CEIG - Spanish Computer Graphics Conference (2013)
M. Carmen Juan and Diego Borro (Editors)

Dynamic Footsteps Planning for Multiple Characters

A. Beacco[1], N. Pelechano[1] & M. Kapadia[2]

[1]Universitat Politècnica de Catalunya
[2]University of Pennsylvania

Abstract

Animating multiple interacting characters in real-time dynamic scenarios is a challenging task that requires not only positioning the root of the character, but also placing the feet in the right spatio-temporal state. Prior work either controls agents as cylinders by ignoring feet constraints, thus introducing visual artifacts, or use a small set of animations which limits the granularity of agent control. In this work we present a planner that given any set of animation clips outputs a sequence of footsteps to follow from an initial position to a goal such that it guarantees obstacle avoidance and correct spatio-temporal foot placement. We use a best-first search technique that dynamically repairs the output footstep trajectory based on changes in the environment. We show results of how the planner works in different dynamic scenarios with trade-offs between accuracy of the resulting paths and computational speed, which can be used to adjust the search parameters accordingly.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Animating groups of human characters in real time is a difficult but necessary task in many computer graphics applications, such as video games, training and immersive virtual environments. There is a large amount of work in the crowd simulation and pedestrian dynamics literature, but most applications still lack convincing character animation that offer a variety of animation styles without noticeable artifacts.

Humans walking in the real world have a cognitive map of the environment which they use for calculating their path through waypoints (doors, corners, etc), Then, we navigate along the path by choosing footsteps to avoid collisions with nearby humans and obstacles. Likewise, a virtual character can be simulated within an environment by first deciding a high level path (sequence of waypoints) using a navigation mesh [Mon09] [OP13] and then calculating the exact trajectory to walk from one waypoint to the next one. That trajectory is going to be defined by the chosen steering behavior algorithm, the output of which is going to encode the state of the agent over time. An agent state can be modeled by different granularities going from a simple point and radius with a velocity vector in a low level representation, to a complete high resolution mesh with joint velocity vectors, rotational angles, torques and any other elements that might improve the simulation on a higher level representation. Intermediate representations [SKRF11] can perform simulations in real-time by using an inverted pendulum model of the lower body of a biped which can be controlled to generate biomechanically plausible footstep trajectories.

This paper focuses on the computation of natural footsteps trajectories for groups of agents. Most work in the literature uses crowd simulation approaches (rules based models, social forces, cellular automata models, continuum forces) to calculate the root displacement between two consecutive waypoints. This leads to smooth root trajectories, but with many artifacts due to lack of constraints between the feet and the floor. There are some approaches that do focus on correct foot placement, but in most cases they are quite limited in the range of animations available or else can only deal with a small number of agents. Our work enforces foot placement constraints and uses motion capture data to produce natural animations, while still meeting real-time constraints for many interacting characters.

Figure 1 illustrates an example of four agents planning their footstep trajectory towards their goal while avoiding collision with other agents, and re-planning when necessary. The resulting trajectories not only respect ground contact

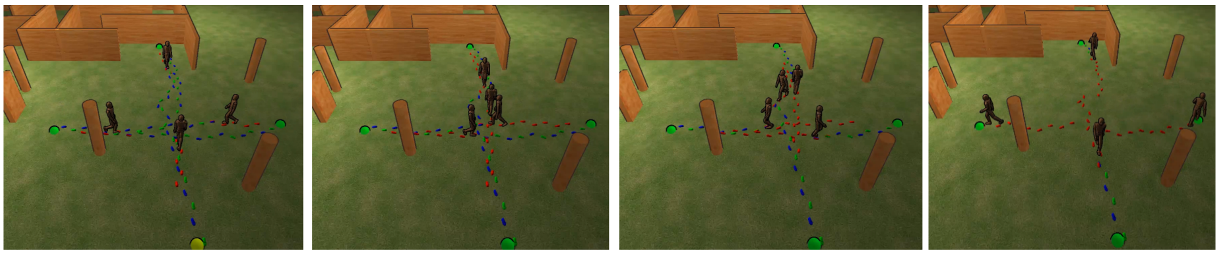

Figure 1: *Footstep trajectories planning for four agents reaching goals in opposite directions*

constraints, but also create more natural paths than traditional multi agent simulation methods.

This paper is organized as follows. We first examine previous approaches in crowd simulations and their methods. Next we give an overview of our framework and explain in detail our pre-process step, planning algorithm and animation system. Finally we show some of our results and present a discussion about the strength of our method and its limitations along with conclusions and future work.

2. Related Work

Crowd simulation approaches can be classified into two main sets based on whether they only focus on calculating the position of the root ignoring the animations, or whether they plan respecting the underlying animations. The first set focuses on simulating realistic behaviors regarding overall character navigation and do not worry about animations. In fact sometimes their goal is to simply model agents as cylinders that move around a virtual environment avoiding collisions. The second set, which carries out planning while being aware of the animation clips available, need to perform some pre-process to analyze the set of animation clips available to plan paths respecting constraints between the feet and the floor. In some cases, if the animation set is handmade, then the analysis is not necessary because the animations have already been built with specific parameters (such as speed, angle of movement and distance between feet) which are taken into consideration when planning.

The first group works with root velocities and forces or rules working on a continuous space, or displacements within a grid. Different models include social forces [HFV00], rule-based models [Rey87], cellular automata [TLCDC01], flow tiles [Che04], roadmaps [SAC*07], continuum dynamics [TCP06], local fields [KSHF09], hybrid methods [SKH*11], and forces models parameterized by psychological and geometrical rules [PAB07]. They can easily represent agents by discs or cylinders to illustrate their steering behavior, but do not care about a final representation using 3D animated characters, so the output trajectory needs to be used to synthesize an animation following it. Synthesizing the animation from a small database can cause artifacts such as foot-sliding that need additional work to be removed [PSB11].

