified values
myplain = Plain(initialGround,groundMax,intensity,durability,visibility);

% show the plain for input of the entry points
pcolor(myplain.ground);
entryPoints = ginput;

% create a state machine with the specified plain
mysm = StateMachine(myplain);
mysm.importance = importance;

% Do 200 timesteps
for i=1:200

    % print every 20th timestep into a .png file
    if(mod(i,20)==0 && i >0)
        str = sprintf('images/triangle/im_%d_d%d_i%d_v%d_%d.png',...
            importance,dur,inten,vis,i);
        saveas(f1,str);
    end

    % specify the function handle which generates new pedestrians
    %newpedsfun = @(size)entries(i,size,entryPoints);
```

```
    %newpedsfun = @(size)corners(i,size);
    newpedsfun = @(size)leftToRight(i,size);

    % compute a new transition in the state machine
    vtr = mysm.transition(newpedsfun);
    pedestrians = mysm.pedestrians;
    %positions = zeros(m,n);
    fprintf('Number of pedestrians: %d\n',length(pedestrians));

    clf(f1);
    suptitle({[];[];['Grid:' num2str(m) 'x' num2str(n)];['Durability:'...
        num2str(dur) ' Visibility:' num2str(vis) ' '];[ 'Intensity:' ...
        num2str(inten) ' Importance:' num2str(importance) ' '];['After '...
        num2str(i) ' timesteps']});
    subplot(1,2,1);
    title('Ground structure (evolving trails)');
    pcolor(myplain.ground);
    caxis([0 50]);
    shading interp;
    axis equal tight off;

    subplot(1,2,2);
    pcolor(vtr);
    caxis([0 1]);
    shading interp;
    axis equal tight off;

    for j=1:length(pedestrians)
        ped = pedestrians(j);

        subplot(1,2,1);
        title('Ground structure (evolving trails)');

        hold on;
        plot(ped.position(2),ped.position(1),'wo');

        subplot(1,2,2);
        title('Attractiveness');

        hold on;
        plot(ped.position(2),ped.position(1),'wo');
    end


    drawnow;
end
end

function peds = leftToRight(i,pSize)
n = pSize(1);
```

```
m = pSize(2);
    ystart = 1 + floor((n).*rand(1,1));
    ydest = 1 + floor((n).*rand(1,1));
    xstart = 1;
    xdest = m;
    ped = Pedestrian([ydest xdest]);
    ped.position = [ystart xstart];
    peds = [ped];
end

function peds = corners(i,pSize)
corners = [1 1;...
    1 pSize(2);...
    pSize(1) pSize(2);...
    pSize(1) 1];

r = randperm(4);
ped = Pedestrian(corners(r(1),:));
ped.position = corners(r(2),:);
peds = [ped];

end

function peds = entries(i,pSize,ent)
corners = floor([ent(:,2) ent(:,1)]);

r = randperm(size(corners,1));
ped = Pedestrian(corners(r(1),:));
ped.position = corners(r(2),:);
peds = [ped];

end
```