Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2009)
E. Grinspun and J. Hodgins (Editors)

ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation

Stephen. J. Guy[†1], Jatin Chhugani[2], Changkyu Kim[2], Nadathur Satish[2], Ming Lin[1], Dinesh Manocha[1], Pradeep Dubey[2]

[1]University of North Carolina at Chapel Hill
[2]Throughput Computing Lab, Intel Corporation

Abstract

We present a new local collision avoidance algorithm between multiple agents for real-time simulations. Our approach extends the notion of velocity obstacles *from robotics and formulates the conditions for collision free navigation as a quadratic optimization problem. We use a discrete optimization method to efficiently compute the motion of each agent. This resulting algorithm can be parallelized by exploiting data-parallelism and thread-level parallelism. The overall approach, ClearPath, is general and can robustly handle dense scenarios with tens or hundreds of thousands of heterogeneous agents in a few milli-seconds. As compared to prior collision avoidance algorithms, we observe more than an order of magnitude performance improvement.*

1. Introduction

Multi-agent systems are used to model a network of loosely coupled dynamic units, often called agents. Based on the pioneering work on distributed behavior models by Reynolds [Rey87], the study of multi-agent simulation has grown tremendously over the last two decades. Many simulation algorithms have been developed based on simple models and local rules. Besides computer graphics, multi-agent systems are widely used to model the dynamics of crowds, robots and swarms in traffic engineering, virtual environments, control theory, and sensor networks.

In this paper, we address the problem of real-time collision avoidance in multi-agent systems that use distributed behavior models. The motion of each agent is typically governed by some high-level formulation and local interaction rules (e.g. collision avoidance). It is important that agents do not collide with their neighbors. Collision avoidance can quickly become a major computational bottleneck in multi-agent simulations, especially in tightly packed scenarios. This is an important issue in computer games, as the system may only be able to devote less than 5% of processing cycles to collision avoidance and behavioral computations.

Figure 1: ***Building evacuation:*** *1,000 independent agents in different rooms of a building move towards the two exit signs and cause congestion. Our new parallel collision avoidance algorithm, P-ClearPath can efficiently perform local collision avoidance for all agents in such tight packed simulations at 550 FPS on Intel quad-core Xeon (3.14 GHz) processor, and 4,500 FPS on a 32-core Larrabee processor. Our algorithm is an order of magnitude faster than prior velocity-obstacle based algorithms.*

Furthermore, applications such as large-scale urban simulations often need to simulate tens or hundreds of thousands of heterogeneous agents at interactive rates.

One of our goals in studying the computational issues involved in enabling real-time agent-based simulation is

† Contact: sjguy@cs.unc.edu

SCA 2009, New Orleans, LA, August 1–2, 2009.

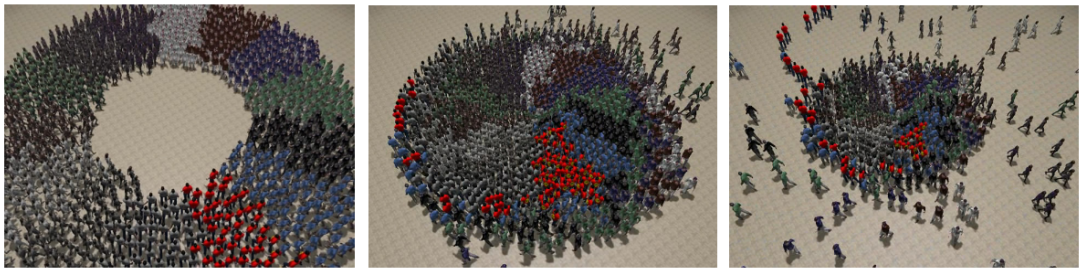

Figure 2: ***Dense Circle Scenario****: 1,000 agents are arranged uniformly around a circle and move towards their antipodal position. This simulation runs at over 1,000 FPS on an Intel 3.14 GHz quad core, and over 8,000 FPS on 32 Larrabee cores.*

to exploit current architectural trends. Recent and future commodity processors are becoming increasingly parallel. Specifically, current GPUs and the upcoming x86-based many-core processor Larrabee consist of tens or hundreds of cores, with each core capable of executing multiple threads and vector instructions to achieve higher parallel-code performance. Therefore, it is important to design collision avoidance and simulation algorithms such that they can exploit substantial amounts of fine-grained parallelism.

Main Results: We present a highly parallel and robust collision avoidance approach, ClearPath, for multi-agent navigation. Our formulation is based on the concept of *velocity obstacles* (VO) that was introduced by Fiorini and Shiller [FS98] in robotics for motion planning among dynamic obstacles. We use an efficient velocity-obstacle based formulation that can be combined with any underlying multi-agent simulation. We show that local collision avoidance computations can be reduced to solving a quadratic optimization problem that minimizes the change in underlying velocity of each agent subject to non-collision constraints (Section 3). We present a polynomial-time algorithm for agents to compute collision-free, 2D motion in a distributed manner. In practice, ClearPath is more than one order of magnitude faster than prior velocity-obstacle based methods.

We show that ClearPath is amenable to data-parallelism and thread-level parallelism on commodity processors and present a parallel extension in Section 4. The resulting parallel extension, P-ClearPath, exploits the structure of our optimization algorithm and architectural capabilities such as gather/scatter and pack/unpack to provide improved data-parallel scalability. We evaluate its performance in various scenarios on different platforms like current multi-core CPUs and the upcoming many-core processor code-named Larrabee. In practice, P-ClearPath demonstrates 8-15X speedup on a conventional quad-core processor over prior VO-based algorithms on similar platforms. When executed on a Larrabee simulator with 32-64 cores, P-ClearPath achieves additional speedup of up to 15X, resulting in up to 100-200X speedup over prior VO-based approaches (Section 5). Overall per frame, for simple game-like scenarios with a few hundred agents, P-ClearPath takes about 2.5 milliseconds on a single Larrabee core, while a complex simulation with few hundreds of thousands of heterogeneous agents takes only 35 milliseconds on the simulated 64-core Larrabee processor. To the best of our knowledge, P-ClearPath is the first scalable approach that performs robust collision avoidance in multi-agent simulations with a few hundred thousand agents at interactive rates.

2. Related Work

Different techniques have been proposed to model behaviors of individual agents, groups and heterogeneous crowds. Excellent surveys have been recently published [TOCD06, PAB08]. These include methods to model local dynamics and generate emergent behaviors [Rey87, Rey99], psychological effects and cognitive models [YT07], cellular automata models and hierarchical approaches. We briefly overview related work in collision detection and avoidance.

2.1. Collision detection and path planning

There is a rich literature on detecting collisions between objects. Many fast algorithms have been proposed for checking whether these objects overlap at a given time instance (discrete collision detection) or over a one dimensional continuous interval (continuous collision detection) [Eri04]. Optimization techniques for local collision detection have been proposed for a pair of objects, including separation or penetration distance computation between convex polytopes [Cam97, Lin93] and local collision detection between convex or non-strictly convex polyhedra with continuous velocities [FT87, KLK*08].

The problem of computing collision-free motion for one or multiple robots among static or dynamic obstacles has been extensively studied in robot motion planning [LaV06]. These include global algorithms based on randomized sampling, local planning techniques based on potential field methods, centralized and decentralized methods for multi-robot coordination, etc. These methods are either too slow for interactive applications or may suffer from local minima problems.

