Local Avoidance Techniques for Real-Time Crowd Simulation

Idris Miles
Bournemouth University SDAGE level I

May 8, 2015

Abstract

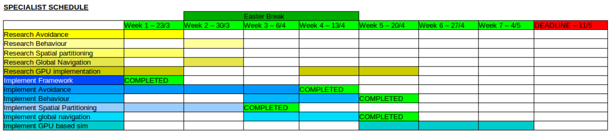
In this paper I demonstrate the use of Reciprocal Velocity Obstacle, RVO, as a viable local avoidance model for real time agent based crowd simulation.

1 Introduction

Crowd simulation is a growing in films and games, where complete CG worlds are brought to life with their own CG inhabitants. Of course it is not feasible to animate 300 pedestrians in the background of a 10 second shot, or thousands of fans in a sports arena. So the use of crowd simulation tools were bought to life.

2 Local Avoidance in Crowd Simulation

Local avoidance is the avoidance of agents within close proximity of each other, it is useful when there are multiple moving objects in the world that the global navigation system may ignore. Global navigation differs from local avoidance because it looks at finding a route from one location to another, while local avoidance just considers its surroundings and desired direction inn order to avoid collisions while steering towards its goal. Artificial Intelligence also differs from local avoidance although the two can be linked together, AI focuses on creating behaviours of the agents which can indirectly incorporate local avoidance but often this isn't enough to satisfy a collision free system, hence local avoidance can be 'layered' on top.



SPECIALIST SCHEDULE

		Easter Break						
	Week 1 – 23/3	Week 2 – 30/3	Week 3 – 6/4	Week 4 – 13/4	Week 5 – 20/4	Week 6 – 27/4	Week 7 – 4/5	DEADLINE – 11/5
Research Avoidance	■							
Research Behaviour		■						
Research Spatial partitioning	■							
Research Global Navigation		■						
Research GPU implementation	■			■	■			
Implement Framework	COMPLETED							
Implement Avoidance	■	■	■	COMPLETED				
Implement Behaviour			■	■	COMPLETED			
Implement Spatial Partitioning	■	■	COMPLETED					
Implement global navigation			■	■	COMPLETED			
Implement GPU based sim					■	■	■	

Figure 1: Project Schedule

3 Initial Research

First of all I made a time schedule in the form of a Gantt Chart, Figure1 that I would follow in order to keep on track, as time management would be crucial in this project. For my initial research I looked into the various methods used for local avoidance and what techniques current systems used. The main methods I came across were Boids Flocking System[8], Social Forces[5], PLE[3],Continuum Crowds[9], and Reciprocal Velocity Obstacle. Reciprocal Velocity Obstacle, RVO, seemed the most relevant and interesting method as it purely focused on object avoidance. RVO is a technique used in various systems including Goalem crowd plugin for Maya and Unreal Engine 4 which shows its relevance in industry.

Figure 2 shows my initial UML diagram of how I felt my code would be structured, to do this I looked into design patterns [1][7].

4 RVO

RVO is an agent based method for collision avoidance and was developed by Jur Van Den Burg et all [10]. It is based upon a similar technique known as Velocity Obstacle (VO), and uses much of the same principle but with some critical tweaks, which

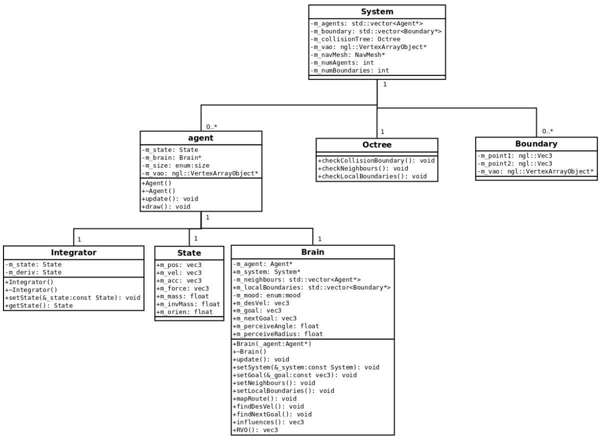


Figure 2: UML class diagram

was initially designed for motion planning of autonomous robots. However these techniques have been extended for use in crowd simulation.

4.1 How RVO Works

VO, which RVO is based from, works by the following:

$$VO^A_B(v_B) = \{v_A | (p_A, v_A v_B) \cap B \oplus A = \varnothing\}$$

This translates to, the velocity obstacle $VO^A_B(vB)$ of agent B to A is the set of velocities V_A for A that will result in a collision with B moving at velocity V_B. The idea is to find a new velocity for the agent A that is outside the VO of B.

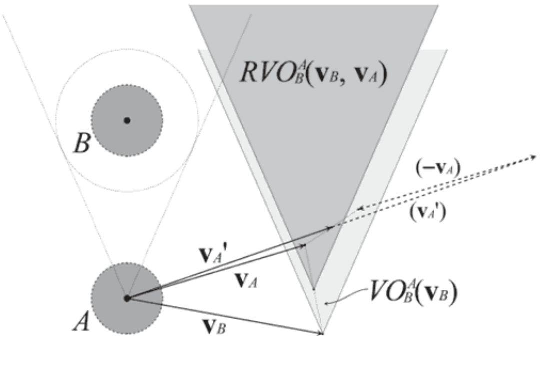


Figure 3: Velocity Obstacle cone

The difference with RVO and VO is instead of finding a velocity outside the VO of B we find the average between the velocity outside and the current velocity [10][4], this can be seen in figure 3. This produces smoother movement of the agent and guarantees oscillation free movement which can be a common problem for local avoidance.

