ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation

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 a loosely coupled dynamic units, often called agents. Based on the pioneering work of Reynolds [1987] on distributed behavior models, 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 any agent does not collide with its neighbors. Collision avoidance can quickly become a major bottlenecks in multi-agent simulation, especially in tightly packed scenarios. This is an important issue in computer games, which may only be able to devote less than 5% of processing cycles to collision avoidance and behavior computations. Furthermore, applications such as urban simulations 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 to exploit the current architectural trends. Recent and future commodity processors are becoming increasingly parallel. Specifically, GPUs and the upcoming x86-based many-core processor Larrabee. These processors 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 that 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* that was introduced by Fiorini and Shiller [1998] in robotics for motion planning in dynamic obstacles. We use an efficient velocity-obstacle based formulation that can be combined with any underlying multi-agent simulation. Local collision avoidance computation is reduced to solving a quadratic optimization problem that minimizes the change in the underlying velocity of each agent subject to non-collision constraints (described in Section 3). We present a polynomial-time algorithm for 2D agents to compute collision-free motion. In practice, ClearPath is up to an one order of magnitude faster than prior velocity-obstacle based methods.

Figure 1: Building evacuation: $1,000$ *independent agents in different rooms of a building move towards the two exit signs and cause congestion. P-ClearPath can efficiently perform local collision avoidance for all agents in such tight packed simulations at* 550 *FPS on Intel quad-core Xeon (*3.16*GHz) processor. Our algorithm is more than an order of magnitude faster than prior velocity-obstacle based algorithms.*

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 algorithm, P-ClearPath, exploits the structure of our optimization algorithm and architectural capabilities like gather/scatter, and pack/unpack to provide improved data-parallel scalability. We evaluate its performance in various scenarios on current multi-core CPUs. In practice, P-ClearPath demonstrates 8-15X speedup on a conventional quad-core processor over prior VO-based algorithms.

2 Related Work

Different techniques have been proposed to model behaviors of individual agents, groups and heterogeneous crowds. Excellent recent surveys are given in [Pelechano et al. 2008; Thalmann et al. 2006]. These include methods to model the local dynamics and generating emergent behaviors [Reynolds 1987; Reynolds 1999], psychological effects and cognitive models [Yu and Terzopoulos 2007], cellular automata models and hierarchical approaches. In this section, we give a brief overview of related work in collision detection and avoidance, and parallel algorithms.

2.1 Collision detection and path planning

There is rich literature on collision detection between two or multiple 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) [Ericson 2004]. 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 [LaValle 2006]. 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.

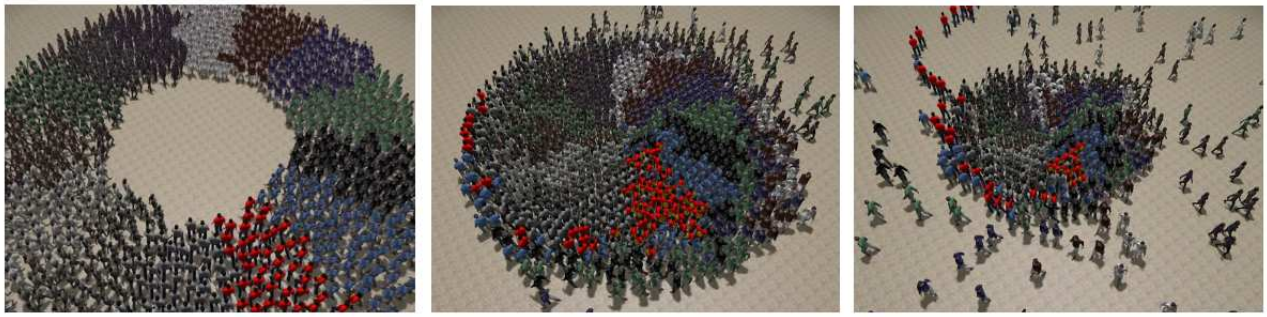

Figure 2: Dense Circle Scenario*: 1,000 agents are arranged uniformly around a circle and move towards antipodal position. ClearPath performs local collision avoidance computation for all agents in this simulation in 4ms on a single 3.16 GHz Xeon core*