2.2. Collision avoidance

Collision avoidance problems have been been studied in control theory, traffic simulation, robotics and crowd simulation.

Different techniques have been proposed for collision avoidance in group and crowd simulations [MT97, Rey99, SNH01, LD04, HBJW05, FN06, TOCD06, SGA*07]. These are based on local coordination schemes, velocity models, prioritization rules, force-based techniques, or adaptive roadmaps. Other techniques [YMMT08] have used LOD techniques to trade-off fidelity for speed.

The notion of velocity obstacles (VO) was proposed for motion planning in dynamic environments and has been extended to deal with uncertainty in sensor data [FS98, FSL07, KP07]. Recently, Berg et al. [BLM08, BPS*08] extended the VO formulation for reducing collisions between agents. Berg's technique however, relies on extensive sampling for computing collision free velocities which prevents fast implementations. Other extensions [SLS01, PPD07] have also been proposed. However, these techniques provide higher-order path-planning with implementations that are not yet fast enough for very large simulations.

3. Local Collision Avoidance

In this section, we present our collision avoidance algorithm. Our approach is general and can be combined with different crowd and multi-agent simulation techniques.

Assumptions and Notation: We assume that the scene consists of heterogeneous agents with static and dynamic obstacles. The behavior of each agent is governed by some extrinsic and intrinsic parameters and computed in a distributed manner for each agent independently. The overall simulation proceeds in discrete time steps and we update the state of each agent, including its position and velocity during each time step. Given the position and velocities of all the agents at a particular time instant T, and a discrete time interval of ΔT, our goal is to compute a velocity for each agent that results in no collisions during the interval $[T, T+\Delta T]$. We will recompute this velocity for every agent, every time step.

We also assume that the agents are moving on a 2D plane, though our approach can be extended to handle agents moving in 3D space. At any time instance, each agent has the information about the position and velocity of nearby agents. This can be achieved efficiently by storing the state of every agent in a KD-tree. We represent each agent using a circle or convex polygon in the plane. If the actual shape of the agent is non-convex, we use its convex hull. The resulting collision-avoidance algorithm becomes conservative in such cases. In the rest of the paper, we describe the algorithm for circular agents, however we note that our algorithm can be easily extended to other convex shapes. Given an agent A, we use $\mathbf{p}_A$, r_A, and $\mathbf{v}_A$ to denote its position, radius and velocity, respectively. We assume that the underlying simulation algorithm uses intrinsic and extrinsic characteristics of the agent or some high level model to compute desired velocity for each agent ($\mathbf{v}_A^{des}$) during the time step. Let $\mathbf{v}_{max}$ and $\mathbf{a}_{max}$ represent the maximum velocity and acceleration, respectively, of the agent during this timestep. Furthermore, $q^{\perp}$ denotes a line perpendicular to line q.

3.1. Velocity Obstacles

Our approach is built on velocity obstacles [FS98]. We use the notion of Minkowski sum, $A \oplus B$, of two objects A and B and let $-A$ denote the object A reflected in its reference point. Furthermore, let $\lambda(\mathbf{p}, \mathbf{v})$ denote the a ray starting at $\mathbf{p}$ and heading in the direction of $\mathbf{v}$: $\lambda(\mathbf{p}, \mathbf{v}) = \{\mathbf{p} + t\mathbf{v} \,|\, t \geq 0\}$.

Let A be an agent moving in the plane and B be a planar (moving) obstacle on the same plane. The velocity obstacle $VO_B^A(\mathbf{v}_B)$ of obstacle B to agent A is defined as the set consisting of all those velocities $\mathbf{v}_A$ for A that will result in a collision at some moment in time $(t \geq T)$ with obstacle B moving at velocity $\mathbf{v}_B$. This can be expressed as:

$$VO_B^A(\mathbf{v}_B) = \mathbf{v}_A \,|\, \lambda(\mathbf{p}_A, \mathbf{v}_A - \mathbf{v}_B) \cap B \oplus -A \neq \emptyset$$

This region has the geometric shape of a cone. Let $\phi(\mathbf{v}, \mathbf{p}, \mu)$ denote the distance of point $\mathbf{v}$ from $\mathbf{p}$ along μ:

$$\phi(\mathbf{v}, \mathbf{p}, \mu) = \{(\mathbf{v} - \mathbf{p}) \cdot \mu\}$$

Henceforth, the region inside the cone is represented as

$$VO_B^A(\mathbf{v}) = (\phi(\mathbf{v}, \mathbf{v}_B, \mathbf{p}_{ABleft}^{\perp}) \geq 0) \wedge (\phi(\mathbf{v}, \mathbf{v}_B, \mathbf{p}_{ABright}^{\perp}) \geq 0),$$

where $\mathbf{p}_{ABleft}^{\perp}$ and $\mathbf{p}_{ABright}^{\perp}$ are the inwards directed rays perpendicular to the *left* and *right* edges of the cone, respectively. The VO is a cone with its apex at $\mathbf{v}_B$ (Fig. 3(a)).

Recently, Van den Berg et al. [BLM08, BPS*08] presented an extension called RVO. The resulting velocity computation algorithm guarantees oscillation free behavior for two agents. An RVO is formulated by moving the apex of the VO cone from $\mathbf{v}_B$ to $\frac{(\mathbf{v}_A + \mathbf{v}_B)}{2}$. If A has N nearby agents, we obtain N cones, and each agent needs to ensure that its desired velocity for the next frame, $\mathbf{v}_A^{des}$, is *outside* all the N velocity obstacle cones to avoid collisions. RVO algorithm performs random sampling of the 2D space in the vicinity of $\mathbf{v}_A^{des}$, and heuristically attempts to find a solution that satisfies the constraints. However, this may not find a collision-free velocity even if there is a feasible solution. Since RVO uses infinite cones, the extent of the feasible region decreases as N increases. In practice, RVO formulation can become overly conservative for tightly packed scenarios.

3.2. Optimization Formulation for Collision Avoidance

In this section, we pose the local collision avoidance problem for N agents as a combinatorial optimization problem. We extend the VO formulation by imposing additional constraints that can guarantee collision avoidance for each agent during the discrete interval. We take into account the discrete time interval and define a truncated cone (FVO) to represent the collision free region during the time interval corresponding to ΔT. This truncation is a similar idea to other work in robotics [FS98], however we extend the technique from one agent moving through unresponsive dynamic obstacles, to handle several responsive agents moving around each other. The original VO or RVO is defined using only two constraints (left and right), as shown in Fig. 3(a). The FVO is