The second group works directly with the set of available animations to construct motion graphs [KGP08, ZS09, RZS10, MC12], or precomputed search trees [LK06]. These approaches try to reach the goal by connecting motions to each other [WP95], sometimes limiting the movements of the agents. Other methods try to use motion graphs in the first group combining it with path planners [vBEG11]. Having a large animation database reduces the limitations in terms of freedom of movement, but also makes the planning more time consuming. The ideal solution would be one that could find a good trade-off between these two goals: freedom of movement and fast planning.

Some approaches have tried to change the simulation paradigm by using more complex agent representations, such as footsteps. They can be physically based but generated off-line [FM12]. Or they can be generated online from an input path computed by a path planner [EvB10], or planning them using an inverse pendulum model instead of root positions [SKRF11]. Recent work [KBG*13] proposes the use of multiple domains of control focusing searches in more complex domains, only when necessary. The resulting behavior offers better results giving characters a better interactivity with the environment and other agents, but they fall in the first group of our classification since they do not take animation into account and need another process to synthesize it.

Some locomotion controllers are able to synthesize in real-time animations according to velocity and orientation parameters [TLP07]. Other locomotion controllers can accurately follow a footstep trajectory by extracting and parameterizing the steps of a motion capture database [vBPE10]. However they all need a very large database and their computational time does not allow to have many characters in real-time.

Our work belongs to the second group of the classification, since it uses an animation-based path planner. However instead of pre-computing a search tree with a few handmade animation clips, we pre-process motion capture data (which allows us to have more natural looking animations and larger

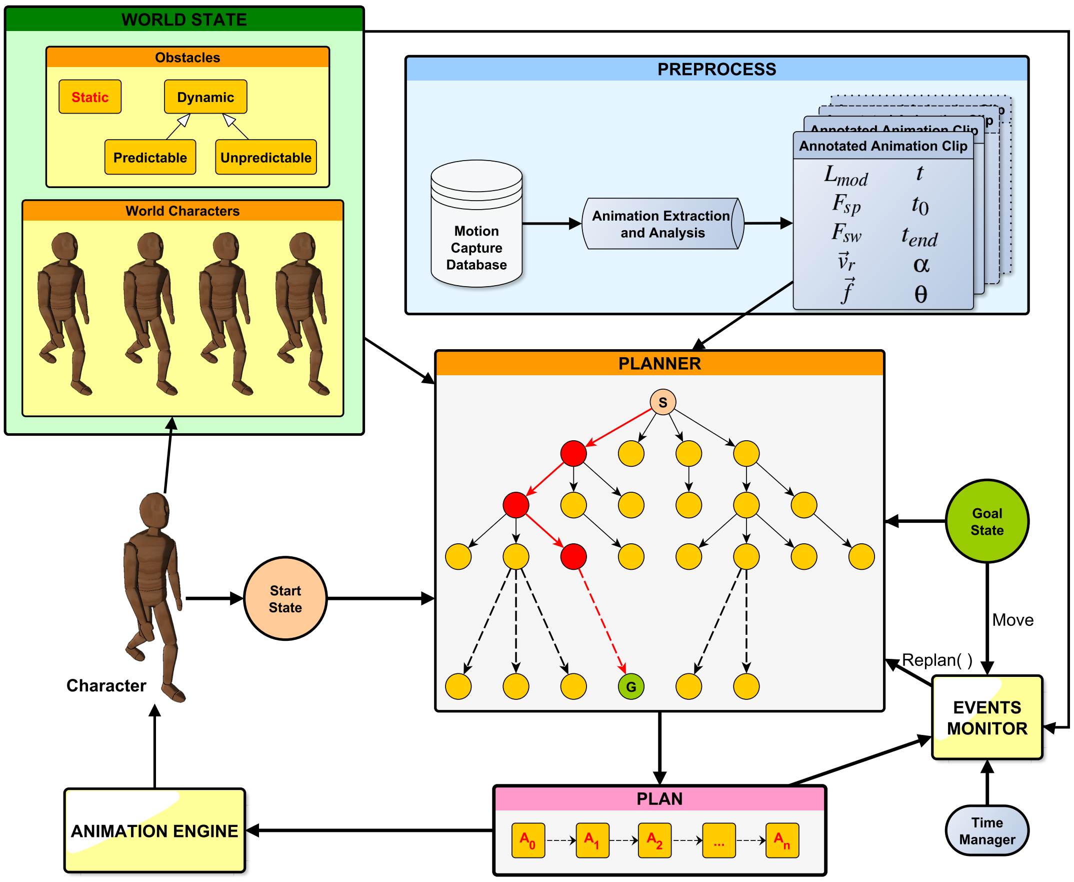

Figure 2: *Diagram showing the process required for the dynamic footstep planning algorithm*

variety), and extract actions from the input animations to compute a graph on the fly with an intelligent pruning based on logical transitions and a collision prediction system. Collisions are predicted and avoided for both static and deterministic dynamic obstacles, as well as for other agents since we expose all known trajectories.

3. Overview

Figure 2 illustrates the process of dynamic footstep planning for each character in real-time. The framework iterates over all characters in the simulation to calculate each individual foot step trajectory considering obstacles in the environment as well as other agents' calculated trajectories.