2.2 Collision avoidance

Collision avoidance problems have been been studied in control theory, traffic simulation, robotics and crowd simulation. Optimization techniques for local collision detection have been proposed for a pair of objects, including separation or penetration distance computation between convex polytopes [Cameron 1997; Lin 1993] and local collision detection between convex or non-strictly convex polyhedra with continuous velocities [Faverjon and Tournassoud 1987; Kanehiro et al. 2008]. Different techniques have been proposed for collision avoidance in group and crowd simulations [Reynolds 1999; Foudil and Noureddine 2027; Sugiyama et al. 2001; Musse and Thalmann 1997; Thalmann et al. 2006; Helbing et al. 2005; Lamarche and Donikian 2004; Sud et al. 2007b]. These are based on local coordination schemes, velocity models, prioritization rules, force-based techniques, or adaptive roadmaps.

The notion of velocity obstacles (VO) was proposed for motion planning in dynamic environments and has been extended to deal with uncertainity in the sensor data [Fiorini and Shiller 1998; Fulgenzi et al. 2007; Kluge and Prassler 2007]. Recently, Berg et al. [2008a; 2008b] extended the VO formulation for reducing collisions between the agents.

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 agents is governed by some extrinsic and intrinsic parameters and computed in a distributed manner. 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 or path for each agent that either results in no collisions or attempts to reduce the number of collisions during the interval $[T, T+\Delta T]$.

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. 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. Given an agent A, we use $\mathbf{p}_A$, $\mathbf{r}_A$, and $\mathbf{v}_A$ to denote its position, radius and velocity, respectively. We assume that the underlying simulation algorithm uses the intrinsic and extrinsic characteristics of the agent or some high level model to compute a 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 (both in 2D).

3.1 Velocity Obstacles

Our approach is build on velocity obstacles [Fiorini and Shiller 1998]. 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 $\mathbf{v}_B$, as shown in Fig. 3(a).

Recently Van den Berg et al. [2008a; 2008b] presented an extension called RVO. The resulting velocity computation algorithm guarantees an 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 of its neighbors to avoid collisions. RVO algorithm performs a random sampling of the 2D space in the vicinity of $\mathbf{v}_A^{des}$, and chooses a feasible solution that satisfies the constraints. However, the RVO algorithm has the following limitations: RVO performs random sampling, and may not find a collision-free velocity even if there is a feasible solution. Since RVO uses infinite cones, the extent of feasible region decreases as N increases. In practice, the RVO formulation can become overly conservative for tightly packed scenarios.

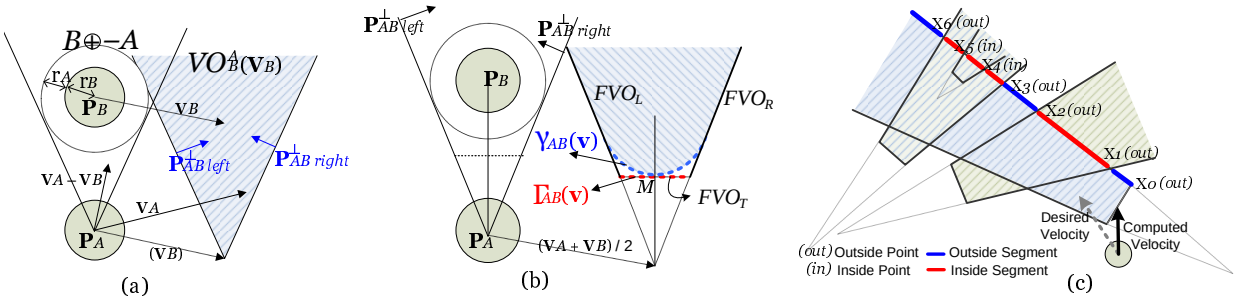

Figure 3: *(a) The Velocity Obstacle $VO^A_B(\mathbf{v}_B)$ of a disc-shaped obstacle B to a disc-shaped agent A. This VO represents the velocities that could result in a collision of A with B (i.e. potentially colliding region). (b) The Fast Velocity Obstacle $FVO^A_B(\mathbf{v}_B)$ of a disc-shaped 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 for multi-agent simulation. Our optimization algorithm performs these classification tests to compute a non-colliding velocity for each agent.*

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.

