PEDESTRIAN DISTRACTIONS AND PHONE USAGE IN DENSE CROWDS

Ziyad Rabeh

Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
ziyad.rabeh@carleton.ca

1. ABSTRACT

There is an increasing need to use real-time crowd simulation in creating virtual scenes for visual media like films and video games, and in crisis training, architecture and urban planning, and evacuation simulation. Existing particle-based methods assume and aim for collision-free trajectories. That is an ideal -yet not overly realistic- expectation, as near-collisions increase in dense and rushed settings compared to typically sparse pedestrian scenarios. This paper presents a method that evaluates the immediate personal space area surrounding each entity to inform its pathing decisions, and how this personal space is reduced by various distractions such as phone usage. While personal spaces have traditionally been modeled as having fixed radii, they actually often change in response to the surrounding context. For instance, in cases of congestion, entities tend to share more of their personal space than they normally would, simply out of necessity (e.g. leaving a concert or boarding a train). Likewise, entities travelling at higher speeds (e.g. strolling, running) tend to expect a larger area ahead of them to be their personal space [1].

2. INTRODUCTION

Modeling and Simulation has become an essential stage for the development of any complex system in which actual experimentation is too dangerous or costly in terms of money and time. Dense crowd simulation is the process of simulating the movement of large numbers of people in order to analyze and predict their motion within a limited space. This gives the people responsible for the design decisions of such a space the ability to see the utility of their design, and how any design change will affect this utility. The applications of this simulation range from urban planning and congestion control to creating realistic virtual crowd scenes in movies and video games.

Monitoring the same person's movement along the same path for multiple times will give different results, and these differences increase when monitoring different people. These path deviations give us the impression that pedestrian movement is nondeterministic to a certain extent, so human movement simulation will always be an exercise of abstraction.[1]

Dense crowd simulation can be very effective in simulation phenomena that rely on aggregate parameters such as pedestrian

congestions and gatherings for specific events where we can bring a sense of determination through bounded stochasticity.

The crowd simulation model illustrated in this paper is modeled using centroidal particles engine, which is a java based program that uses Processing IDE to visualize the simulation. The features that were added to the crowd's model include the ability to simulate phone usage and the ability to detect collisions between passing pedestrians, and it focuses on local dynamics in an interactive environment. This paper presents improvements made on the already existing crowd's model using centroidal particles.

A study that was done by researchers at the University of Washington's Injury Prevention Center and secretly watched 1102 people crossing the street at "20 high-risk intersections during ... randomly assigned time windows," found that 29.8 percent (nearly one third) of all pedestrians "performed a distracting activity while crossing"[2]. There's a big need to simulate pedestrian distractions for their high occurrence rate and their effect on the personal safety of the distracted pedestrian and the safety of the public around them.

3. BACKGROUND

In centroidal particle crowd simulation, each entity has a global vector that leads to the final desired destination. This means that in the absence of any interference from any other dynamic entity, following the global vector will lead the entity to its final destination in an ideal time. Centroidal particle dynamics (CPD) have rules for each entity to navigate and avoid other dynamic entities in its surrounding, with the least deviation possible from the global path with the least time cost.

3.1 Personal Space

Each entity in the crowd engine is represented as a particle in a 2D plane. The circular area (0.8m radius) surrounding the particle is considered its personal space. The CPD rules are enacted when an entity's personal space is violated. The personal space of an entity changes according to the activity if it's not idle. When a person is distracted on his phone, his awareness area is reduced to right below and in front of him. So to simulate this behaviour, the entity's personal space is reduced.

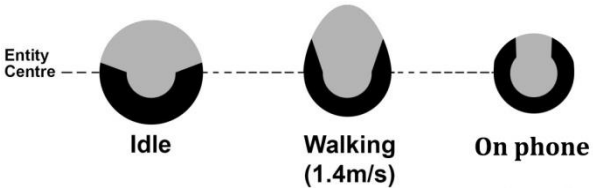

Figure 1: Proposed PS shape: light area affects the entity and its neighbours; dark area only affects neighbours [1]

3.2 Particle Interaction

Each entity's movement is governed by a net force which is a combination of two forces, the global pathing force and the reactive penalty force. The global pathing force is the force leading to the final destination, while the penalty force is applied when another entity or object violates its personal space to avoid any collision.

