Simulation algorithm for crowd evacuation in complex situations based on RVO algorithm

Weian Wang1*, Huijie Xue1, Runjia Zhou1, Xudong Chen1, Ziyang Wang2
1 School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
2 School of Traffic and Transportation, Beijing Jiaotong University, Beijing 100044, China
Email: 18301023@bjtu.edu.cn

Abstract. In order to better evacuate the crowd and ensure the safety of the crowd's life and property in emergencies, we designed and developed simulation software for crowds under complex conditions based on the RVO(reciprocal velocity obstacles) algorithm. The system completes crowd simulation, obstacle simulation, evacuation path simulation, crowd collision simulation, supports automatic crowd evacuation and sets special evacuation paths for special crowds. This article focuses on the algorithm part.

1. Introduction
If there is a situation that requires evacuation or emergency evacuation in crowded places, if the evacuation route is not planned or simulated in advance, it is easy to form a large crowd of people and the evacuation efficiency is very low. A large number of crowds may cause a stampede to cause harm to life safety. The RVO algorithm can simulate the automatic obstacle avoidance of the crowd during the evacuation process, and supports the creation of obstacles and the judgment of whether there is an obstacle between two points. However, the automatic obstacle avoidance of the RVO algorithm is a blind obstacle avoidance with a single target point. Poor performance in complex obstacles. Based on the RVO algorithm, this article provides the creation and switching method of the global target point, and completes the visualization operation.

2. The outermost node design
The outermost node is a node that covers the whole world, consisting of an outer frame, a large number of nodes, and connections between nodes. The outer frame is used to set the range of the node. The node is the middle point that the simulated crowd will choose to pass when moving, and the connection means that the two nodes can be switched. After the outermost node is generated, it will be saved in the form of an adjacency matrix. The density of nodes can be set by adjusting parameters. The figure below shows nodes with normal density(Fig.1) and nodes with relatively dense density(Fig.2).
Fig. 1 nodes with normal density

Fig. 2 nodes with relatively dense density

The generation of the outermost node goes through three steps:

2.1. Select the outer frame
Select the boundary points of the outer frame in a counterclockwise order. The range of the outer frame directly determines the range of the node, which should cover the entire simulation area when selected.

2.2. Triangulate the outer frame
Start triangulation from the first point selected by the outer frame, and divide the outer frame into individual triangles.
2.2.1. Triangulation algorithm
The selected points in the outer frame are triangulated on the premise that they are saved in counterclockwise order.
Assuming there are m points in total, let Ni be the i+1th point, and set i=0.
1. If m=2, the algorithm terminates, otherwise proceed to 2.
2. i=i mod m, j=(i+1) mod m, k=(i+2) mod m, proceed to 3.
3. A=Nj-Ni, B=Nk-Nj, proceed to 4.
4. If the Cross Product of A and B is greater than 0, proceed to 5, otherwise proceed to 7.
5. Traverse the remaining m points except Ni, Nj, Nk point T, calculate (T-Ni) × (Nj-T) and (T-Nj) × (Nk-T) and (T-Nk) × (Ni-T) If the results are all less than 0, then proceed to 6, otherwise proceed to 7.
6. Record the triangles Ni, Nj, Nk, remove the points Nj, m--, and proceed to 8.
7. i++, proceed to 1.
8. If i>j, i-- go to 1.

2.3. Generate nodes inside the triangle
Select the midpoint and the center of gravity of the three sides of the triangle as nodes, divide the triangle into 6 smaller triangles(Fig.3), and then divide the small triangles(Fig.4). Each division will record new nodes, update adjacent edges, and delete extra edges.

2.3.1. Node generation steps/Triangle_node algorithm
Use an adjacency matrix to save nodes and reachable neighbors
Initialize the outermost layer of the triangle, record
A B C
B A C
C A B
The first line means that node A can reach node B and node C
Set a value size to limit the number of triangle divisions
Triangle_node(A,B,C)
1. Calculate node D=(A+B+C)/3 according to the coordinates of nodes A, B, and C, and proceed to 2.
2. Add a row starting with node D to the matrix, add neighbor points A, B, and C of D, and update D to neighbor points of A, B, and C at the same time, proceed to 3.
3. Determine in turn whether the midpoints E, F, and G of AB, AC, and BC exist in the matrix, if not, proceed to 4, otherwise proceed to 5.
4. Take node E as an example, F and G are the same
Delete A’s neighbor point B, delete B’s neighbor point A, add E to the matrix, add E’s neighbor points A, C, D, add E to A, B’s neighbor points, and proceed to 6.
5. Take node E as an example, F and G are the same
Add the neighbor point D of E and proceed to 6 (this step also ensures that the different large triangles from the triangulation can be directly connected)
6. Determine in turn whether the area of triangles AED, AGD, BGD, BFD, CFD, CED is greater than the preset value size, if greater than 7.
7. Take triangle AED for example, other triangles are the same
Triangle_node(A,E,D)