The original VO or RVO are defined using only two constraints (left and right), as shown in Fig. 3(a). The FVO is defined using a total of four constraints.

The *two* boundary cone constraints of the FVO are the same as that of RVO:

$\text{FVO}_{L^A_B}(\mathbf{v}) = \phi(\mathbf{v}, (\mathbf{v}_A + \mathbf{v}_B)/2, \mathbf{p}^{\perp}_{ABleft}) \geq 0$

$\text{FVO}_{R^A_B}(\mathbf{v}) = \phi(\mathbf{v}, (\mathbf{v}_A + \mathbf{v}_B)/2, \mathbf{p}^{\perp}_{ABright}) \geq 0$

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 the time interval. As shown in Figure 3(b), the truncated cone (expressed as a shaded region) is bounded by a curve $\gamma_{AB}(\mathbf{v})$ (detailed formulation 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 approximation increases the area of the truncated cone by a small amount. This constitutes our additional FVO constraint, represented as: $\text{FVO}_{T^A_B}(\mathbf{v}) =$

$$\Gamma_{AB}(\mathbf{v}) = \lambda\left(M - (\widehat{\boldsymbol{p}_B - \boldsymbol{p}_A})^{\perp} \times \eta, (\boldsymbol{p}_B - \boldsymbol{p}_A)^{\perp}\right) \quad \text{, where}$$

$$\eta = \tan(\sin^{-1}\frac{(\boldsymbol{r}_A + \boldsymbol{r}_B)}{|\boldsymbol{p}_B - \boldsymbol{p}_A|}) \times (|\boldsymbol{p}_B - \boldsymbol{p}_A| - (\boldsymbol{r}_A + \boldsymbol{r}_B)) \text{ and}$$

$$M = (|\boldsymbol{p}_B - \boldsymbol{p}_A| - (\boldsymbol{r}_A + \boldsymbol{r}_B)) \times (\widehat{\boldsymbol{p}_B - \boldsymbol{p}_A}) + \frac{\mathbf{v}_A + \mathbf{v}_B}{2}$$

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 joining their centers [Berg et al. 2008b]. Without loss of generality, we force each agent to choose its *right* side and impose this constraint as: $\text{FVO}_{C^A_B}(\mathbf{v}) = (\phi(\mathbf{v}, \mathbf{p}_A, \mathbf{p}^{\perp}_{AB}) \leq 0)$

Any feasible solution to both types of constraints will guarantee collision avoidance. In this case, we minimize the deviation from $\mathbf{v}^{des}_A$, which is desired by the underlying simulation algorithm, and compute a new velocity w.r.t the FVO constraints. 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 $4N$ non-convex linear constraints. It can be shown to be NP-Hard [Kann 2000] 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 (PCR), and the boundary segments of each neighbor's FVO as collectively the Boundary Edges (BE). BE consists of $4N$ boundary segments – 4 from each neighbor of A. We exploit the geometric nature of the problem, and can derive the following characterization (with proof in the appendix):

Lemma 1: *If $\mathbf{v}^{des}_A$ is inside PCR, $\mathbf{v}^{new}_A$ must lie on the boundary segment of the FVO of one of the neighbors.*

In many simulations, there are other constraints on the velocity of an agent. For example kineodynamic constraints [LaValle 2006] 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 Algorithm

We use the mathematical formulation to design a fast algorithm to compute a collision-free for each velocity. Specially, we exploit Lemma 1 and compute all possible intersections of the boundary segments of BE with each other. Consider segment X in Figure 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 – increasing distance from the corresponding end point (X_0) of the segment. We further classify each intersection point as being *Inside* or *Outside* of PCR. After performed this classification, we now classify the subsegments between these points as being *Inside* or *Outside* of the PCR, based on the following lemma (proof in the 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 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, X_1X_2 is tagged as *Inside* since X_0X_1 is *Outside* the *PCR*.

After classifying the subsegments of BE, we simply consider the *Outside* subsegments, and compute the one closest to $\mathbf{v}_A^{des}$, and return the closest point. The ClearPath algorithm performs the following steps for each agent.