When the personal space is violated, its shape is reduced and a new centroid is calculated. The net force (f) experienced by an entity is a linear combination of the global pathing force

(g), and the penalty force (p) which falls along the direction of the new centroid.

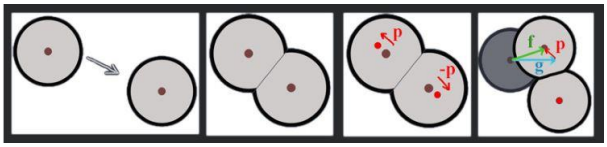

Figure 2: Forces that govern an entity's motion [1]

3.3 Velocity and Comfort Speed

The comfort speed of each entity changes inversely to their local density. The personal space is used to calculate the local crowd density around each entity. This density is used to modify the comfort speed of each entity dynamically. The velocity is used as an upper limit for speed regardless of density.

Experimental data has shown that comfort speed and its relation to local density differ according to cultural differences and the context of the crowd (indoor, outdoor...).

4. SYSTEM MODEL

Crowd simulation importance can't be overstated and it's been already explained in previous sections of this paper. This project implements some additional features and optimizations on the existent centroidal crowd simulation engine.

The effects of phone usage distraction on pedestrians described above are being modeled. The main aspects being implemented are: Personal space truncation, speed reduction, and a collision detection system.

Personal space truncation happens because the pedestrians are looking down at their phones, so they will have less forward and peripheral range of visibility. Different percentages of personal space reductions were tried, but we found that a 40 percent reduction had the most realistic effects on movement. Since a distracted pedestrian's personal space is reduced, they tend to reduce their movement speed to give themself more time to react to obstacles. The comfort speed of pedestrians was reduced by about 50 percent to mimic their behaviour in real life. This reduction is overall and dynamic since an entity's speed also depends on the crowd density and on the max velocity allowed.

As for the collision detection system, different ways of implementation were tried. At first, we implemented a rudimentary method by checking the distance between the centroids of all entities at every frame and seeing if the distance between any two entities is less than two times the radius of the entity's personal space, which means an intrusion of personal space has occurred. But this method was too costly performance wise and it caused a huge dip in the running and display speed of the simulation. The better and more sophisticated way that was later implemented was searching the immediate neighbourhood of each entity, similar to CELL-DEVS, instead of searching the whole canvas. This method required fewer resources in terms of time and processing power.

4.1 Main Engine Code

The centroidal crowd engine base code is organized and divided in to many sections according to functionality. There was no need to modify many sections in the main code except for minor alterations. The bulk of the code modifications went into the Mods and Canvas sections.

Below is an overview of the relevant code sections:

Global variables: contains the declaration and initialization of all the variables that will be visible and accessible across all different functions.

```
int[] specialIDs;     // A list of IDs of the modified entities
PVector[] initialPositions;      // A list of their initial positions
PVector groupAvgPosition;        // Keeps track of the average of special entity positions
PVector mousePosition;           // Keeps track of mouse position
boolean meetingOccured;          // Becomes true if the special entities have met/gathered
PImage obstacleMap;              // Holds an image of the obstacle map
int obstacleChoice;              // Choice of multiple available obstacles
boolean recordTimeToMeeting=false;     // Enable printing of Time-To-Meeting (in seconds)
boolean recordCollisions= true;     // Enable printing of collision count
int WIDTH = 500, HEIGHT = 600;   // Canvas dimensions (1 pixel = 10cm in real life)
// Non-special entities initial distribution
String NONSPECIAL_INIT_DISTRIBUTION = "bidirectional";
// Are they in bidirectional flow, or standing still
boolean NONSPECIAL_BIDIRECTIONAL = true;
// Simulation starts paused?
boolean START_PAUSED = false; boolean PRINT_FRAMERATE;
boolean DISPLAY_OBJECTIVE = false;
boolean PAUSE_SIGNAL;
int MOUSE_DETRACT;
String INITIAL_DISTRIBUTION;
```