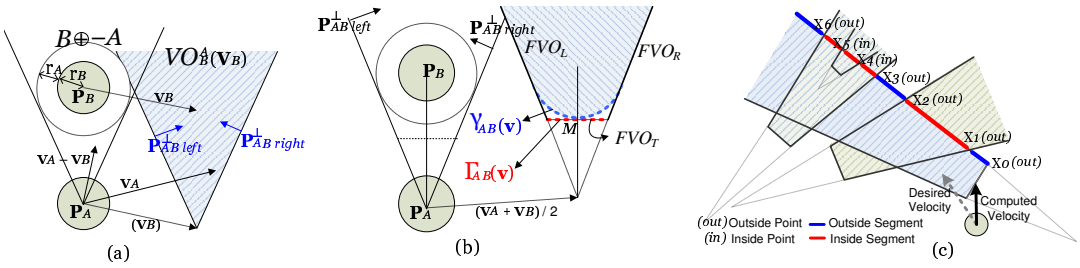

Figure 3: *(a) The Velocity Obstacle $VO^A_B(\mathbf{v}_B)$ of a disc-shaped obstacle B to a disc-shaped agent A. This VO represents the velocities that could result in a collision of A with B ($\mathcal{PCR}$). (b) $FVO^A_B(\mathbf{v}_B)$ of a disc-shaped obstacle B to a disc-shaped agent A. This formulation takes into account the time interval for local collision avoidance. (c) Classifying FVO boundary subsegments as Inside/Outside of the remaining FVO regions. These classifications are used to compute a non-colliding velocity for each agent.*

defined using a total of four constraints. The *two* boundary cone constraints of the FVO are same as that of RVO:

$$\mathrm{FVO}^A_{LB}(\mathbf{v}) = \phi(\mathbf{v}, (\mathbf{v}_A+\mathbf{v}_B)/2, \mathbf{p}^{\perp}_{ABleft}) \geq 0$$
$$\mathrm{FVO}^A_{RB}(\mathbf{v}) = \phi(\mathbf{v}, (\mathbf{v}_A+\mathbf{v}_B)/2, \mathbf{p}^{\perp}_{ABright}) \geq 0$$

Additionally, we impose *two* more types of constraints:

Type-I Constraint - Finite time interval: We only guarantee collision avoidance for the duration ΔT. We compute a finite subset of the RVO cone that corresponds to the forbidden velocities that could lead to collisions in ΔT. The truncated cone (expressed as shaded region in Fig. 3(b)) is bounded by a curve $\gamma_{AB}(\mathbf{v})$ (derivation is given in the appendix). Due to efficiency reasons, we replace $\gamma_{AB}(\mathbf{v})$ with a conservative linear approximation $\Gamma_{AB}(\mathbf{v})$. In practice, this increases the area of the truncated cone by a small amount. This additional constraint is represented as: $\mathrm{FVO}^A_{TB}(\mathbf{v}) =$

$$\Gamma_{AB}(\mathbf{v}) = \lambda\left(M - \widehat{\mathbf{p}^{\perp}_{AB}}\eta, \mathbf{p}^{\perp}_{AB}\right), \text{where}$$
$$\eta = \tan\left(\sin^{-1}\frac{r_A+r_B}{|\mathbf{p}_{AB}|}\right)(|\mathbf{p}_{AB}| - (r_A+r_B)), \text{and}$$
$$M = (|\mathbf{p}_{AB}| - (r_A+r_B))\widehat{\mathbf{p}_{AB}} + \frac{\mathbf{v}_A+\mathbf{v}_B}{2}$$

Type-II Constraint - Consistent velocity orientation: A sufficient condition to avoid collisions among multiple agents is to ensure that each agent chooses a velocity that is on the same side of the line bisecting their RVO cone [BPS*08]. We have each agent pass on the side *closest* to its current velocity, imposed by: $\mathrm{FVO}^A_{CB}(\mathbf{v}) = (\phi(\mathbf{v}, (\mathbf{v}_A + \mathbf{v}_B)/2, \phi(\mathbf{v}_A, \mathbf{v}_B, \mathbf{p}^{\perp}_{AB})\mathbf{p}^{\perp}_{AB}) \geq 0)$. This is also a conservative formulation to guarantee collision-free motion.

Any feasible solution to all constraints, which are separately formulated for each agent, will guarantee collision avoidance. In our technique, we compute in a distributed manner, a new, constraint satisfying velocity for each agent which minimizes the deviation from $\mathbf{v}^{des}_A$ – the velocity desired by the underlying simulation algorithm. Let B_1,.., B_N represent the N nearest neighbors of an agent A. We pose the computation of a new velocity ($\mathbf{v}^{new}_A$) as the following optimization problem:

Minimize $\|(\mathbf{v}^{new}_A - \mathbf{v}^{des}_A)\|_2$ such that

$$\left(\left(\sim FVO^A_{LB1}(\mathbf{v}^{new}_A) \cup \sim FVO^A_{RB1}(\mathbf{v}^{new}_A) \cup \sim FVO^A_{TB1}(\mathbf{v}^{new}_A)\right) \cap \sim FVO^A_{CB1}(\mathbf{v}^{new}_A)\right) \cap$$
$$\vdots$$
$$\left(\left(\sim FVO^A_{LBN}(\mathbf{v}^{new}_A) \cup \sim FVO^A_{RBN}(\mathbf{v}^{new}_A) \cup \sim FVO^A_{TBN}(\mathbf{v}^{new}_A)\right) \cap \sim FVO^A_{CBN}(\mathbf{v}^{new}_A)\right)$$

This is a quadratic optimization function with non-convex linear constraints for each agent. It can be shown to be NP-Hard [Kan00] for non-constant dimensions via reduction to quadratic integer programming . It has a polynomial time solution when the dimensionality of the constrains is constant – two in our case.

We refer to union of each neighbor's FVO as its potentially colliding region ($\mathcal{PCR}$), and the boundary segments of each neighbor's FVO as collectively the Boundary Edges ($\mathcal{BE}$). $\mathcal{BE}$ consists of $4N$ boundary segments – 4 from each neighbor of A. We exploit the geometric nature of the problem to derive the following lemma (proof in appendix):

Lemma 1: *If $\mathbf{v}^{des}_A$ is inside $\mathcal{PCR}$, $\mathbf{v}^{new}_A$ must lie on $\mathcal{BE}$.*

In many simulations, there are other constraints on the velocity of an agent. For example, kineodynamic constraints [LaV06], which impose certain bounds on the motion (e.g. maximum velocity or maximum acceleration). In case the optimal solution to the quadratic optimization problem doesn't satisfy these bounds, we relax *the constraints* by removing the furthest agent from A, and recompute the optimal solution by considering only $N-1$ agents. This relaxation step is carried on until an optimal solution satisfying all the constraints is obtained.

3.3. ClearPath-1: Guaranteed Collision Avoidance

A key aspect of our algorithm is to derive rigorous conditions for collision avoidance during a given interval. This is given by the following theorem (with a proof in the appendix):

Theorem 1: *If ClearPath finds a feasible solution for all the agents, then the resultant path is guaranteed to be collision-free.*

Based on the collision free guarantees, we use the following algorithm to compute collision free path for each agent.

ClearPath-1: Our goal is to compute a collision-free path during discrete time interval ΔT. In case there exists a feasible solution to the optimization algorithm, we have a collision-free solution. However, it is possible that the optimization algorithm may not find a feasible solution. It is possible that there may be no collision-free path for the entire interval or our constraints are overly conservative. In this case, we reduce the time interval to (ΔT/2) and recompute the constraints and the feasible solution for a shorter duration. This process is repeated till we find a feasible solution.