The *Preprocess* phase is responsible for extracting annotated animation clips from a motion capture database. The real-time *Planner* uses the annotated animations as transitions between state nodes in order to perform a path planning task to go from an input *Start State* to a *Goal State*. The output of the planner is a *Plan* consisting of a sequence of actions $A_0, A_1, \ldots, A_n$, which are clips that the *Animation Engine* must play in order to move the *Character* along the computed path. Both state and plan of the *Character* are then input to the *World State* and thus exposed to other agents' planners, together with the nearby static or dynamic obstacles. The *World State* is used to prune and accelerate the search in order to predict and avoid potential collisions. The *Time Manager* is responsible for checking the elapsed time between frames to keep track of the expiration time of the current plan. Finally the *Events Monitor* is in charge of detecting events that will force the planner to recompute a new path. The *Events Monitor* receives information from the *World State*, the *Time Manager*, *Goal State* and the character's current *Plan*. Events include: a possible invalid plan or the detection of a new dynamic obstacle or the goal position changing.

3.1. Events Monitor

The events monitor is the module of the system in charge of deciding when a new path needs to be recomputed. Elements that will trigger an event are:

- *Goal state changed*: when the goal changes its position or a new goal is assigned for the current character.
- *New agent or deterministic dynamic obstacle nearby*: other agents or dynamic obstacles enter the surrounding area of our character. A new path needs to be calculated to take into account the potential collision.
- *Collision against non-deterministic obstacle*: sometimes an unpredictable dynamic obstacle could lead to a collision (for example: a dynamic obstacle moved by the user), so when the events monitor detects such situation it triggers an event in order to react to it.
- *Plan expiration*: a way to ensure that each agent is taking into account the latest plans of every other agent is to give every plan an expiration time and force re-planning if this is reached. A time manager helps monitoring this task, but instead of a time parameter this event can also be measured and launched by a maximum number of actions that we want to perform (play) before re-planning.

4. Preprocess

During an offline stage, we analyze a set or a database of animation clips in order to extract the actions that our planner will then use as transitions between states. Each action consists of a sequence of skeleton configurations that perform a single animation step at a time, i.e., starting with one foot on the floor, until the other foot (swing foot) is completely resting on the floor. Our preprocess should work with any animation clip, since we tried both handmade and motion capture clips (from the CMU database [CMU13]). After analyzing each animation clip, we calculate mirrored animations. Mirroring animations is done in order to have each analyzed animation clip with either feet starting on the floor. The output of this stage is a set of annotated animations that can be used by the planner and the animation engine. This set can be easily serialized and stored to be reused for all instances of the same character type (same skeleton and the same scale, otherwise even if they share animations these could produce displacements of different magnitudes), reducing both preprocess time and the global memory consumption.

4.1. Locomotion Modes

In order to give our characters a wider variety and agility of movements we define different locomotion modes that need to be treated differently. Each animation clip will be tagged with its locomotion mode. We thus have the following set of locomotion modes:

- Walking: these are the main actions that will be used by the planner and the agents since they represent the most common way to move. We therefore have a wide variety of walks going from very slow to fast and in different angles (not just forward and backwards).
- Running: these are going to be treated in the same way as the walking actions with an additional cost penalty (since running consumes more energy than walking). We have also noticed empirically that for running actions it is not necessary to have as many different displacement angles as for walking actions.
- Turns: turns are going to be clips of animation where the agent turns in place or with a very small root displacement. They are going to be defined by their turning angle and velocity.
- Platform Actions: in this group we will find actions like jumping or crouching in order to avoid some obstacles. Such actions should have a high energy cost and should only be used in case of an imminent danger of collision.

While turns and platform actions need to be performed completely from start to end, and they do not have any intrinsic pattern we can easily detect, walking and running animations can be segmented by clips containing a single step. So animations of both walking and running locomotion modes will have a special treatment as we will need to extract the footsteps and keep only the frames of the animation covering a single step.

4.2. Footsteps Extraction

As previously mentioned in the paper, an action starts with one foot on the floor and ends when the other foot is planted on the floor. But animation clips, especially motion capture animations, do not always start and end in this very specific way. Therefore we need a foot plant extraction process to determine the beginning and end ending of each animation clip that will be used as an action.

Simply checking for the height of the feet in the motion capture data is not enough, since it usually contains noise and artifacts due to targeting. In most cases, when swinging the foot forwards while walking, the foot can come very close to the ground, or even traverse it.

Other techniques also incorporate the velocity of the foot during foot plant, which should be small. However this solution can also fail, since foot skating can introduce a large velocity. We detect foot plants using a height and velocity based detector similar to the method described in [vBE09], where foot plant detection is based on both height and time. First, the height-based test provides a set of foot plants, but only those where the foot plant occurs in a group of adjacent frames, are kept.

Our method combines this idea with changes on velocity for more accurate results, so we detect a foot plant when for a discretisized set of frames the foot is close to the ground for a few adjacent frames and with a change in velocity (deceleration, followed by being still for a few frames, and finishing with an acceleration). Notice that this method works for any kind of locomotion ranging from slow walking to running including turns in any direction.

4.3. Clip annotation

An analysis is performed by computing some variables over the whole duration of the animation. Each analyzed animation clip is annotated with the following information:

L_{mod}	Locomotion mode
F_{sp}	Supporting Foot
F_{sw}	Swing Foot
$\vec{v}_r$	Root velocity vector
$\vec{f}$	Foot displacement
t	Time duration
t_0	Initial time
t_{end}	End time
α	Movement angle
θ	Rotation angle
$\mathbb{P}$	Set of Sampled positions

Table 1: *Information stored in each annotated animation clip.*

Locomotion mode, indicates the type of animation (walk short step, walk long step, run, walk jump, climb, turn, etc). *Supporting foot* is the foot that is initially in contact with the floor, and the *swing foot* corresponds to the foot that is moving in the air towards the next footstep. The *supporting foot* is calculated automatically based on its height and velocity vector from frame to frame.