State Initialization: This section contains all the special entity ID and size, initial positions for all special entities, and any initialization for any other state variable. This section runs only once at start-up.

```
void modification_Initialization(){
  // Specify the IDs of the special entities
   int size=90;
   specialIDs = new int[size];
   for(int i=0;i<size;i++) { specialIDs[i] = i; }
   initialPositions = new
PVector[specialIDs.length]; // allocate empty array
   for(int i=0; i<specialIDs.length/2; i++){
    initialPositions[i] = new
PVector(random(0.05*width, 0.95*width),
random(0.05*height, height/2));
    // Rejection Sampling

while(canvas.drawingSurface.get((int)initialPositions[i].x, (int)initialPositions[i].y) == COLOR_TO_ID(-5)){
      initialPositions[i] = new
PVector(random(0.05*width, 0.95*width),
random(0.05*height, height/2));
    }
   }
```

Forces: In this section we modify and add all applicable forces to the special entities. The main applicable forces are:

Bidirectional force: This is the global pathing force in a bidirectional setting (hallway, crosswalk). It splits the crowd into two groups each heading to the opposite side of the modeled space.

VoroForce: This is the collision avoidance force that works according to the personal space. When any PS intrusion happens, this force tries to retract and restore the PS to its initial position.

Friction Force: This force helps humanize the entities in a way that they don't appear to be gliding across the field.

```
void modification_Forces(){
  PGraphics ds = canvas.drawingSurface;    // get drawing surface
  // For each special entity:
  for(int i=0; i<specialIDs.length; i++){
    Entity entity = crowd.getEntities()[specialIDs[i]];    // get the special entity
    PVector pos   = entity.getPosition();    // get its current position
    PVector modForce = new PVector(0, 0, 0);    // modification net force (starts empty)
    // Collision avoidance force towards voro centroid = centroidPosition - currentPosition
    PVector voroForce = PVector.sub(entity.getCentroidPosition(), pos).limit(1);
    PVector frictionForce = PVector.mult(entity.getVelocity(),-0.7);
    PVector bipolarForce = new PVector(0, ds.height*(i>floor(specialIDs.length*0.5)?0.15:0.85)-y, 0).limit(3).mult(0.1);
    modForce.add(voroForce.mult(0.7));
    modForce.add(frictionForce.mult(1));
    modForce.add(bipolarForce.mult(1));
    entity.setForce(modForce.mult(0.1)).updatePositionMod();
  }}
```

Visualization: In this section we alter how each state of the special entities is displayed on the screen.

```
void modification_Visualization(){
  // Create some colors we might use (Red, Green, Blue, Alpha) 0-255 each
  color specialHighlight    = color(250,170,20,240);  // special entity highlight color
  color specialHighlightMet = color(20,250,30,240);   // highlight color if they've met
  color mouseHighlight      = color(100,240,10,255);  // mouse position highlight
  color collisionDetected   = color(250, 50,50,200);  // red for collision detection
  // Heighlight the special entities
  for(int i=0; i<specialIDs.length; i++){
    Entity entity = crowd.getEntities()[specialIDs[i]];
    PVector pos = entity.getPosition(); // get entity position
    if (entity.getCollided())
      fill(collisionDetected); // Set the fill color
    else
      fill(specialHighlight); // Set the fill color
    ellipse(pos.x, pos.y, CONE_RADIUS, CONE_RADIUS);  // draw a highlighting circle
  }}
```

Each pixel of the displayed canvas equals 0.1m in real distance. And each frame of the simulation equals 50ms in real time.

State	Regular Pedestrian	Distracted Pedestrian	Collided Pedestrian
Color			

Table 1: Special states colors

4.2 Phone Distraction Code

These are the code modifications done by me to mode the change in Personal Space and speed of pedestrians distracted on their cellphones and the resulting increase in collision potential for all pedestrians. These can be found by searching for [ModDistracted] throughout the project code comments.