Step 1. Compute the normals of the each of the segments in BE.
Step 2. Compute the intersection points along each segment of BE with the remaining segments of BE.
Step 3. Classify the intersection points as *Inside* or *Outside*.
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 resultant solution does not satisfy the kineodynamic or velocity constraints, relaxing the constraints by removing the FVO corresponding to the furthest neighbor, and repeat the algorithm with fewer agents.

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

3.4 Guaranteed Collision Avoidance

A key aspect of our algorithm is 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 collision-free.*

Based on the collision free guarantees, we use the following algorithm that can compute collision free path for each navigation.

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, than 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 solution 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 can be repeated till we find a feasible solution.

3.5 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 constraint and therefore have only $3N$, as opposed to $4N$ constraints. The resulting algorithm, ClearPath-2, computes $\mathbf{v}_A^{new}$ w.r.t only 3N constraints. In this case, some agents can still collide and we use the following scheme to resolve collisions.

Collision Resolution: In cases when agents are colliding, they should chose a new velocity which resolves the collision as quickly as possible. This results in an additional set of constraints in velocity space which we can conservatively approximate as a cone. In this case, the Potentially Colliding Region is the intersection

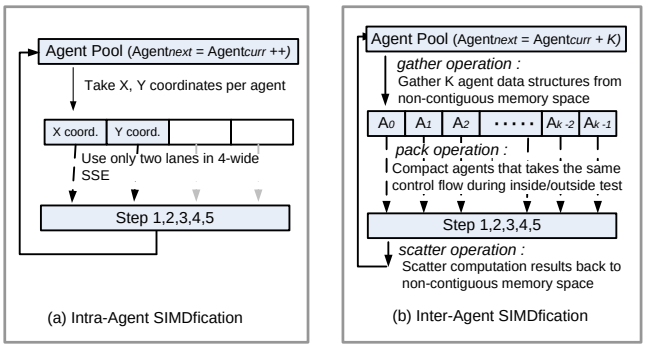

Figure 4: *Data-Parallel Computations: (a) Intra-Agent SIMDification for SSE (b) Inter-Agent SIMDfication for wide SIMD*

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 1^{st} circle, but not the second is the set of velocities which escapes the collision in 1 timestep. The region which lies in both circles is the set of reachable velocities which do not. By conservatively approximating this area with a cone, we can create an additional constraint for any agents in collisions. This constraint is unioned with the existing PCR from the neighboring agents. A colliding agent's preferred velocity is set to $\mathbf{0}$ so that the smallest possible velocity which resolves the collision is chosen. This will minimize oscillations due to collision resolution.

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.

ClearPath operates on a per-agent basis, and computes the nearest neighbors and a collision-free velocity for the next time step. There are *two* fundamental ways of exploring Data-Level parallelism (henceforth referred to as DLP).

Intra-Agent: Consider Figure 4(a). For each agent, we explore DLP within its 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.

Inter-Agent: 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 computation requires *gathering* their obstacle data structure into a contiguous location. After computing the collision-free velocity, the results need to be *scattered* back to their respective non-contiguous locations. Such data accesses become 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 for ClearPath algorithm with intersection and inside/outside tests. One or more of the agents may terminate early, while the remaining ones are still performing their comparison operations.

The current SSE architecture does not have efficient instructions to

resolve the above two problems. Hence, we used the intra-object SIMDfication approach and obtain 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. Figure 4(b) shows a detailed mapping of the various steps in ClearPath computation. 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 delineated in Section 3.3.

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 concerns* that need to be addressed.

1. Varying number of neighbors for each agent: This affects each of the steps, bringing down the SIMD utilization. For example, if one of the agents in the SIMD computation has N neighbors, and the other one has $N/2$, for half of the computation, the other agent is masked out, and does not do any computation. We address this issue by reordering the agents based on their neighbor count, and executing agents that have the same number of neighbors together. Since the number of agents is a relatively small number, this reordering runs in linear time, and takes up insignificant portion of the runtime.