3.4. ClearPath-2: Relaxing Collision Constraints

It is possible that the algorithm highlighted above may need to consider a very short interval to find a feasible solution. Every step of the bisection approach involves solving a new optimization problem. Furthermore, the Type-II constraint in the optimization formulation can be overly restrictive. In this case, we present an alternate algorithm that only considers Type-I constraints in terms of the optimization formulation. The resulting algorithm, ClearPath-2, computes $\mathbf{v}_A^{new}$ with respect to 3N constraints. In this case, some agents may collide and we use the following scheme to resolve collisions.

Collision Resolution: In cases when agents are colliding, they should choose a new velocity that resolves the collision as quickly as possible. This results in an additional constraint in velocity space that we conservatively approximate as a cone. In this case, the $\mathcal{PCR}$ is the intersection between two circles. The first circle is the set of maximal velocities reachable by an agent in a single time step (a circle of radius $a_{max} \times \Delta T$). The second circle is the Minkowski difference of the two agents: $B \oplus -A$. The region which lies in the first circle, but not the second is the set of velocities which escapes the collision in one time step. The region which lies in both circles are reachable velocities which *fail* to resolve the collision next time step. This area is conservatively approximated with a cone, creating an additional constraint for any colliding agents. These constraints are combined with the existing $\mathcal{PCR}$ from the neighboring agents. A colliding agent's preferred velocity is set to zero so that the smallest possible velocity which resolves the collision is chosen. This will minimize oscillations due to collision resolution.

3.5. ClearPath Implementation

We use the mathematical formulation to design a fast algorithm to compute a collision-free velocity for each agent independently. Specially, we exploit Lemma 1 and compute all possible intersections of the boundary segments of $\mathcal{BE}$ with each other. Consider segment X in Fig. 3(c). The k intersection points of the FVO region labeled as $X_1, .., X_k$. Note that these points are stored in a sorted order of increasing distance from the corresponding end point (X_0) of the segment. We further classify each intersection point as being *Inside* or *Outside* of $\mathcal{PCR}$. After performing this classification, we now classify the subsegments between these points as being *Inside* or *Outside* of the $\mathcal{PCR}$, based on the following lemma (proof in appendix):

Lemma 2: *The first subsegment along a segment is classified as Outside iff both its end points are tagged as Outside. Any other subsegment is classified as Outside iff both its end points are Outside, and the subsegment before it is Inside the $\mathcal{PCR}$.*

For example, both X_0 and X_1 are tagged as *Outside*, and hence the subsegment X_0X_1 is tagged as *Outside*. However, the subsegment X_1X_2 is tagged as *Inside* since X_0X_1 is *Outside* the $\mathcal{PCR}$. The closest point on the *Outside* subsegments of $\mathcal{BE}$ is the new velocity.

ClearPath performs the following steps for each agent:
Step 0. Given an agent, query the KD-tree for the N nearest agents, and compute the FVO constraints to arrive at $\mathcal{BE}$.
Step 1. Compute the normals of the each segment in $\mathcal{BE}$.
Step 2. Compute the intersection points along each segment of $\mathcal{BE}$ with the remaining segments of $\mathcal{BE}$.
Step 3. Classify the intersection points as *Inside* or *Outside* of the $\mathcal{PCR}$.
Step 4. Sort the intersection points for each segment with increasing distance from its corresponding end point.
Step 5. Classify the subsegments along each segment as *Inside* or *Outside*, and compute/maintain the closest point for the *Outside* subsegments.
Step 6. In case the resulting solution does not satisfy the kineodynamic or velocity constraints, relax the constraints by removing the FVO corresponding to the furthest neighbor by distance, and repeat the algorithm with fewer agents.

For M number of total intersections segments in $\mathcal{BE}$, the runtime of the algorithm for a single agent is $O(N(N+M))$.

4. P-ClearPath: Parallel Collision Avoidance

The current trend is for processors to deliver high-performance through multithreading and wider SIMD instructions. In this section, we describe a data-parallel extension of ClearPath that can exploit the capabilities of current multi-core CPUs and many-core accelerators. The algorithms described in this section are applicable to both ClearPath-1 and ClearPath-2 variants of ClearPath.

ClearPath operates on a per-agent basis in a distributed manner, finding each agent's nearest neighbors and computing a collision-free velocity with respect to those neighbors. There are *two* fundamental ways of exploring Data-Level parallelism (DLP).

Intra-Agent: Consider Fig. 4(a). For each agent, we explore DLP within the ClearPath computation. Since the agents operate in 2D, they can perform their X and Y coordinate updates in a SIMD fashion. This approach does not scale to wider SIMD widths.

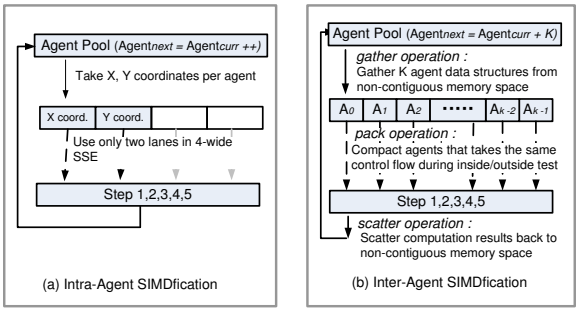

Figure 4: *Data-Parallel Computations: (a) Intra-Agent SIMDfication for SSE (b) Inter-Agent SIMDfication for wide SIMD*

Inter-Agent: We can operate on multiple agents at a time, with each agent occupying a slot in the SIMD computation. This approach is scalable to larger SIMD widths, but needs to handle the following two issues:

1. Non-contiguous data access: In order to operate on multiple agents, ClearPath requires *gathering* their obstacle data structure into a contiguous location in memory. After computing the collision-free velocity, the results need to be *scattered* back to their respective non-contiguous locations. Such data accesses become a performance bottleneck without efficient *gather*/*scatter* operations.
2. Incoherent branching: Multiple agents within a SIMD register may take divergent paths. This degrades SIMD performance, and is a big performance limiter during intersection computations and inside/outside tests. One or more of the agents may terminate early, while the remaining ones may still be performing comparisons.

Current SSE architectures on commodity CPUs do not have efficient instructions to resolve the above two problems. Therefore, when we used the intra-agent SIMDfication approach we obtained only moderate speedups (see Section 5). For the remainder of this section, we focus on exploiting wider SIMD, with the SIMD width being $\mathcal{K}$-wide.

P-ClearPath adopts the inter-agent approach, and performs computation on $\mathcal{K}$ agents together. Fig. 4(b) shows a detailed mapping of the various steps in ClearPath algorithm. For collision-free velocity computation, each agent A_i is given as input its neighboring velocity obstacles (truncated cones) and the desired velocity. The steps performed by each agent are described in Section 3.5.

We start by *gathering* the obstacle data structure of $\mathcal{K}$ agents into contiguous chunks of memory and then loading various fields as needed by the SIMD operation. Although each of the steps listed there map themselves well to SIMD, there are a few *important issues* that need to be addressed.

1. Varying number of neighbors for each agent: This affects each of the steps, decreasing the SIMD utilization. For example, if one of the agents in the SIMD computation has N neighbors, and the second one has $N/2$, the second agent is masked out [SCS*08] for half of the execution and does not perform any computation. To address this, we reorder agents based on their neighbor count, and execute agents that have the same number of neighbors together in a SIMD fashion. Since number of agents is relatively small, reordering runs in linear time and takes up insignificant portion of runtime.