First we start by reducing the comfort speed of the entity. The comfort speed is not constant; it is a dynamic speed that's inversely proportional to the local density and limited by the maximum allowed velocity. The 50 percent reduction in comfort speed is achieved by decreasing the limiting velocity. The velocity reduction factor was deduced by experimenting and trying multiple reduction factors. The optimal reduction factor was found to be 20 percent. When the reduction factor is applied, the comfort speed fluctuates between 45 and 60 percent of the original comfort speed.

```
// velocity Change
   PVector v1 = entity.getVelocity(); // get the original velocity
   PVector v2 = v1.mult(0.8); // reduce it by 20%
   entity.setVelocity(v2); // apply it to the distracted entity
```

The second factor in the distraction modeling process is the reduction in the personal space. The personal space in the code is the search area around the entity's centroid where it checks the pixels color and sees if there is a different color among there which means there is an intrusion. If an intrusion is detected, the collision avoidance force is activated to repel the entity in a direction opposite to the occurring intrusion. To reduce the personal space, we reduce the search area for intrusions by 40 percent. After the reduction in personal space, the centroid of the entity will have to be recalculated and shifted to account for the reduction in personal space. Consequently the collision avoidance will be delayed until a bigger intrusion has happened, which gives the entity less time to react to that intrusion.

```
// Find Centroid when on phone (with reduced awareness of PS)
   entity.setCentroidPosition(new PVector(0, 0));    // Reset centroid calc
   int hits = 0, w = ds.width, h = ds.height;
   float s = CONE_RADIUS+1;  // search radius box
   float x = entity.getPosition().x, y= entity.getPosition().y;
   int x1 = (int)clamp(x-s, 0, w), y1 = (int)clamp(y-0.8*s, 0, h); //top-left corner
   int x2 = (int)clamp(x+s, 0, w), y2 = (int)clamp(y+0.8*s, 0, h); //bot-right corner
   for (int xx=x1; xx<=x2; xx++) { // Iterate over local neighbourhood (Personal Space)
     for (int yy=y1; yy<=y2; yy++) {
       if (ds.get(xx, yy)==entity.getColorCode()) {//check for color match
         entity.getCentroidPosition().add(xx, yy, 0);
         hits++;// total number of color matches
       }
     }
   }
   entity.getCentroidPosition().div(hits);
```

The collision avoidance force applied to distracted entities was reduced to 50 percent, compared with 70 percent for non-distracted entities.

```
  //Collision avoidance force
   PVector voroForce =
PVector.sub(entity.getCentroidPosition(),
pos).limit(1);
   //Reduce the force to a realsistic factor
   modForce.add(voroForce.mult(0.5));
   // Apply the net mod force to the entity, and
update its position
entity.setForce(modForce.mult(0.1)).updatePo
sitionMod();
```

4.3 Collision Detection:

A collided state is applied to an entity if a certain degree of intrusion has been recorded in its personal space. The amount of intrusion required to go into a collided state is chosen in a way that disregards minor collisions like shoulder rubs, and only accounts for serious collisions that have an impact. This intrusion factor was chosen by experimenting with multiple factors and monitoring the crowd simulation to see what accounted for a collision. The collisions count increases whenever any entity changes its state from non-collided to collided. The displayed collision count is the total collision count divided by two since each collision involves two entities. The collision tracking starts after 20 frames (1 second) from the start of the simulation to allow for the initial orientation of the entities to occur since they're positioned with a certain degree of randomness.

```
   // Detect Collisions
   int[] hitC = new int[CROWDCOUNT]; //
visible portion of the softicle
   for (int i = CROWDCOUNT; i-->0; ) {
    hitC[i] = 0;
    float s = CONE_RADIUS/2+1;  // search
radius box
    float x = entities[i].getPosition().x,
y=entities[i].getPosition().y;
    int x1 = (int)clamp(x-s, 0, w), y1 =
(int)clamp(y-s, 0, h); //top-left corner
    int x2 = (int)clamp(x+s, 0, w), y2 =
(int)clamp(y+s, 0, h); //bot-right corner
    for (int xx=x1; xx<=x2; xx++) { //
     for (int yy=y1; yy<=y2; yy++) {
      if (ds.get(xx,
yy)==entities[i].getColorCode()) {
       hitC[i]++;
      }
     }
    }
    if(frameCount>20){ // allow 1s for intial
orientation of entities
    if (hitC[i] <= s*s*3.9 &&
!entities[i].getCollided()){
     entities[i].setCollided(true);
     COLLISIONCOUNT++;
    }else if (hitC[i] > s*s*3.9 &&
entities[i].getCollided()){
     entities[i].setCollided(false);}
   }
   }
```

4.4 Crowd Orientation:

The simulation was carried on two crowd flow orientations, Unidirectional and Bidirectional. Unidirectional flow represents a crowd of people heading in the same direction, so all the crowd entities have a global pathing vector with the same direction. Bidirectional flow is the most interesting orientation to study since it's the most common and it applies to most pedestrian traffic scenarios (hallway traffic, street crossing, bridges …). Bidirectional flow is simulated by dividing the crowd into two halves; the first half has a global pathing vector with opposite direction to second half.