3. Obstacle creation and crowd generation

3.1. Creation of obstacles
In the RVO algorithm, the obstacle is composed of a closed quadrilateral formed by four points. Only two points at the beginning and end are selected. Naturally, a minimum value is assigned to the remaining two points to form a thin wall. On the basis of selecting two points to form a wall, consecutively selected points form a room. (The selected points of the generated room are also selected counterclockwise)

The left side of the picture below is an example of a wall (Fig.5), and the right is an example of a room (Fig.6).

![Fig.5 an example of a wall](image1)

![Fig.6 an example of a room](image2)

3.2. Crowd generation
The set of crowds in a room with a known number of crowds is as follows:
1. Get the counterclockwise selected point coordinate sequence of the room.
2. Triangulate the room (see 2.2.1 for the triangulation algorithm)
3. Allocate the number of people in each triangle according to the proportion of each triangle in
the entire room.

(In order to prevent too few people from causing crowds to be concentrated in the big triangle but not in the small triangle, a preset minimum number of people can be set. If the number of people in the room is less than this number, the number of people will be randomized according to this number. Select the number of people in the room from the generated crowd).

4. Get the vertex coordinates A, B, C of triangle ABC because the points on triangle ABC can use the formula \[A+x*(BA)+y*(CA),\ x+y=1,\ x>0,\ y>0\] means, then \[A+x*(BA)+y*(CA),x+y<1,x>0,y>0\] means the point inside the triangle, so only 0 Take two random numbers and add them between 1. If it is less than 1, then the two numbers are x and y respectively; if it is 1, then randomize again; if it is greater than 1, divide the two random numbers by 0.5 and then take it as x, y. At this point, the coordinates of a person are generated, and different random numbers are selected multiple times, and the coordinates of the crowd in a triangle are completed.

5. Substitute the obtained crowd coordinates into the added crowd function of the RVO algorithm.

4. **Shortest path calculation**

Now that we have obtained the adjacency matrix of the record node, obstacles have also been added to the RVO algorithm, and we can begin to assign values to the path. The assignment process is as follows:

1. Read each row in the adjacency matrix in turn.
2. If it is a non-neighbor node, the weight of the path from this node to the node is recorded as infinity.
3. If it is a neighbor node, substitute this node and the neighbor node into the RVO algorithm to determine whether there are obstacles in the two nodes. If there are obstacles, the path weight is recorded as infinity, if there is no obstacle, the distance between the two nodes is used as the weight.

Substitute the assigned adjacency list into the shortest path algorithm to calculate the shortest path and weight for each point to the end and save it.

5. **Shortest path selection**

Take a single person as an example:

1. Traverse all nodes and choose the node closest to you in a straight line with no obstacles as the starting target point.
2. After reaching the target point, select the node with the smallest weight to reach the end point among the neighbor nodes of the target point.
3. After reaching the end, mark the person.
4. When everyone reaches the end, the program ends.

You can dynamically change the weight of nodes by setting some environmental variable factors (such as the degree of congestion), making the simulation more complicated.

**References**

[1] Lu Junjun, Tian Zhenghua. GIS island polygon triangulation algorithm[J]. Surveying and Spatial Geographic Information, 2016.

[2] Liu Shaohua, Tang Jun, Wu Dongsheng, et al. A fast algorithm and application for triangulation of simple polygons [J]. Computer Applications and Software, 2008, 25(003): 79-80.

[3] Berg J P V D, Lin M C, Manocha D. Reciprocal Velocity Obstacles for Real-Time Multi-agent Navigation[C]// 2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA. IEEE, 2008.

[4] Snape J, Berg J P V D, Guy S J, et al. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles[C]// 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2009.

[5] Sainte-Catherine M, Lucet E. A modified Hybrid Reciprocal Velocity Obstacles approach for
multi-robot motion planning without communication[C]// IROS 2020. 2020.