2. Classifying points as inside/outside: This is the most important part of the algorithm. While classifying points as being *inside* or *outside* of the truncated cones, we test their orientation with respect to each truncated cone. As soon as it is detected being *inside* any of the cones, it does not need to be tested against the remaining cones. However, it is often the case that some other point within the SIMD register is still being tested and the computation for other lanes is wasted. In the worst case, the SIMD utilization may be as low as $1/\mathcal{K}$. We exploit the *pack instructions* [SCS*08] to improve the efficiency of this step.

We adapt the ClearPath algorithm to improve the efficiency as follows. After testing the orientation w.r.t. the *first* cone, we *pack* the points contiguously as described above. In subsequent iterations, the points are loaded and tested against the next cone, and the process repeated. Note that the SIMD code tests each point against the same number of points as the scalar version of the code. However, to improve SIMD efficiency, the points are *packed*, and then retrieved for each cone they are checked against. This increases the overhead, but improves the SIMD efficiency to around $\mathcal{K}/3$-$\mathcal{K}/2$. Note that after termination, the computed results need to be *scattered* to appropriate memory locations.

With the above discussed modifications, and appropriate support for *gather*/*scatter* and *pack* instructions, P-ClearPath should achieve around $\mathcal{K}/2$ SIMD speedup as compared to the scalar version.

5. Implementation and Results

In this section, we describe implementation of our algorithms and report performance on various benchmarks.

5.1. Multi-agent simulation

To test ClearPath, we incorporated it into two open-source crowd/multi-agent simulation systems: the RVO-Library [BLM08] and OpenSteer [Rey99]. These simulations provide the desired velocities, which ClearPath minimally modifies to provide collision free motion among the agents. A diagram of this process is shown in Fig. 5. *New Velocities* were applied using simple Euler integration. The *Nearest Neighbor List* was obtained efficiently using a KD-tree. The algorithm for the *Desired Velocity* for each agent depends on the simulation and is described below.

RVO-Library: This library provides a global goal for each agent and can perform collision avoidance. We removed the collision avoidance, using ClearPath instead. The global navigation uses a graph-based roadmap that is pre-computed

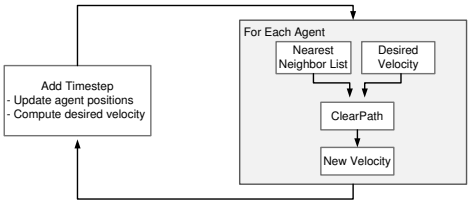

Figure 5: *Each time step, for each agent, ClearPath uses the agent's desired velocity (from a multi-agent simulation), and its of nearby neighbors and computes a new minimally perturbed velocity which avoids collisions. This is used as the agent's new velocity.*

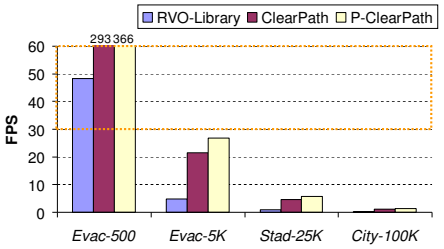

Figure 6: *Performance of RVO-Library, ClearPath and SSE implementation of P-ClearPath measured on a single core Xeon. Target FPS range of* $30-60$ *is highlighted. The ClearPath implementation is about 5X faster than RVO-Library.*

for a given environment. At runtime, A* is used to search the graph and compute a desired path for each agent towards its goal position. The direction along the path scaled by the agent's desired speed provides its desired velocity, $\mathbf{v}^{des}$.

OpenSteer: OpenSteer uses steering forces to guide agents towards their goals and away from each other. We can take either this output as an agent's desired velocity or optionally replace OpenSteer's collision avoidance with ClearPath's.

We used different kinds of benchmarks to evaluate the performance of our algorithms. These vary from simple game-like scenes with a few dozen agents to complex urban scenes with tens to hundreds of thousands of agents. In our tests, the collision avoidance part of local navigation can take $50\% - 80\%$ of the total runtime and is a major bottleneck in the overall multi-agent system. Each agent is modeled as a heterogeneous agent and we perform separate collision avoidance on each agent in a distributed manner. All timings reported in this section are based on the ClearPath-2 algorithm described in Section 3.5.

Performance Comparison of Serial Algorithms: We compared the performance of the serial implementation of ClearPath with roadmaps, versus the original RVO-Library and open source implementation of collision-avoidance in OpenSteer. All of them were running on a single Xeon core. Even though each of these algorithms performs a goal-directed simulation where each agent has a desired goal, they still result in different agent behaviors with varying velocities and motions.

We observe $8-12$X improvement in performance when using ClearPath over the original RVO-Library (Fig. 6), and the absolute running time of ClearPath is comparable to the collision-avoidance routine in OpenSteer. However, OpenSteer can result in many collisions among the agents (shown in the video and Fig. 8), making a direct comparison very difficult. Fig. 6 shows the performance of RVO-Library, ClearPath and P-ClearPath.

5.1.1. Behavior Evaluation

We set up artificial scenarios to evaluate the local navigation and some emergent behaviors of ClearPath: lane-formation, arching at narrow passages, slow down in congestion, response to collisions, avoiding each other, jamming at exits, swirling to resolve congestion, vortices, etc.

Circle-1K: $1,000$ agents start arranged uniformly around a circle and try to move directly through the circle to their antipodal position on the other side (Fig. 2). We observe swirling behavior in the middle.

4-Streams: $2,000$ agents are organized as four streams that walk along the diagonals of the square. This is similar to the benchmark in Continuum Crowds [TCP06], though ClearPath results in different behaviors, including smooth motion, lane formation and some swirls.

Back&Forth: 10 - 100 agents move back and forth along a line. This test is run side-by-side with OpenSteer to compare the number of collisions of unmodified OpenSteer vs. OpenSteer combined with ClearPath. With both techniques, we see some agents smoothly avoiding each other. However, when ClearPath local collision avoidance algorithm is added, we see *all* agents avoiding penetrations.

5.1.2. Complex Scenarios

We set up different scenarios and also measure the scalability of the algorithm as we increase the number of agents.

Building Evacuation: The agents move towards the goal positions corresponding to the exit signs (Fig. 1). We use three versions of this scenario with 500 (Evac-500), 1K (Evac-1K) and 5K (Evac-5K) agents.

Stadium Scene: We simulate the motion of 25K agents as they exit from their seats out of a stadium. The scene has around 1.4K obstacles. The agents move towards the corridors and we observe congestion and highly-packed scenarios. We denote this benchmark as Stad-25K.

City Simulation: We use a city model with buildings and streets with 1.5K obstacles. The agents move at different speeds and overtake each other and avoid collisions with oncoming agents. We simulate three versions with 10K (City-10K), 100K (City-100K) and 250K (City-250) agents.