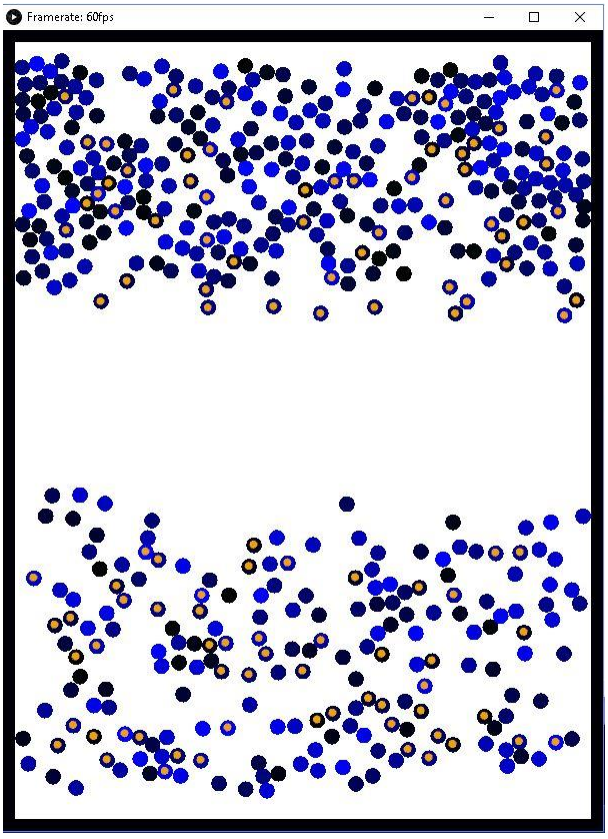

Figure 3.1: Bidirectional crowd flow

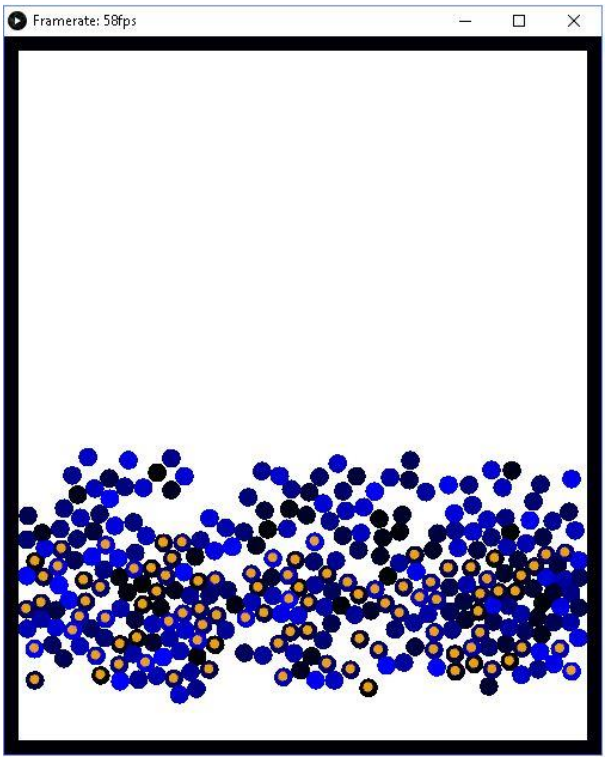

Figure 3.2: Unidirectional crowd flow

5. Simulation Result

To see the full effects of phone usage among pedestrians, simulations were designed around three factors to see their effects on total collision count. The factors are total **crowd size**, **distraction ratio** (percentage of people in the crowd using their phones), and **map width** (simulation canvas width). Simulations were done by increasing each factor while maintain the other two factors constant.