The *root velocity vector* indicates, taking the starting frame of the extracted clip as reference, the total local displacement vector of the root during the whole step. We therefore know the magnitude, the speed in m/s and the angle of its movement. Similarly, *foot displacement* tracks the movement of the swing foot.

Movement angle in degrees indicates the angle between the swing foot displacement vector and the initial root orientation. Therefore an angle equal to 0 means an action moving forward and 180 means it is a backward action. An Angle equal to 90 means an action moving to the left if the swing foot is the left one, or the right if the swing foot is the right one. Finally the *rotation angle* is the angle between the root orientation vector in the first and last frame of the clip.

t indicates the total time duration of the extracted clip, with t_0 and t_{end} storing the start and end point of the original animation that the extracted clip covers. These values will be used by the animation engine to play the extracted clip.

$\mathbb{P}$ corresponds to a set of sampled positions for certain joints of the character within an animation clip, and it is used for collision detection (see section 5.5)

5. Planning Footstep Trajectories

In this section, we first present the high level path planning on the navigation mesh. Then we define the problem domain we are dealing with when planning footsteps trajectories. Next we give details of the real-time search algorithm that we use as well as the pruning carried out to accelerate the search. Finally we explain how the collision detection and prediction is performed.

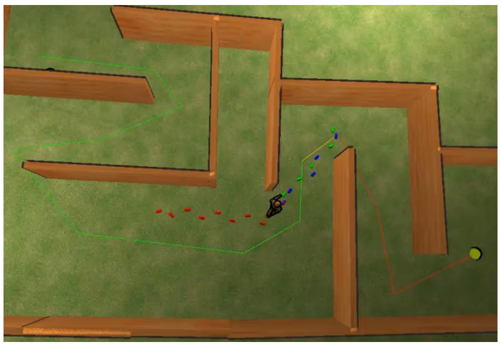

Figure 3: *High level path with local footstep trajectory between consecutive visible waypoints.*

5.1. High Level Path Planning

Footstep trajectories are calculated between waypoints of the high level path (see Figure 3). This path is calculated over the navigation mesh using Recast [Mon09]. An A* algorithm is used to compute the high level path, and then footstep trajectories are calculated between consecutive visible waypoints. So given a sequence of waypoints $\{w_i, w_{i+1}, w_{i+2}, ..., w_{i+n}\}$), if there is a collision-free straight line between w_i and w_{i+n}, then the footstep trajectory is calculated between those two waypoints, and any other intermediate point is ignored. This provides more natural trajectories as it avoids zig-zagging over unnecessary waypoints. Waypoints are considered by the planner as goal states, and each time that we change a waypoint the change of goal is detected by the events monitor, thus forcing a new path to be computed.

5.2. Problem Definition

The algorithm for planning footstep trajectories needs to calculate the sequence of actions that each agent needs to follow in order to go from their start position to their goal position. This means solving the problem of moving in a footstep domain between two given positions in a specific amount of time. Therefore, characters calculate the best trajectory based on their current state, the cost of moving to their destination and a given heuristic. The cost associated with each action is given by the bio-mechanical effort required to move (i.e: walking has a smaller cost than running, stopping for a few seconds may have a lower cost than wandering around a moving obstacle). The problem domain that we are dealing with is thus defined as:

$$\Omega = \left(\mathbb{S}, \mathbb{A}, c\left(s, s'\right), h(s, s_{goal})\right)$$

Where $\mathbb{S}$ is the state space and is defined as the set of states composed of the character's own state *self*, the world composition *environment*, and the *other agents* state. The action space $\mathbb{A}$ indicates the set of possible transitions in the state space and thus will have an impact on the branching factor of the planner. Each transition is an action, so we will have as many transitions as extracted clips times the possible speed variations we allow to introduce (we can for example reproduce a clip at half speed to obtain its displacement two times slower). Actions are then going to be defined by their corresponding annotated animation. $c(s,s')$ is the cost associated with moving from state s to state s'. Finally $h(s,s_{goal})$ is the heuristic function estimating the cost to go from s to s_{goal}.

5.3. Real-Time Planning Algorithm

Planning footsteps trajectories in real time requires finding a solution in the problem domain Ω described earlier. The planner solution consists of a sequence $A_0, A_1, \ldots, A_n$ of actions. Our planner interleaves planning with execution, because we want to be able to replan while consuming (playing) the action. For this purpose, we use a best-first search technique (e.g., A*) in the footstep problem domain, defined as follows:

- $\mathbb{S}$: the state space will be composed of the character's own state (defined by position, velocity, and the collision model chosen), the state of the other agents plus their plan, and the state and trajectory of the deterministic dynamic obstacles. For more details about collision models and obstacles avoidance see section 5.5.
- $\mathbb{A}$: the action space will consist of every possible action that can be concatenated with the current one without leading to a collision, so before adding an action we will perform all necessary collision checks.
- $c(s,s')$: the cost of going from one state to another will be given by the energy effort necessary to perform the animation:

$$c(s,s') = M \int_{t=0}^{t=T} e_s + e_w |v|^2 \, dt$$

where M is the agent mass, T is the total time of the animation or action being calculated, v the speed of the agent in the animation, and e_s and e_w are per agent constants (for an average human, $e_s = 2.23 \frac{J}{Kg.s}$ and $e_w = 1.26 \frac{J.s}{Kg.m^2}$) [KWRF11].
- $h(s,s_{goal})$: the heuristic to reach the goal comes from the optimal effort formulation:

$$h(s,s_{goal}) = 2Mc^{opt}(s,s_{goal})\sqrt{e_s e_w}$$

where $c^{opt}(s,s_{goal})$ is the cost of the optimal path to go from s to s_{goal}, in our case we chose the euclidian distance between s and s_{goal} [KWRF11]. The optimal effort for an agent in a scenario is defined as the energy consumed in taking the optimal route to the target while traveling at the average walking speed: $v_{av} = \sqrt{\frac{e_s}{e_w}} = 1.33 m/s$