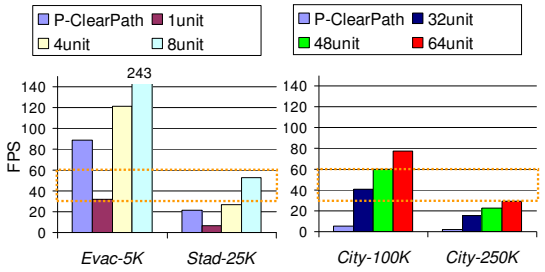

Figure 7: *Performance (FPS) of P-ClearPath on SSE (left most column) and Larrabee (with different units) architectures. Target FPS range of* $30-60$ *is highlighted.*

5.2. Parallel Implementation

We parallelized our algorithm across multiple agents since the computation performed by each agent is local and independent of the remaining ones. Specifically, we tested ClearPath performance on two kinds of parallel processors:

1. Multi-core Xeon processors: A PC workstation with a Intel quad-core Xeon processor (X5460) running at 3.14 GHz with 32KB L1 and 12MB L2 cache. Each core runs a single thread. There is no support for gather/scatter operations.

2. Many-core Larrabee simulator: Larrabee [SCS*08] is an x86-based many-core processor architecture based on small in-order cores. A Larrabee processor core consists of a vector unit (VPU) together with a scalar unit augmented with 4-way multi-threading. VPU supports 16 32-bit float or integer operations per clock. Each core has 32KB L1 and 256KB L2 cache. Hardware support for masking, gather/scatter, and pack instructions allows us to exploit the substantial amount of fine-grained parallelism in P-ClearPath.

5.2.1. Data-Parallelism

Fig. 6 shows the improvement due to SSE instructions for P-ClearPath on Xeon processors. We observe only about $25-50\%$ speedup with SSE instructions as Xeon processors do not support scatter and gather instructions. Fig. 7 shows the performance of P-ClearPath on SSE and Larrabee architectures. For Larrabee, we measured performance in terms of Larrabee units. A Larrabee unit is defined to be one Larrabee core running at 1 GHz. The reported performance data is derived from detailed performance simulation using a cycle-accurate system simulator, which is used in and validated for designing Intel multi-core CPUs.

5.2.2. Thread-Level Parallelism (TLP)

One of the issues that affects scaling is the load balancing amongst different threads. Some agents in dense scenarios may perform more computations than those in sparse regions, as they consider more neighbor s within the discrete optimization computation. Hence, a static partitioning of agents amongst the threads may suffer from severe load balance problems, especially in simulations with few number of agents for large number of threads. The main reason is that the agents assigned to some specific thread(s) may finish their computation early, while the remaining ones are still performing computations. We use a scheme based on dynamic partitioning of agents to reduce the load imbalance. Specifically, we use Task Queues [MKH90], and decompose the execution into parallel tasks consisting of small number of agents. This allows the runtime system to schedule the tasks on different cores. In practice, we *improve* our scaling by *more than 2X* as compared to static partitioning for 16 threads. We observe this speedup in small game like scenarios with tens or hundreds of agents. By exploiting TLP, P-ClearPath achieves around 3.8X parallel speedup on the quad-core.

Additionally, in Larrabee, we also evaluated the combined benefits of TLP and data-level parallelism. Running four threads per core, the working set fits in L1 data cache and the implementation is not sensitive to memory latency issues. 16-wide SIMD provides around 4X scaling over scalar code. Hardware gather/scatter provides 50% additional speedup, resulting in 4.5X scaling. The pack instructions provide additional scaling to achieve the overall 6.4X scaling.

6. Analysis and Comparisons

6.1. Performance Analysis

A key issue for many interactive applications is the fraction of processor cycles that are actually spent in collision avoidance and multi-agent simulation. Collision avoidance can take a high fraction of frame time, especially when we are dealing with dense scenarios with a high number of agents. The left-hand side graph in Fig. 7 highlights the performance on simulations with 5K and 25K agents. P-ClearPath achieves real-time simulation rates of $30-60$FPS with only one Larrabee unit of computation. Using 64 Larrabee units, this would take up less than 2% of the total computation time and the rest of the remaining 98% time could be used for AI, Physics, behavior, rendering and related computations. Even on a quad-core Xeon CPU, P-ClearPath takes up only 20% of the available computation time for 5K agents. As a result, P-ClearPath running on a commodity many-core processor may be fast enough for game-like scenes. The right-hand side graph in Fig. 7 highlights the scalability of ClearPath on large simulations. For a simulation with 100K to 250K agents, we observe real-time rates with 32-64 Larrabee units. The runtime is a function of the number of agents, goals, obstacles and the complexity of the roadmap.

6.2. Behavioral Analysis

Evaluating behavior of the simulation is difficult, as there is no clear standard for the "right way" to avoid collisions, especially among a high number of agents in a dense scenario. However qualitatively, as can be seen from the videos, ClearPath results in smooth collision free motion. Despite

Figure 8: *A snapshot of the 80 agent Back&Forth demo. OpenSteer (on the left) averages 16 collisions per frame. When combined with ClearPath (on the right) the simulation is collision free. Two particularly egregious collisions are circled in red.*

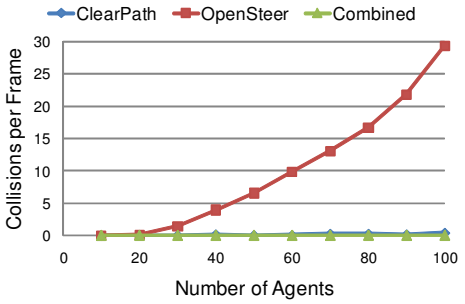

Figure 9: *As density increases, the number of collisions per frame in OpenSteer increases. Using ClearPath on its own, or combining OpenSteer with ClearPath results in practically no collisions.*

the large increase in speed, we still provide behavior that is similar to the original RVO-Library.

Quantitatively, we can directly measure the number of times agents are colliding. In the 80 agent version of the **Back&Forth** benchmark, OpenSteer agents were colliding with each other nearly 16 times per frame. In contrast, when ClearPath was added there were no collisions among the agents over the entire simulation (Fig. 8). As the density increases OpenSteer may result in more collisions, whereas using ClearPath or adding it to OpenSteer results in virtually no collisions regardless of the density of the agents (Fig. 9).

6.3. Comparison and Limitations

In this section, we compare the features of the multi-agent or crowd systems that use the parallel capabilities of GPUs or multiple CPUs. Our multi-agent system based on P-ClearPath can handle heterogeneous agents, global navigation and support collision response between the agents. Some earlier algorithms also offered similar capabilities. These include Parallel-SFM [QMHz03], which is an implementation based on a social force model that parallelizes the simulation process over 11 PCs and used for simulations with thousands of agents. Sud et al. [SAC*07] used GPU-based discretized Voronoi diagrams for multi-agent navigation (MANG), but this approach doesn't scale to large number of agents. It is hard to make direct comparisons with Parallel-SFM and MANG, as they have very different behavior than our system.

Recently, Bleiweiss [Ble09] ported the RVO library to the NVIDIA GTX280 architecture. As compared to their GPU performance numbers, P-ClearPath is around 1.8X faster on quad-core Xeon, and around 4.9X faster on 8 Larrabee units. There are other approaches that can handle some complex scenarios. But it is hard to make direct comparison with them because some of the underlying features of these approaches are different. FastCrowd [CM04] is an implementation of a similar social force model on a single GPU, but it doesn't include collision response. PSCrowd [Rey06] implements a simple flocking model on a Cell Processor, but provides only limited collision avoidance. Continuum Crowds [TCP06] is designed to handle homogeneous groups of large agents. It achieves impressive performance for large homogeneous groups, but is inappropriate for heterogeneous crowds.