5.1 Bidirectional Flow

The first set of simulations will be performed in a crowd with a bidirectional flow.

Crowd Size:

Crowd size affects the crowd density and the amount of space available for each pedestrian. Multiple simulations were done by varying the total crowd count while maintaining the distraction ratio at 30% and the canvas width at 600px. The total collision count was measured for all simulations.

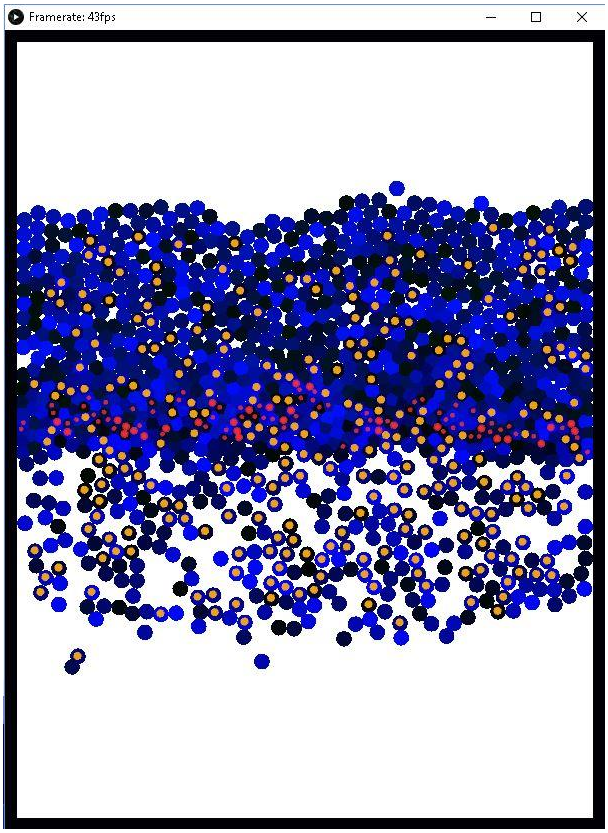

Figure 4: 1200 people crowd count

Table 2 shows the results from running multiple simulations on a crowd of 400 to get the average collision count for this crowd size.

Simulation number	Collision count
1	14
2	12
3	13
4	25
5	13
6	37
7	25
8	45
9	24
10	18
Average	**22.6**

Table 2: Results from 10 simulations for crowd of 400

Table 3 and figure 5 show the results from simulations of different crowd sizes. Each crowd size was simulated 10 times to get an average collision count.

Crowd size	Average collision count
10	0
50	0.2
100	1.6
200	4.4
300	18.5
400	39.5
600	182.4

Table 3: Average collisions count vs. crowd size

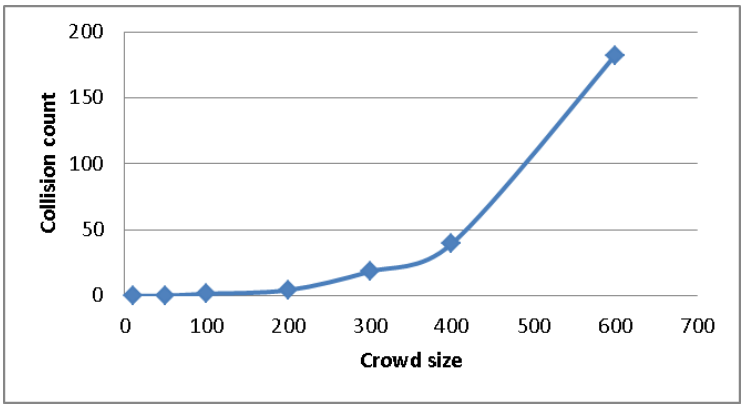

Figure 5: Collisions count v. crowd size graph

We can see in figure 5 that as we increase the crowd size, the collisions count increases exponentially. That's because as we increase the crowd size and keep the field width constant, the crowd density increases and there's less space for the non-distracted entities to manoeuvre the distracted ones.

Distraction Ratio:

To see the real effect of phone usage on collisions between pedestrians, we ran simulations on crowds with varying distraction ratios. For all the distraction ratios tested, we kept the crowd size at 350 and canvas width at 500px.