Taking all these components into consideration the planner can search for the path with least cost and output the footstep position with their time marks that the animation engine will follow by playing the sequence of actions planned (see figure 4).

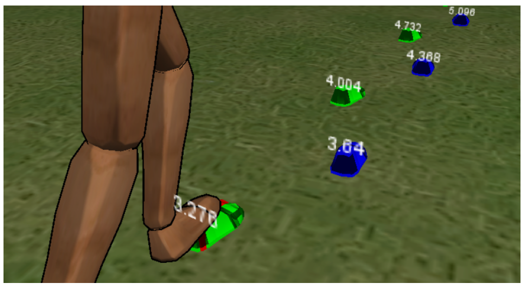

Figure 4: *Footsteps trajectory with time constraints that need to be followed by the animation controller.*

5.4. Pruning Rules

In order to accelerate the search we can add simple rules to help prune the tree and reduce the branching factor. A straight forward way to halve the size of the tree consists of considering only consecutive actions starting with the opposite foot. So given a current node with a supporting foot, expand the node only for transitions that have that same foot as the swing foot. Actions which are not possible due to locomotion constraints on speed or rate of turning are also pruned to ensure natural character motion (so after a staying still animation, we will not allow a fast running animation). The next pruning applied is based on collision prediction as we will see in the following section. The idea is that when a node is expanded and a collision is detected, the whole graph that could be expanded from it gets automatically pruned. The pruning process reduces the branching factor of the search, and also ensures natural footstep selection

5.5. Collision Prediction

While expanding nodes the planning algorithm must check for each expanded node whether the future state is collision free or not. If it is collision free, then it maintains that node and continues expanding it. Otherwise, it will be discarded. In order to have large simulations in complex environments we need to perform this pruning process in a very fast manner.

In order to predict collisions against other agents or obstacles (both dynamic or static), we introduce a multi-resolution collision detection scheme which performs collision checks for two resolution levels. Our lowest resolution collision detection model is a simple cylinder centered at the root of the agent with a fixed radius. The higher resolution model consists of five cylinders around the end joints (head, hands and feet) that are used to make finer collision tests Figure 5.

We could introduce more collision models, where high resolution ones will be executed only in case of detecting collisions using the coarser ones. At the highest complexity mode we could have the full mesh collision check, but for the purpose of our simulation the 5 cylinders model gives us enough precision to avoid agents walking with their arms intersecting against other agents as they swing back and forth. Compared against simpler approaches that only consider obstacle detection against a cylinder, our method gives better results since it allows us to have closer interactions between agents. All obstacles have simple colliders (boxes, spheres, capsules) to accelerate the collision checks by using a fast physics ray casting test.

It is also important to mention that collision tests are not only performed using the initial and end positions of the expanded node, but also with sub-sampled positions inside the animation (for the 5 cylinder positions). For example, an agent facing a thin wall as a start position and the other side of the wall as end position of its current walk forward step. If we only check for possible collisions with those start and end positions we would not detect that the agent is actually going through the wall.

The sub-sample for each animation is performed off-line and stored in the annotated animation. To save memory, this sampling is performed at low frequencies and then in real time intermediate positions can be estimated by linear interpolation.

Finally, we provide the characters with a surrounding view area to maintain a list of obstacles and agents that are potential threats to our path (see figure 6). For each agent, we are only interested in those obstacles/agents that fall within the view area in order to avoid running unnecessary collision tests.

5.5.1. Static World

Static obstacles are part of the same static world that is used to compute the navigation mesh with Recast [Mon09]. They do not need to have a special treatment since the high-level path produces waypoints that avoid collisions with static obstacles..

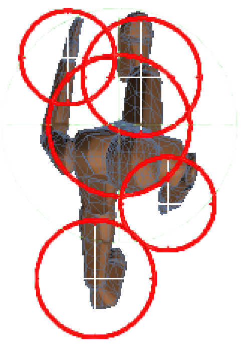

Figure 5: *Collision model of 5 cylinders around the head, the left and right hands, and the left and right foot.*

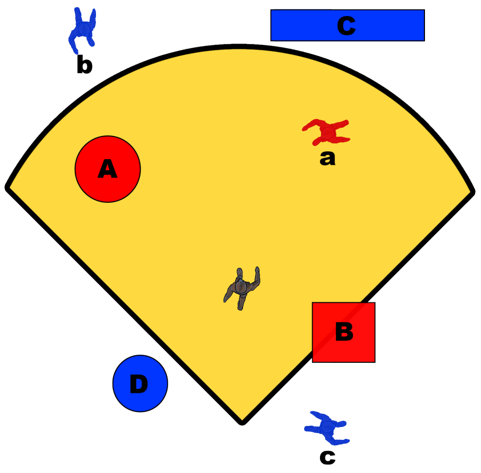

Figure 6: *When planning we only consider obstacles and agents that are inside the view area. Obstacles A, B and agent a are inside it and the agent will try to avoid them, while it will ignore obstacles C, D and agents b and c .*

5.5.2. Deterministic Dynamic Obstacles and Other Agents

Deterministic obstacles move with a predefined trajectory. Other agents have precomputed paths which can be queried to predict their future state. To avoid interfering with those paths we allow access to their temporal trajectories. So, for each expanded node with state time t we check for collisions with every obstacle and agent that falls inside his view area at their trajectory positions at time t. Figure 7 shows an example of an agent avoiding two dynamic obstacles.