Limitations: ClearPath has some limitations. The FVO constraints highlighted in Section 3.2 are conservative. It is possible that there is a collision free path for the agents, but our algorithm may not be able to compute it. Moreover, we compute a new velocity for each agent, $\mathbf{v}^{new}$, which can change behavior of the agents or their path for the rest of the simulations. The data parallel algorithm can obtain up to 50% improvement as a function of SIMD width and the performance varies based on cache size and memory bandwidth.

7. Conclusions and Future Work

In this paper, we present a robust algorithm for collision avoidance among multiple agents. Our approach is general and works well on complex multi-agent simulations with tightly-packed and dense scenarios. The algorithm is almost one order of magnitude faster than prior VO-based approaches. Moreover, our algorithm can be combined with other crowd simulation systems and used to generate collision-free motion for each agent (e.g. OpenSteer). We describe a parallel extension using data-level and thread-level parallelism and use that for real-time collision avoidance in scenarios with hundreds of thousands of agents.

There are many avenues for future work. We would like to port P-ClearPath to many-core GPUs and evaluate its runtime performance. We would like to compare and validate the agent behavior generated by ClearPath with other recent techniques (e.g. [KSHF09]) and real-world data. Our general framework can be extended to use linear constraints to maintain a convex region of admissible velocities, allowing for potential performance improvements [BGLM09]. An interesting extension would be to incorporate other constraints related to human dynamics and human behavior with

ClearPath. Finally, we would like to integrate ClearPath with modern game engines.

Acknowledgements

This research is supported in part by ARO Contract W911NF-04-1-0088, NSF award 0636208, DARPA/RDECOM Contracts N61339-04-C-0043 and WR91CRB-08-C-0137, Intel, and Microsoft. The first author is also supported by an Intel fellowship.

References

[BGLM09] BERG J., GUY S. J., LIN M., MANOCHA D.: Reciprocal n-body collision avoidance. In *International Symposium on Robotics Research (To Appear)* (2009), Springer.

[Ble09] BLEIWEISS A.: Multi agent navigation on gpu. http://developer.download.nvidia.com/presentations/2009/GDC/MultiAgentGPU.pdf, 2009.

[BLM08] BERG J., LIN M., MANOCHA D.: Reciprocal velocity obstacles for realtime multi-agent navigation. *Proc. of IEEE Conference on Robotics and Automation* (2008), 1928–1935.

[BPS*08] BERG J., PATIL S., SEWALL J., MANOCHA D., LIN M.: Interactive navigation of individual agents in crowded environments. *Proc. of ACM Symposium on I3D* (2008), 139–147.

[Cam97] CAMERON S.: Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conf. on Robotics and Automation* (1997), 3112–3117.

[CM04] COURTY N., MUSSE S.: *FastCrowd: Real-Time Simulation and Interaction with Large Crowds based on Graphics Hardware*. Tech. rep., INRIA, March 2004.

[Eri04] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann, 2004.

[FN06] FOUDIL C., NOUREDDINE D.: Collision avoidance in crowd simulation with priority rules. *European Journal of Scientific Research 15*, 1 (2006).

[FS98] FIORINI P., SHILLER Z.: Motion planning in dynamic environments using velocity obstacles. *International Journal on Robotics Research 17*, 7 (1998), 760–772.

[FSL07] FULGENZI C., SPALANZANI A., LAUGIER C.: Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *ICRA* (2007), pp. 1610–1616.

[FT87] FAVERJON B., TOURNASSOUD P.: A local based approach for path planning of manipulators with a high number of degrees of freedom. *ICRA* (1987), 1152–1159.

[HBJW05] HELBING D., BUZNA L., JOHANSSON A., WERNER T.: Self-organized pedestrian crowd dynamics and design solutions: Experiments, simulations and design solutions. *Transportation Science 39*, 1 (2005), 1–24.

[Kan00] KANN V.: Maximum quadratic prog. www.nada.kth.se/∼viggo/wwwcompendium/node203.html, 2000.

[KLK*08] KANEHIRO F., LAMIRAUN F., KANOUN O., YOSHIDA E., LAUMOND H.: A local collision avoidance method for non-strictly convex polyhedral. In *Robotics: Science and Systems* (2008).

[KP07] KLUGE B., PRASSLER E.: Reflective navigation: Individual behaviors and group behaviors. In *Proc. of International Conf. on Robotics and Automation* (2007), pp. 4172–4177.

[KSHF09] KAPADIA M., SINGH S., HEWLETT W., FALOUTSOS P.: Egocentric affordance fields in pedestrian steering. In *I3D '09* (2009), ACM, pp. 215–223.

[LaV06] LAVALLE S. M.: *Planning Algorithms*. Cambridge University Press (also at http://msl.cs.uiuc.edu/planning/), 2006.

[LD04] LAMARCHE F., DONIKIAN S.: Crowd of virtual humans: a new approach for real-time navigation in complex and structured environments. *CG Forum 23*, 3 (2004), 509–518.

[Lin93] LIN M.: *Efficient Collision Detection for Animation and Robotics*. PhD thesis, U.C. Berkeley, December 1993.

[MKH90] MOHR E., KRANZ D. A., HALSTEAD JR. R. H.: Lazy task creation: a technique for increasing the granularity of parallel programs. In *Proc. of the ACM conference on LISP and functional programming* (1990).

[MT97] MUSSE S. R., THALMANN D.: A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation* (1997), 39–51.

[PAB08] PELECHANO N., ALLBECK J., BADLER N.: *Virtual Crowds: Methods, Simulation, and Control*. M.C. Publ., 2008.

[PPD07] PARIS S., PETTRE J., DONIKIAN S.: Pedestrian reactive navigation for crowd simulation: a predictive approac. *Computer Graphics Forum 26* (2007), 665–674.

[QMHz03] QUINN M. J., METOYER R. A., HUNTER-ZAWORSKI K.: Parallel implementation of the social forces model. In *in Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics* (2003), pp. 63–74.

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics 21* (1987), 25–34.

[Rey99] REYNOLDS C. W.: Steering behaviors for autonomous characters. *Game Developers Conference* (1999).

[Rey06] REYNOLDS C.: Big fast crowds on ps3. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames* (2006), ACM, pp. 113–121.

[SAC*07] SUD A., ANDERSEN E., CURTIS S., LIN M., MANOCHA D.: Real-time path planning for virtual agents in dynamic environments. *Proc. of IEEE VR* (2007), 91–98.

[SCS*08] SEILER L., CARMEAN D., SPRANGLE E., FORSYTH T., ET AL.: Larrabee: A Many-Core x86 Architecture for Visual Computing. *ACM Trans. Graph. 27*, 3 (2008), 1–15.

[SGA*07] SUD A., GAYLE R., ANDERSEN E., GUY S., LIN M., MANOCHA D.: Real-time navigation of independent agents using adaptive roadmaps. *Proc. of ACM VRST* (2007), 99–106.

[SLS01] SHILLER Z., LARGE F., SEKHAVAT S.: Motion planning in dynamic environments: obstacles moving along arbitrary trajectories. In *ICRA* (2001), vol. 4, pp. 3716–3721.