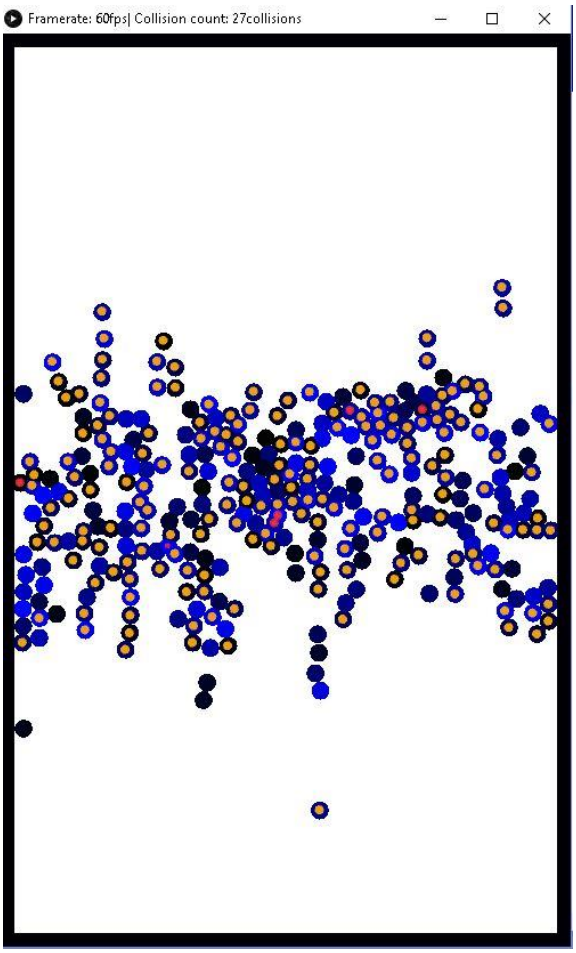

Figure 6: 60% phone usage

Table 4 and figure 7 below show the average collision count for all tested distraction ratios. Each distraction ratio was tested 10 times to give some statistically significant result.

Distraction ratio (%)	Average collisions count
0	1.5
10	4.9
20	8.6
30	11.1
40	17.2
60	32.8
80	62.2
100	89.6

Table 4: Average collisions count vs. distraction ratio

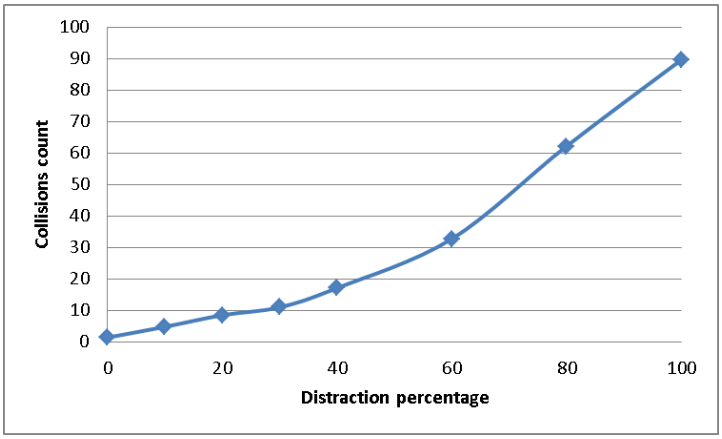

Figure 7: Collisions count vs. crowd distraction percentage

Figure 7 shows that as the percentage of pedestrians using their phones increases, the average collisions count increases linearly. This means that the number of collisions is inversely proportional to the cumulative level of awareness among a crowd.

Passage Width:

Another way to manipulate the crowd density is by varying the map space available, and especially the width of the available passage. Multiple simulations were done by varying the canvas width while maintaining the distraction ratio at 30% and the crowd size at 320 pedestrians.

Each pixel on the canvas equals 0.1m.