5.5.3. Unpredictable Dynamic Obstacles

Unlike deterministic dynamic obstacles and other agents, unpredictable dynamic obstacles are impossible to be accounted for while planning. Therefore they can be ignored when expanding nodes, but we need a fast way to react to them. This is the reason why we need the events monitor to detect immediate collisions and force re-planning. Figure 8 shows an example where a wall is arbitrarily moved by the user and the agent needs to continuously re-plan its trajectory.

6. Animation Engine

The animation engine is in charge of playing the output sequence of actions given by the planner. These actions contain all the data in the annotated animation. When a new action is

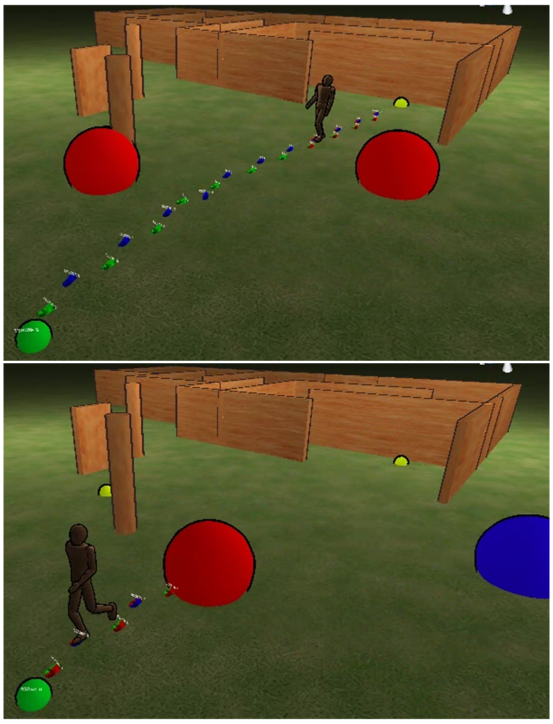

Figure 7: *An agent planning with two dynamic obstacles in front of him (top). After executing some steps the path is re-planned. The blue obstacle indicates that it is not in his nearby area anymore, so that obstacle is not considered in the collision check of this new plan. (bottom)*

played it sets t_0 as the initial time of the animation. When the current animation reaches t_{end} the animation engine blends the current animation with the next one in the queue.

The Animation Engine also tracks the global root position and orientation, and applied rotation corrections by rotating the whole character using the rotation values of the annotated animation (rotation angle θ). The blending time between actions can be user defined within a short time (for example $0.5s$).

7. Results

The presented framework has been implemented using the ADAPT simulation platform [SMKB13] which works with Unity Game Engine [Uni13] and C# scripts. Our current framework can simulate around 20 agents at approximately 59-164 frames per second (depends on the maximum planning time allowed), and 40 agents at 22-61 frames per second (INtel Core i7-2600k CPU @ 3.40GHz and 16GB RAM). Figure 9 shows the frame rates achieved on average for an increasing number of agents. The black line corresponds to a maximum planning time of 0.01s, and the red line corresponds to 0.05s. Additionally, by setting planner

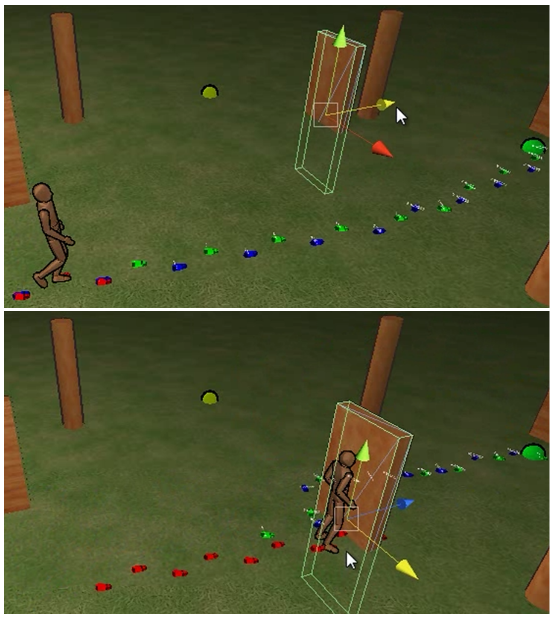

Figure 8: *An agent reacting to a non-deterministic obstacle by re-planning his path.*

parameters such as the horizon of the search, we can achieve significant speedup at the expense of solution fidelity. For example, we can produce purely reactive simulations where the character only plans one footstep ahead by reducing the search horizon to 1.

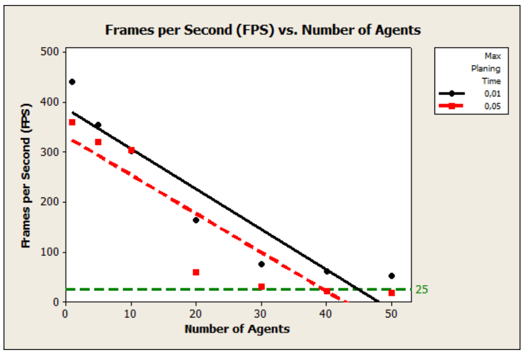

Figure 9: *This graph shows the frames per second on average for different simulations with increasing number of agents. We have used two values for the maximum planning time: 0.01 resulting in higher frame rates, and 0.05 resulting in lower frame rates but better quality paths*

The results showed have been made with a database of 28 motion captured animations. This is a small number compared to approaches based on motion graphs (generally having around 400 animation clips), but a large number compared with techniques based on handmade animation (such as pre-computed search trees). This decision allows us to achieve results that look natural and yet can be used for real time applications.