2. Constraint Relaxation by some agents: After computing the collision-free velocity, it is indeed possible for some agents not to have satisfied their kinodynamic or other constraints – thereby going back to the start and computing the solution with one less constraint in an iterative manner. We exploit the *pack instructions* [Seiler et al. 2008] to improve the efficiency. After all the agents have completed one iteration, we identify and mask the lanes that need recomputation, and *pack* their data structure (i.e. the cone information) together in a separate memory location. We perform this step for all the agents. After the first iteration, we have a *contiguous* list of agents that have failed the constraints, and need to be operated upon. We again operate on them – loading $\mathcal{K}$ agents each time and operating on them in a SIMD fashion as before. This process is carried out in an iterative fashion, with all the cones failing the constraints in one iteration being operated upon in the subsequent iteration. Note that after termination, the computed result needs to be *scattered* to the appropriate memory location.

3. 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 w.r.t. each truncated cone, and 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 adapt the ClearPath algorithm to improve the efficiency. After testing the orientation w.r.t. the *first* cone, we *pack* the points contiguously as described above. In the subsequent iteration, the points are loaded and tested against the next cone, and the process repeated. Note that in comparison to the scalar version of the code, each point is tested against the same number of the cones in the SIMD code. However, to improve SIMD efficiency, it is *packed*, and then retrieved for each cone it is checked against. This increases the overhead, but improves the SIMD efficiency to around $\mathcal{K}/3$-$\mathcal{K}/2$ in our benchmarks.

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. We believe it should scale near-linearly with SIMD-width. A detailed analysis with SIMD scaling for each of the steps in the CleatPath algorithm is discussed in Section 5.

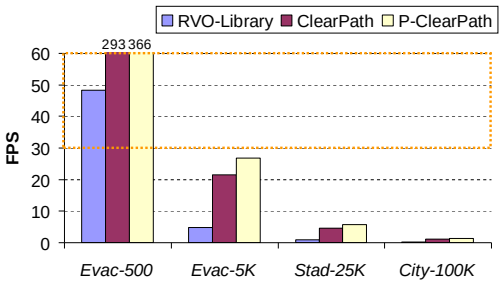

Figure 5: *Performance comparison between the multi-agent simulation systems that use RVO-Library, ClearPath, SSE implementation of P-ClearPath measured on a single core CPU. This graph shows the absolute frame rate the multi-agent simulation systems. The overall simulation using ClearPath is about 5X faster then multi-agent simulation based on RVO-Library. The SSE capability offers additional* $25-50\%$ *speedup.*

5 Implementation and Results

In this section, we describe the implementation of ClearPath and its parallel extension. We evaluate its performance on different processors and highlight their performance on different benchmarks.

5.1 Multi-agent simulation

We used different kind of benchmarks to evaluate the performance of our algorithms. These use simple game like scenes with a few hundred agents to complex scenes with tens to hundreds of thousands of agents. We used a multi-agent system that computes the path of each agent using global and local navigation. The global navigation is based on a graph-based roadmap that is pre-computed for a given environment. We sample a number of nodes around the obstacles in a scene and connect nearby agents with collision-free paths. At runtime, we search the graph using A* and compute a desired path for each agent towards its goal position.

Local Navigation: The local navigation step is performed for each agent at every time step. We use the goal position to compute the desire velocity, $\mathbf{v}_A^{des}$, for the agent A.The local navigation is performed in the velocity space and we use ClearPath to compute the new velocity to reduce collisions with nearby agents. The nearest neighbors for each agent are determined using a KD-tree and we also use simple smoothing approaches to avoid sharp turns in the motion.

In our system, the collision avoidance part of local navigation can take $50-80\%$ of the total runtime and is a major bottleneck in the the performance of the overall multi-agent system. We evaluated our algorithms on different scenarios, varying the number of agents from 500 to 250K. Each agent is modeled as a heterogeneous agent and we perform separate collision avoidance for it. All timings reported in this section are based on ClearPath-2 algorithm described in Section 3.5 and also include timings for collision resolution.

Serial Performance Comparison: We compared the performance of the serial implementation of the collision-avoidance library with RVO-Library [Berg et al. 2008a] and the implementation of collision-avoidance in OpenSteer [Reynolds 1999]. All of them were running on a single Xeon core (running at 3.16 GHz) with no data parallelism. Note that all these algorithms result in different behaviors for the agent and the actual motion and velocities of the agents is different. So it is hard to make direct comparisons, esp. with OpenSteer. We observe $8-12$X improvement over RVO-

Library (see Fig. 5), and the absolute running time of ClearPath is comparable to the collision-avoidance routine in OpenSteer. Fig. 5 shows the absolute frame rates of the simulation system that use RVO-Library, ClearPath and P-ClearPath for collision avoidance.