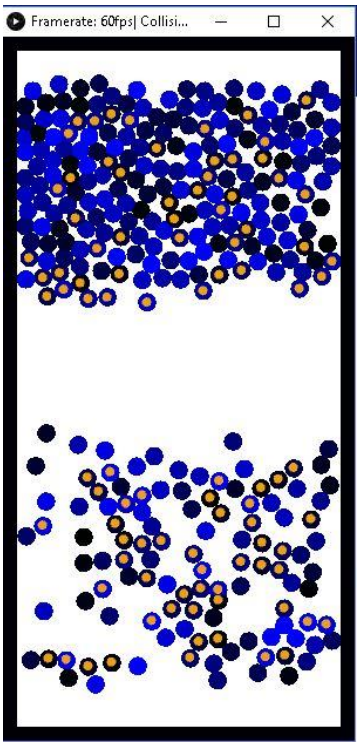

Figure 8: 300px canvas width

Table 5 and Figure 9 below show the average collision count for all tested passage widths. Each width was tested 10 times to give some statistically significant result.

Passage width (m)	Average collisions count
10	378
20	323
30	184
40	77
60	7
80	3
100	1.6

Table 5: Collisions count change as a function of passage width

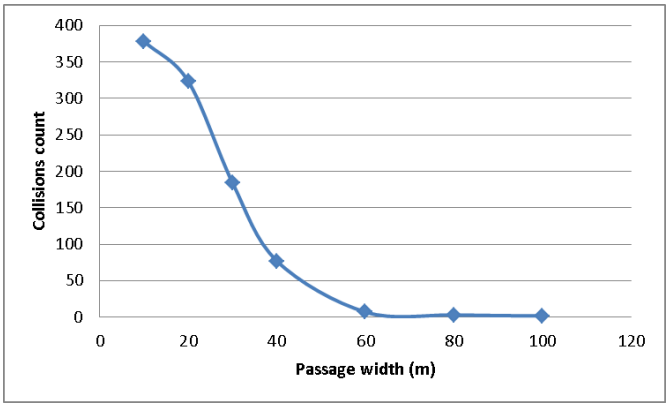

Figure 9: Collisions count vs. passage width

Figure 9 above shows that pedestrian collisions decrease exponentially as the available passage width increases. This means that given enough passage space, non-distracted pedestrians can manoeuvre distracted ones and avoid collisions.

5.2 Unidirectional Flow:

Testing unidirectional flow of crowds where all pedestrians are heading in the same direction we found the following results.

Table 6 shows the collisions results from varying the distraction ratio, and table 7 shows the results from varying the total crowd size.

Crowd size = 350

Distraction ratio (%)	Average collisions count
0	0
10	0
30	0
60	1
100	2

Table 6: Average collisions count vs. distraction ratio in a unidirectional flow

Table 6 shows that increasing the distraction ratio increases the collisions count. The observed increase was to a much lesser extent than in bidirectional flow. This is because all pedestrians are heading in the same direction, so the relative speed between them is extremely small which gives a large amount of time to react and manoeuvre the crowd when needed.

Distraction ratio = 30%

Crowd size	Average collision count
10	0
100	0
400	1
800	189

Table 7: Average collisions count vs. crowd size in a unidirectional flow

Table 7 shows that increasing the crowd increases the collisions count but only in very large crowds. This shows that the space needed for each pedestrian to manoeuvre the crowd is drastically less than in bidirectional crowds.

6. Conclusion

We have shown how centroidal crowd simulation is a powerful tool that can give serious insights on the different factors that affects local crowd dynamics. Simulating distracted pedestrian adds an important feature to the centroidal particle engine.

There were many results that were concluded from this study. The first deduction was that decreasing the number of distracted people in a given crowd will lead to a decrease in accidents. So any effective measure to decrease the use of phone among pedestrians (city ban, fines…) will be effective in decreasing pedestrian accidents.

The second deduction was that increasing the available space for each pedestrian movement either by reducing crowd numbers or by giving more movement space is effective in reducing accidents and reducing the effects of distracted behaviour among pedestrians. So increasing the number of crosswalks and providing wider sidewalks is an effective measure in combating the effects of distracted behaviour.

7. References

[1] Hesham, Omar, and Gabriel Wainer. "Centroidal particles for interactive crowd simulation" Proceedings of the Summer Computer Simulation Conference. Society for Computer Simulation International, 2016.

[2] Thompson LL, Rivara FP, Ayyagari RC. "Impact of social and technological distraction on pedestrian crossing behaviour: an observational study" Injury Prevention Published Online First: 13 December 2012