Our approach solves different scenarios where several agents are simulated in real-time achieving natural looking paths while avoiding other obstacles and characters (see accompanying videos). The quality of the results in terms of natural paths and collision avoidance depends on the planner. The planner will be given a specific amount of time to find a solution (which translates in how many nodes of the graph are expanded). Obviously when we allow larger search times (larger number of nodes to expand) the resulting trajectory looks more natural and is collision free, but at the expense of being more computationally expensive. Alternatively, if we drastically reduce the search time (smaller number of nodes to expand) we may end up having collisions as we can see in the resulting videos and in Figure 10.

Interleaving planning with execution provides smooth animations, since not all the characters plan their paths simulateneously. At any time, the new plan is calculated with the start position being the end position of the current action.

We have also shown how the Events Monitor can successfully plan routes when deterministic obstacle invalidate a character's plan, as well as efficiently react to non deterministic obstacles (see Figures 7 and 8)

8. Conclusions and Future Work

We have presented a multi-agent simulation approach where planning is done in the action space of available animations. Animation clips are analyzed and actions are extracted and annotated, in order to be used in real time to expand a search tree. Nodes are only expanded if they are collision free. To predict collisions we sample animations and use a new collision model with colliders for each end joint (head, hands and feet). This way we are able to simulate agents avoiding more detailed collisions. The presented framework handles both deterministic and non-deterministic obstacles, since the former can be taken into consideration when planning, while the later needs a completely reactive behavior.

Unlike pre-computed search trees our set of transitions is composed of actions, and mainly footsteps, which allows us to build online the search tree and to dynamically prune it, considering not only start and goal positions, but also departure and arrival times. An events monitor can help us to decide when to re-plan the path, based on the environment situation such as obstacle proximity or velocity.

We would like to further extend the hierarchical nature of this work to add granularity (both in models and domains) to adaptively switch between them [KCS10, Lac02, SG10]. Solutions from a coarser domain could also be reused to accelerate the search into a finer domain, using techniques such as tunneling [GCB*11]. Another idea would be to have a special class of actions constituting a reactive domain that would only be used in case of an imminent threat. Since non-deterministc obstacles invalidating the current plan force to replan constantly, it would be interesting to carry out a quantitative study on the impact of the number of non-deterministic obstacles in the frame rate obtained for different number of agents.

As Illustrated in 9, the computational complexity of our framework scales linearly with number of agents. By reducing the search depth and maximum planning time, we can simulate a larger crowd of characters at interactive rates. Choosing the optimal value of these parameters that balance computational speed and agent behavior is an interesting research direction, and the subject of future work. Our framework is not memory bound, and is amenable to parallelization with each agent planning on an independent thread.

Notice that memory is required per animation (to store sub-sampled animations) and not per agent in the simulation, therefore increasing the size of the simulated group of agents would not have an impact on the memory requirements of our system. If we wanted to simulate crowds of characters we would need more CPU power, but not memory as long as we had more instances of characters sharing the same skeleton and animations.

We would also like to improve our base search algorithm with a faster one taking into account repairing capacities such as ARA* [LGT03]. Having more characters and different sets of actions that can be used depending on the situation, like a reaction domain, would also accelerate the search and give better results to our simulations in constantly changing dynamic virtual environments.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation under Grant TIN2010-20590-C01-01. A. Beacco is also supported by the grant FPUAP2009-2195 (Spanish Ministry of Education). We would also like to acknowledge Francisco Garcia for his implementation of the Best First Search algorithm.

References

[Che04] CHENNEY S.: Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), Eurographics Association, pp. 233–242. 2

[CMU13] CMU: Cmu graphics lab motion capture database, 2013. http://mocap.cs.cmu.edu/. 4

[EvB10] EGGES A., VAN BASTEN B.: One step at a time: Animating virtual characters based on foot placement. *The Visual Computer 26*, 6-8 (apr 2010), 497–503. 2

[FM12] FELIS M., MOMBAUR K.: Using Optimal Control Methods to Generate Human Walking Motions. *Motion in Games* (2012), 197–207. 2

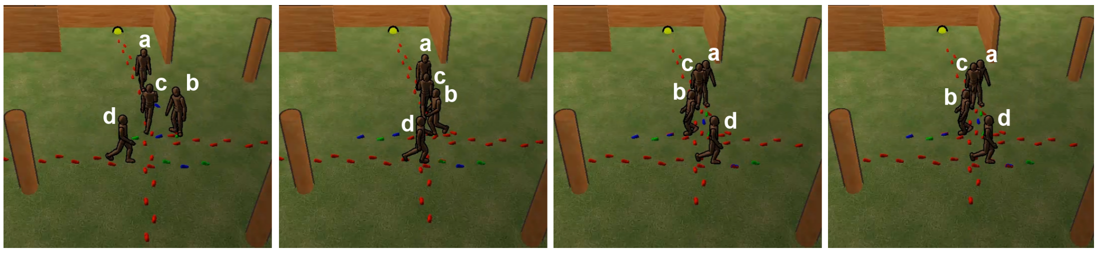

Figure 10: *Example with four agents crossing paths with a drastically reduced search time resulting in agents* ***a*** *and* ***c*** *not being able to avoid intersection as seen in the last two images of this secuence. Also notice how agent* ***b****, walks straight towards* ***c****, steps back and then continues, instead of following a smooth curve around* ***c****.*