5.1.1 Behavior Evaluation

We set up artificial scenarios to evaluate the local navigation and some emergent behaviors of ClearPath. We have observed the following emergent behaviors from ClearPath in our benchmarks: lane-formation, vortices, slow down in congestion, respond to collisions, avoiding each other, swirl to resolve congestion, jam at exits, arching at narrow passages, etc.

Circle-1K: $1,000$ agents start arranged uniformly around a circle and try to move directly through the circle to the their antipodal position on the other side (see Fig. 2). The scenario becomes very crowded when all the agents meet in the middle and we observe swirling behavior.

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 [Treuille et al. 2006], though ClearPath results in different behaviors, including smooth motion, lane formation and some swirls.

5.1.2 Complex Scenarios

We setup different scenarios and also measure the scalability of the system as we increase the number of agents.

Building Evacuation: We setup initial position of the agents in different rooms in an office building. The scene has 218 obstacles and the roadmap consists of 429 nodes and 7.2K edges. As part of a drill, the agents move towards the goal positions corresponding to exit signs. The hallway quickly fill up with the agents and there is congestion at the exits, which allow only for $1-2$ agents to leave at a time. We use three versions of this scenario with 500, 1K and 5K agents and they are denoted as Evac-500, Evac-1K, and Evac-5K, respectively.

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 and the roadmap consists of almost 2K nodes and 3.2K edges. The agents moves towards the corridors and we observe a lot of congestion and highly-packed scenarios. We denote this benchmark as Stad-25K.

City Simulation: We have a model of the city with buildings and streets with 1.5K obstacles. The roadmap has 480 nodes and 916 edges. We simulate the motion of different agents as they walk around the city and at intersection. The agents move at different speeds and overtake each other and avoid collisions with oncoming agents. We simulate three versions with 10K, 100K and 250K agents, and denote them as City-10K, City-100K and City-250K, respectively.

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, focused on two kind of parallel processors and systems with different characteristics. They are:

1. Multi-core Xeon processors: We tested the performance of different algorithms on a PC workstation with a Intel quad-core Xeon processor (X5460) running at 3.16 GHz with 32KB L1 cache and 12MB L2 cache. There is no support for gather and scatter operations. Each core runs a single thread.

2. Generic Many-core Processor: Modern many-core processors such Larrabee [Seiler et al. 2008], Cell Processor, or programable GPUs focus on computing throuput, featuring wide vector units (for SIMD parallelism) and hardware support for data gather and scatter. As well as several independant multi-threading processors.

5.2.1 Data-Parallelism

Figure 5 shows the improvement due to SSE instructions for P-ClearPath on Xeon processors. We observe only about $25-50\%$ speedup with SSE instructions as the Xeon processors don't support scatter and gather instructions, have limited support for incoherent branches.

On wide SIMD processors, we expect resonable SIMD utilization when using Inter-Agent SIMDfication. Preliminary simulations suggest that around 8X scaling on 16-wide SIMD is possible.

5.2.2 Thread-level Parallelism (TLP)

One of our goals that affects scaling is the load balancing amongst different threads. Some of the agents in a dense scenarios may perform more computations than the ones in sparse regions, as they consider more neighbors 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 the collision avoidance and other computations. We use a scheme based on dynamic partitioning of agents to reduce the load imbalance. Specifically, we Task Queueing [Mohr et al. 1990], and decompose the execution into parallel tasks consisting of a 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 X5460 for all our benchmarks.

6 Analysis and Comparisons

In this section, we analyze the performance of our approach and compare with other multi-agent and crowd systems.

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. Note that collision avoidance can take a high fraction of frame time, especially when we are dealing with dense scenarios with a high number of agents. On a quad-core Xeon CPU, P-ClearPath takes up only 20% of the available computation time for 5K agents. The rest of the remaining 80% time could be used for AI, Physics, behavior, rendering and related computations. As a result, P-ClearPath running on commodity many-core processor may be fast enough for game-like scenes. For an urban simulation with 100K and 250K agents, we achieve interactive rates.

