Local Avoidance Techniques for Real-Time Crowd Simulation

Idris Miles
Bournemouth University SDAGE level I

May 7, 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.

Figure 1: Project Schedule

1 Introduction

Crowd simulation is a growing niche for films in the animation and VFX world, where complete CG worlds are brought to life with their own CG inhabitants. Of course it would be too much to ask animators to animate the 300 pedestrians in the background, or the thousands of fans in a sports arena. So the use of crowd simulation tools were bought to life. There are many different aspects involved in crowd simulation: behaviour of the characters, also known as agents, global navigation of the agents, local obstacle avoidance, the animation system, so the characters don't glide around, physics if you want characters to react appropriately to various effects, the list goes on.

2 Local Avoidance in Crowd Simulation

Local avoidance is the avoidance of agents within a close proximity of each other, it is useful when there are multiple moving objects in the world that the global navigation system may ignore, due to the fact their position is constantly changing. 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 and aims to avoid collisions while staying on track to its goal. Artificial Intelligence also differs from local avoidance although the two can be linked together, AI focuses on creating behaviours of the agents such as forming groups or separating from other types of agent/objects in the virtual world. AI can indirectly incorporate local avoidance due to the behaviours it may create but often this isn't enough to satisfy a collision free system, hence local avoidance can be 'layered' on top. Boids flocking and social forces are examples of behaviour that can result in indirect local avoidance however they are not enough on their own to guarantee it. Reciprocal Velocity Obstacle, RVO, is not a behaviour but is purely a form of local avoidance. It is the RVO technique that is the main focus of my project.

3 Initial Research

The first call of action was to make a time schedule that I would follow in order to keep on track, as time management would be crucial in this project. Figure 1 shows this in the form of a gant graph. 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[7], Social Forces and Reciprocal Velocity Obstacle. I felt flocking was not suitable as it was better

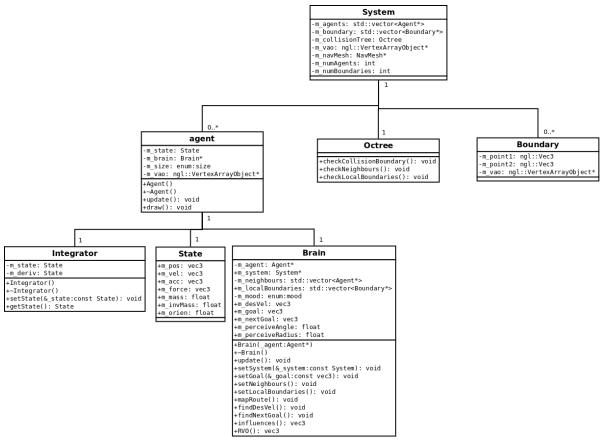

Figure 2: UML class diagram

for replicating animal behaviour such as birds rather than humans. Social forces is an interesting technique that is better for producing human behaviours but not necessarily guaranteed object avoidance. Reciprocal Velocity Obstacle, RVO, is a method not so focused on the behaviour of the agent but rather aims to choose an appropriate velocity that will not result in a collision. RVO is a technique used in various systems including Goalem crowd plugin for Maya and Unreal Engine 4. I decided to focus on the use of RVO as it seemed the most suitable for my project.
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][6].

4 RVO

RVO is an agent based method for collision avoidance and was developed by Jur Van Den Burg et all [8]. 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 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. 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 [8][3], 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 for all agents tested against the current test velocities return *true* we can exit all loops and accept the test velocity, however if this

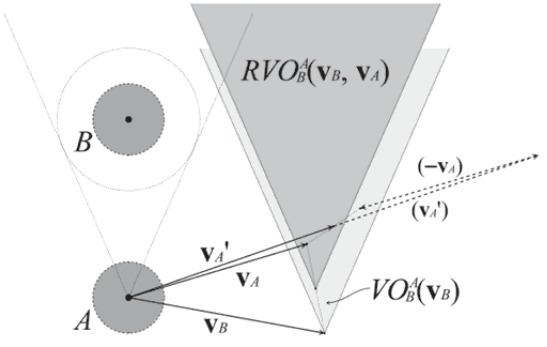

Figure 3: Velocity Obstacle cone generated during RVO

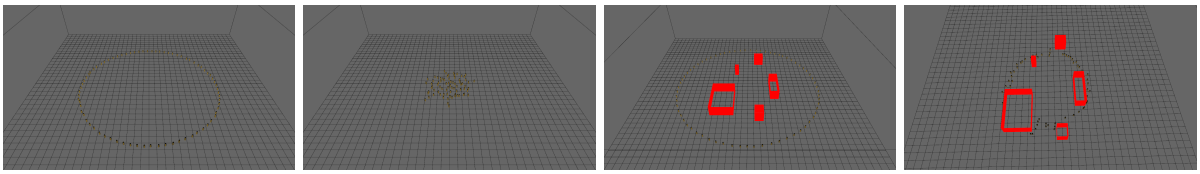

Figure 4: RVO avoidance with 100 agents in a circle

is not the case we move onto the next velocity and repeat. In the case of no velocity being directly acceptable we work out each of the penalty values to determine the best velocity[2][9].

5 Optimizations

Using hash table and Moving heavy computation to a compute shader. [4] The first optimization I implemented was incorporating a hash table. The hash table speeds up the neighbour search considerably, rather than being of $O(n^2)$ time complexity it becomes $O(n)$. A further optimization is to have the agent only search half the necessary neighbouring hash table cells, when it finds a neighbour each agent adds each other to their neighbour list. This means that half the number of neighbour queries have to be completed.
The next optimization was not for speed increase but rather refining RVO. During times when the crowd became dense the agents could become very conservative with their movement, this is due to the shape of the VO cone. The tip of the cone can be cut so that we have 3 edges to check against rather than 2[3], although increasing the number of calculations it also increases the available velocities for the agent. In the case of a velocity being selected due to having the smallest penalty value, it is multiplied by t, the time to collision, this means we can slow down the agent. 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. 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[5].

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, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: Highly parallel collision avoidance for multi-agent simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2009.

optimizations of existing RVO library, tweaks RVO implementation

and describes parellizing implemenation.

[4] Wentong Cai. Nasri Bin Othman., Linbo Luo. and Michael Lees. Spatial indexing in agent-based crowd simulation. *6th International ICST Conference on Simulation Tools and Techniques pages 92-100*, 2013.

spatial partitioning

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

shader techniques, discusses implementing hash table on GPU

[6] 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.

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

flocking systems

[8] 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.

[9] 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