[GCB*11] GOCHEV K., COHEN B., BUTZKE J., SAFONOVA A., LIKHACHEV M.: Path planning with adaptive dimensionality. In *Fourth Annual Symposium on Combinatorial Search* (2011). 9

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *Nature 407*, 6803 (2000), 487–490. 2

[KBG*13] KAPADIA M., BEACCO A., GARCIA F., REDDY V., PELECHANO N., BADLER N. I.: Multi-Domain Real-time Planning in Dynamic Environments. In *Proceedings of the 2013 ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation* (2013), SCA. 2

[KCS10] KRING A. W., CHAMPANDARD A. J., SAMARIN N.: Dhpa* and shpa*: Efficient hierarchical pathfinding in dynamic and static game worlds. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference* (2010). 9

[KGP08] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *ACM SIGGRAPH 2008 classes* (2008), ACM, p. 51. 2

[KSHF09] KAPADIA M., SINGH S., HEWLETT W., FALOUTSOS P.: Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 215–223. 2

[KWRF11] KAPADIA M., WANG M., REINMAN G., FALOUTSOS P.: Improved benchmarking for steering algorithms. In *Motion in Games*. Springer, 2011, pp. 266–277. 6

[Lac02] LACAZE A.: Hierarchical planning algorithms. In *AeroSense 2002* (2002), International Society for Optics and Photonics, pp. 320–331. 9

[LGT03] LIKHACHEV M., GORDON G., THRUN S.: Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems (NIPS) 16* (2003). 9

[LK06] LAU M., KUFFNER J. J.: Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2006), Eurographics Association, pp. 299–308. 2

[MC12] MIN J., CHAI J.: Motion Graphs++. *ACM Transactions on Graphics 31*, 6 (Nov. 2012), 1. 2

[Mon09] MONONEN M.: Recast navigation toolkit webpage, 2009. http://code.google.com/p/recastnavigation/. 1, 5, 7

[OP13] OLIVA R., PELECHANO N.: Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments. *Computer & Graphics 37*, 5 (2013), 403–412. 1

[PAB07] PELECHANO N., ALLBECK J. M., BADLER N. I.: Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), SCA '07, Eurographics Association, pp. 99–108. 2

[PSB11] PELECHANO N., SPANLANG B., BEACCO A.: Avatar locomotion in crowd simulation. In *International Conference on Computer Animation and Social Agents (CASA)* (Chengdu, China, 2011), vol. 10, pp. 13–19. 2

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics* (1987), vol. 21, ACM, pp. 25–34. 2

[RZS10] REN C., ZHAO L., SAFONOVA A.: Human Motion Synthesis with Optimization-Based Graphs. *Computer Graphics Forum (Proceedings of Eurographics 2010) 29*, 2 (2010). 2

[SAC*07] SUD A., ANDERSEN E., CURTIS S., LIN M., MANOCHA D.: Real-time path planning for virtual agents in dynamic environments. In *Virtual Reality Conference, 2007. VR'07. IEEE* (2007), IEEE, pp. 91–98. 2

[SG10] STURTEVANT N. R., GEISBERGER R.: A comparison of high-level approaches for speeding up pathfinding. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)* (2010), 76–82. 9

[SKH*11] SINGH S., KAPADIA M., HEWLETT B., REINMAN G., FALOUTSOS P.: A modular framework for adaptive agent-based steering. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 141–150 PAGE@9. 2

[SKRF11] SINGH S., KAPADIA M., REINMAN G., FALOUTSOS P.: Footstep Navigation for Dynamic Crowds. *Computer Animation And Virtual Worlds 22*, April (2011), 151–158. 1, 2

[SMKB13] SHOULSON A., MARSHAK N., KAPADIA M., BADLER N. I.: Adapt: the agent development and prototyping testbed. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2013), I3D '13, ACM, pp. 9–18. 8

[TCP06] TREUILLE A., COOPER S., POPOVIĆ Z.: Continuum crowds. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 1160–1168. 2

[TLCDC01] TECCHIA F., LOSCOS C., CONROY-DALTON R., CHRYSANTHOU Y.: Agent behaviour simulator (abs): A platform for urban behaviour development. 2

[TLP07] TREUILLE A., LEE Y., POPOVIĆ Z.: Near-optimal character animation with continuous control. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 7. 2

[Uni13] UNITY: Unity - game engine, 2013. http://unity3d.com/. 8

[vBE09] VAN BASTEN B. J. H., EGGES A.: Evaluating distance metrics for animation blending. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (New York, NY, USA, 2009), FDG '09, ACM, pp. 199–206. 4

[vBEG11] VAN BASTEN B., EGGES A., GERAERTS R.: Combining Path Planners and Motion Graphs. *Computer Animation and Virtual Worlds 22*, 1 (2011), 59–78. 2

[vBPE10] VAN BASTEN B. J. H., PEETERS P. W. A. M., EGGES A.: The step space: example-based footprint-driven motion synthesis. *Computer Animation and Virtual Worlds 21*, 3-4 (May 2010), 433–441. 2

[WP95] WITKIN A., POPOVIC Z.: Motion warping. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM, pp. 105–108. 2

[ZS09] ZHAO L., SAFONOVA A.: Achieving good connectivity in motion graphs. *Graphical Models 71*, 4 (2009), 139–152. 2