6.2 Comparison

We compared the performance of our serial implementation of ClearPath with other approaches in Section 5.1. In this section, we compare the features of the multi-agent or crowd systems that use the parallel capabilities of a GPU 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 [Quinn et al. 2003], which is

Figure 6: Urban simulation: *25K independent agents walking in a city model on sidewalks and intersections. P-ClearPath can efficiently perform local collision avoidance for all agents in this simulationat 20 fps on Intel quad-core Xeon (3.14GHz) processor*

an implementation based on a social force model which parallelizes the simulation process over 11 PCs and used for simulations with thousands of agents. A multi-core implementation of RVO-Library on a 16 Core system is described in [Berg et al. 2008b]. Sud et al. [2007a] used GPU-based discretized Voronoi diagrams for multi-agent navigation (MANG), but this approach doesn't scale to very high number of agents. It is hard to make direct comparisons with Parallel-SFM and MANG, as they have very separate behavior than our system.

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 [Courty and Musse 2004] is an implementation of a similar social force model on a single GPU, but it doesn't include collision response. PSCrowd [Reynolds 2006] implements a simple flocking model on a Cell Processor with 6 SPUs, but may not support global navigation. Continuum Crowds [Treuille et al. 2006] can mainly handle homogeneous group of a large agents and has impressive performance numbers on a single CPU for large homogeneous groups.

6.3 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 the behavior of the agents. The data parallel algorithm can obtain up to 50% improvement as a function of SIMD width and the performance also 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. We describe a parallel extension using data-parallelism and thread-level parallelism and use that for real-time collision avodiance in complex 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 P-ClearPath with other systems and real-world data. 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 some game engines.

References

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

BERG, J., PATIL, S., SEAWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of individual agents in crowded environments. *Proc. of ACM Symposium on Interactive 3D Graphics and Games*, 139–147.

CAMERON, S. 1997. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conference on Robotics and Automation*, 3112–3117.

COURTY, N., AND MUSSE, S. 2004. Fastcrowd: Real-time simulation and interaction with large crowds based on graphics hardware. Tech. rep., INRIA, March. http://home.tele2.fr/ncourty/fastCrowd.htm.

ERICSON, C. 2004. *Real-Time Collision Detection*. Morgan Kaufmann.

FAVERJON, B., AND TOURNASSOUD, P. 1987. A local based approach for path planning of manipulators with a high number of degrees of freedom. *Proceedings of International Conference on Robotics and Automation*, 1152–1159.

FIORINI, P., AND SHILLER, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *International Journal on Robotics Research 17*, 7, 760–772.

FOUDIL, C., AND NOUREDDINE, D. 2027. Collision avoidance in crowd simulation with priority rules. *European Journal of Scientific Research 15*, 1.

FULGENZI, C., SPALANZANI, A., AND LAUGIER, C. 2007. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *Proceedings of International Conference on Robotics and Automation*, 1610–1616.

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

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

KANN, V. 2000. Maximum quadratic programming. http://www.nada.kth.se/ viggo/wwwcompendium/node203.html.

KLUGE, B., AND PRASSLER, E. 2007. Reflective navigation: Individual behaviors and group behaviors. In *Proceedings of International Conference on Robotics and Automation*, 4172–4177.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real-time navigation in complex and structured environments. *Computer Graphics Forum 23*, 3, 509–518.

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

LIN, M. 1993. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

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

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

PELECHANO, N., ALLBECK, J., AND BADLER, N. 2008. *Virtual Crowds: Methods, Simulation, and Control*. Morgan and Claypool Publishers.

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

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

REYNOLDS, C. W. 1999. Steering behaviors for autonomous characters. *Game Developers Conference 1999*.

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

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: a many-core x86 architecture for visual computing. In *Proc. of ACM SIGGRAPH*, 1–15.

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

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

SUGIYAMA, Y., NAKAYAMA, A., AND HASEBE, K. 2001. 2-dimensional optimal velocity models for granular flows. In *Pedestrian and Evacuation Dynamics*, 155–160.

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

TREUILLE, A., COOPER, S., AND POPOVIC, Z. 2006. Continuum crowds. *Proc. of ACM SIGGRAPH*, 1160 – 1168.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 119–128.