4.2 RVO Implementation

My implementation works by first creating the sample velocities according to the desired velocity. Then iterate through each velocity, within each iteration we also loop through all the neighbours. Inside this second loop we perform RVO. First we create the VO cone, then we check the current test velocity and if it intersects with the VO we find the time to collision, t, and store it in an array for analysis later, however if the velocity does not collide we return a *true* value that is stored in another array. If all agents tested against the current test velocities return *true* we can exit all loops and accept the test velocity, however if this is not the case we move onto the next velocity and repeat. In the case of no velocity being directly acceptable we work out a penalty value for each velocity according to the value of t to determine the best velocity[2][11].

5 Optimizations

The first optimization I implemented was incorporating a hash table to speed up the neighbour search, rather than being of $O(n^2)$ time complexity it became $O(n)$. A further optimization is to have the agents only search half the necessary neighbouring hash cells, when a neighbour is found the two agents add each other to their neighbour list. This means that half the number of neighbour queries have to be completed.

The next optimization was to refine RVO. During dense crowds the agents can become conservative with their movement, this is due to the shape of the VO cone restricting velocities that are temporarily acceptable. However the tip of the cone can be "cut off" thus shrinking the area of the VO[4], this increases the number of available

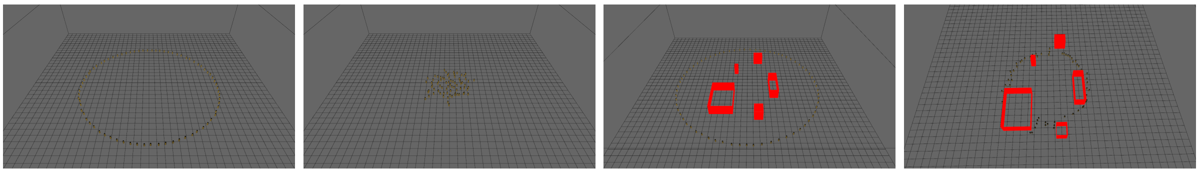


Figure 4: RVO avoidance with 100 agents in a circle

velocities for the agent. In the case of a velocity being selected due to its penalty value, we can multiply the velocity by t, the time to collision, to allow the agent to change its speed. This is not perfect as it means some optimal velocities may be overshadowed however it is a quick method which is necessary for real-time applications.

6 Conclusion and Future Work

I am pleased with the outcome of the project as I feel I was successful in investigating local avoidance techniques focusing on real-time requirements. My implementation is proof off the success of my research as my simulation works with upto 200 agents smoothly. Future work will entail adding global navigation through the use of A* path finding as well as incorporating more complex behaviour such as a refined social forces model. I would also like to further optimize my simulation through GPGPU, certain parts of the program such as the nearest neighbour search on the hash table can be accomplished in parallel[6]. Even the implementation of RVO, with some tweaking, can be performed on the GPU, I started implementing this with the use of OpenGL Compute Shaders however could not finish this in the time frame.

References

[1] Object oriented design patterns. `http://www.oodesign.com/`. accessed: 20th March 2015.

Website designated to various object oriented design patterns

[2] David Cherry. Rvo collision avoidance in unity 3d. 2013.

A students project to incorporate RVO avoidance into a Unity based crowd simulation.

[3] Stephen. J. Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. Pledestrians: A least-effort approach to crowd simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)*, 2010.

Algorithm for simulating large scale crowd simulation based on principles of least effort.

[4] Stephen J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: Highly parallel collision avoidance for multiagent simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2009.

optimizations of existing RVO library, tweaks RVO implementation and describes parellizing implemenation.

[5] Cheng-Te Li and Shou-De Lin. 2010.

Paper on modelling social behaviours in crowd simulation such as the generation of social networks and comminities.

[6] Nvidia. Gpu gems 3. `http://http.developer.nvidia.com/GPUGems3/gpugems3_ch07.html`. accessed 3rd May.

> shader techniques, discusses implementing hash table on GPU

[7] Robert Nystrom. Game programming patters. `http://gameprogrammingpatterns.com/`. accessed: 21th March 2015.

> Website describing many objected oriented design patterns, typically found in games. The website gives examples of how each design pattern is used and why, very useful.

[8] Craig Reynolds. Boids. `http://www.red3d.com/cwr/boids/`. accessed: 27th March 2015.

> flocking systems

[9] Adrien Treuille, Seth Cooper, and Zoran Popovic. Continuum crowds. *SIGGRAPH '06 ACM SIGGRAPH 2006 papers, Pages 1160-1168.*, 2006.

> Real-time crowd model based on continuum dynamics.

[10] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. *IEEE, International Conference on Robotics and Automation (ICRA)*, 2008.

> Velocity Obstacle and many other avoidance models inspired by VO explained and prooved.

[11] Jur van den Berg, Sachin Patil, Jason Sewall, Dinesh Manocha, and Ming Lin. Interactive navigation of multiple agents in crowded environments. *Symposium on Interactive 3D Graphics and Games (I3D)*, 2008. `hhttp://gamma.cs.unc.edu/RVO/NAVIGATE/`.

> Paper abour crowd simulation using RVO for local avoidance, discusses other aspects of crowd simulation as well.

Report word count: x
Annotated bibliography word count: x