[SNH01] SUGIYAMA Y., NAKAYAMA A., HASEBE K.: 2-dimensional optimal velocity models for granular flows. In *Pedestrian and Evacuation Dynamics* (2001), pp. 155–160.

[TCP06] TREUILLE A., COOPER S., POPOVIC Z.: Continuum crowds. *Proc. of ACM SIGGRAPH* (2006), 1160 – 1168.

[TOCD06] THALMANN D., O'SULLIVAN C., CIECHOMSKI P., DOBBYN S.: *Populating Virtual Environments with Crowds*. Eurographics Tutorial Notes, 2006.

[vdBLM08] VAN DEN BERG J., LIN M., MANOCHA D.: Reciprocal velocity obstacles for realtime multi-agent navigation. *Proc. of IEEE Conference on Robotics and Automation* (2008).

[YMMT08] YERSIN B., MAÏM J., MORINI F., THALMANN D.: Real-time crowd motion planning. *The Visual Computer 24* (Oct 2008), 859–870.

[YT07] YU Q., TERZOPOULOS D.: A decision network framework for the behavioral animation of virtual humans. In *Proc. of the Symposium on Computer Animation* (2007), pp. 119–128.

Appendix A. Derivations and Proofs

A.1. Derivations of $\Gamma_{AB}(\mathbf{v})$ and $\gamma_{AB}(\mathbf{v})$

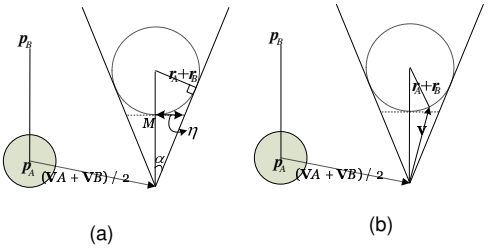

Figure 1: *Derivation of (a) $\Gamma_{AB}(\mathbf{v})$ and (b) $\gamma_{AB}(\mathbf{v})$*

For symbols, refer to Section 3.2.

In Fig. 1(a), $\sin\alpha = \frac{(r_A+r_B)}{|\mathbf{p}_{AB}|}$. Therefore,

$$\eta = \tan\left(\sin^{-1}((r_A+r_B)/|\mathbf{p}_{AB}|)\right)(|\mathbf{p}_{AB}| - (r_A+r_B))$$

M is computed as follows:

$$M = (|\mathbf{p}_{AB}| - (r_A+r_B))\widehat{\mathbf{p}_{AB}} + (\mathbf{v}_A+\mathbf{v}_B)/2$$

$$\Gamma_{AB}(\mathbf{v}) = \lambda\left(M - (\widehat{\mathbf{p}^{\perp}_{AB}}\eta), \mathbf{p}^{\perp}_{AB}\right) (Sec.3.2)$$

To derive $\gamma_{AB}(\mathbf{v})$, we define $|\mathbf{v}| = \delta$ (refer to Fig. 1(b)).

$$t_{collision} = (\mathbf{v}_A+\mathbf{v}_B)/2 + \delta/(2\Delta T)$$

Using the triangle cosine rule,

$$\delta = (\mathbf{p}_{AB})\cdot\hat{\mathbf{v}} - \sqrt{((\mathbf{p}_{AB})\cdot\hat{\mathbf{v}})^2 - |\mathbf{p}_{AB}|^2 + (r_A+r_B)^2}$$

Therefore,

$$\gamma_{AB}(\mathbf{v}) = \frac{\mathbf{v}_A+\mathbf{v}_B}{2} + \hat{\mathbf{v}}\left(\frac{(\mathbf{p}_{AB})\cdot\hat{\mathbf{v}} - \sqrt{((\mathbf{p}_{AB})\cdot\hat{\mathbf{v}})^2 - |\mathbf{p}_{AB}|^2 + (r_A+r_B)^2}}{2\cdot\Delta T}\right)$$

A.2. Proof of Lemma 1

We prove by contradiction. Let $\mathbf{v}_A^{new}$ not be on the edge of one of the segments of *PCR* and say it minimizes the distance from $\mathbf{v}_A^{des}$. Consider a circle of radius $\varepsilon|\mathbf{v}_A^{new} - \mathbf{v}_A^{des}|$ centered at $\mathbf{v}_A^{new}$ ($\varepsilon > 0$). The circle is completely outside the *PCR*, and we have a point $\mathbf{v}_A^{new} + \varepsilon(\mathbf{v}_A^{des} - \mathbf{v}_A^{new})$ that is outside and at a distance of $(1-\varepsilon)|\mathbf{v}_A^{new} - \mathbf{v}_A^{des}|$ from $\mathbf{v}_A^{des}$. Thus we arrive at a contradiction. Therefore, $\mathbf{v}_A^{new}$ must lie on the boundary segment of one of the neighbors.

A.3. Proof of Lemma 2

This follows from the fact that the truncated cones are convex regions. Hence once we identify a point that is outside the remaining truncated cones, the subsegment before it would be inside or outside. We use this fact to compute the inside/outside regions on the constraint cone boundaries.

ClearPath Steps	% Time Breakdown	SIMD Scaling
Step 1	14%	7.9X
Step 2	39%	5.6X
Step 3	21%	5.9X
Step 4	13%	9.3X
Step 5	13%	5.6X
Total	100%	6.4X

Table B.1: *Average time (%) spent in various steps (Section 3.5) of ClearPath and the corresponding SIMD scalability on Larrabee simulator of execution cycles as compared to the scalar version.*

System	Hardware	Year	Agents	FPS
FastCrowd	GPU	2005	10,000	35†
Cont. Crowds	CPU	2006	10,000	5
PSCrowd	Cell (6 SPUs)	2006	15,000	8
ClearPath	4 Cores	2009	10,000	**36**
ClearPath	LRB (32 Cores)	2009	10,000	**302**

Table B.2: *Performance Comparison with other crowd simulation systems. † includes rendering time.*

A.4. Proof of Theorem 1

All the agents are in a collision-free state if the resultant velocity is outside the VO of its neighbors. $FVO^A_{LBi}(\mathbf{v})$ and $FVO^A_{RBi}(\mathbf{v})$ ensure that the resultant velocity is outside the RVO of the neighboring agents. Furthermore, for every pair of agents, $FVO^A_{CBi}(\mathbf{v})$ ensures that we are on the same side of the line joining the centers of the agents. Hence it follows from RVO pair-wise collision-free property by Berg et al. [vdBLM08] that all pairs of agents are collision-free w.r.t each other, and hence the system is in a collision-free state.

Appendix B. Performance Comparison

Table B.1 shows the SIMD scaling achieved by each of the steps of ClearPath (Section 3.5). We perform packing operations to handle path divergence and early termination of the agent cone intersection and inside/outside computation.

Table B.2 provides performance numbers of recent crowd simulation systems. Since the simulations run on different benchmarks with varying behaviors and on different systems, it is not fair to directly compare the reported runtimes. The performance numbers are given only to provide a rough idea of the relative performance. Some of the other systems use only a few groups of agents pursuing a few distinct goals, while our simulations treat each agent individually with heterogeneous behaviors.