Procedural Locomotion of Multi-Legged Characters in Dynamic Environments

Ahmad Abdul Karim[1] [2], Thibaut Gaudin[2], Alexandre Meyer[1], Axel Buendia[2] [3], Saida Bouakaz[1]

[1] Université de Lyon, CNRS

[1] Université Lyon 1, LIRIS, UMR5205, F-69622, France

[2] Spir.Ops Artificial Intelligence, Paris, France

[3] CNAM−CEDRIC, 292, rue St Martin, 75003 Paris, France

Abstract

We present a fully procedural method capable of generating in real-time a wide range of locomotion for multi-legged characters in a dynamic environment, without using any motion data. The system consists of several independent blocks: a Character Controller, a Gait/Tempo Manager, a 3D Path Constructor and a Footprints Planner. The four modules work cooperatively to calculate in real-time the footprints and the 3D trajectories of the feet and the pelvis. Our system can animate dozens of creatures using dedicated level of details (LOD) techniques, and is totally controllable allowing the user to design a multitude of locomotion styles through a user-friendly interface. The result is a complete lower body animation which is sufficient for most of the chosen multi-legged characters: arachnids, insects, imaginary n-legged robots, *etc.*

Keywords: Procedural Animation, Level Of Details, Multi-Legged Characters, Dynamic Environments.

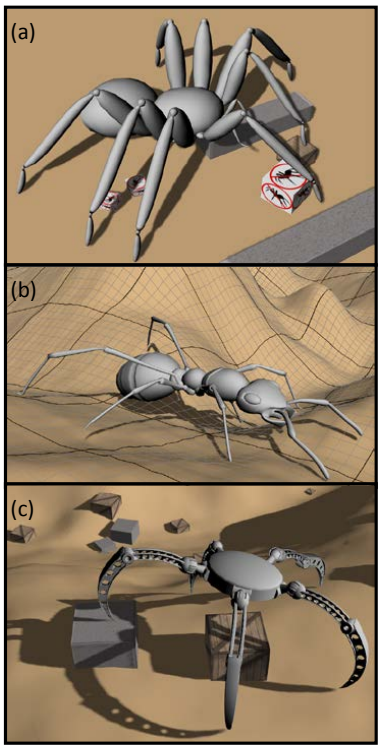

Figure 1: *(a) Spider model avoids crates flagged as forbidden and steps on others flagged as safe (b) Ant model on a height map (c) Imaginary 5-legged Robot.*

1 Introduction

Real or imaginary animals make frequent appearances in video games, films and virtual world simulations. In this context, animated virtual multi-legged characters like arachnids, insects, crustaceans or any imaginary n-legged robots/creatures make these virtual worlds believable and life like. The most common task that these virtual multi-legged characters perform is locomotion: the ability to move in these virtual worlds toward the points of interest. These locomotion animations are quite rich due to the variety of morphologies, gaits, body sizes/proportions and due to the complex-

ity of the environment where these multi-legged characters move: overcoming objects, crossing uneven terrain, avoiding moving obstacles, *etc.* The challenge of modeling such motions arises from the lack of motion capture data inherent to the difficulty to obtain them and by the fact that existing animation techniques mostly address human-like characters [1].

We propose a system (See Figure 1) that procedurally generates locomotion animations for dozens of multi-legged characters, without any motion data, in real-time using dedicated level of details (LOD) techniques, and with a reactive adaptation to a dynamic environment. We focused on generating plausible movements [2] that are, at the same time, believable and fully controllable. Our technique produces a multitude of locomotion styles, generated out of many input parameters like the creature morphology, gait/tempo, the overall locomotion speed, *etc.* These input parameters can be edited through a user-friendly interface in real-time. We propose a foot path planning system that assign to each foot the best couple (footprint-trajectory), where the footprint is selected among several candidates, and the trajectory is the 3D path that reach this footprint. Our system generates plausible locomotion animation at different level of details (LOD): the evaluation of the possible couples (footprint-trajectory) may be adapted according to the desired LOD, typically depending on quality criteria such as the on-screen importance, visibility, *etc.* These footprints and trajectories may change in real-time in a reactive manner according to the dynamic of the environment and the obstacles. Finally, our system generates the pelvis trajectory according to the overall desired direction, speed, the actual context of the environment and the feet positions and feedbacks. Thus, our procedural approach provides two important features for interactive applications: reactivity and control.

2 Related Work

There are several techniques for generating locomotion, Multon *et al.* in [3] and Van Welbergen *et al.* in [1] identified in their surveys the three major techniques: data-driven, physics-based and procedural. Notice that few of the presented approaches were dedicated or tested on the family of animals and creatures presented in this paper, and to our knowledge, none addresses the performance issues with level of details techniques introduced in our system.

Data-driven techniques use input data to produce the animation (like Motion Capture Data). These techniques, although the most natural looking, are fixed for a specific morphology and a specific context. **Physics-based techniques** are more generic and adapt better to the environment. They do so by simulating the actual physical forces/torques that act on the articulated body. But they are computationally expensive due to the number of equations to solve, which limits the number of characters simulated at the same time. Several works were proposed to overcome the problems of the previous techniques or to mix them, like: Adapting the motion data to different morphologies (Motion Retargeting) using inverse kinematics in [4, 5] or mesh based techniques in [6]. Using inverted pendulum model (IPM) [7] to avoid complex kinematics and dynamics calculations on the original multi-segments skeleton. In [8] they animate the upper body of the character using Motion Capture data while the lower body is animated using an IPM. In [9] they optimize their IPM controller using Motion Capture data. In [10] they animate a quadruped in real-time using forward dynamics, using forces data extracted out of real quadruped Motion Capture data. Finally, several papers introduce a specific offline optimization pass, to accommodate for predefined environmental topologies and changes [11, 12]. The lack of motion data and the difficulties to obtain them for the family of animals and creatures that we simulate has led us to avoid data-driven techniques. In our system we concentrated more on generating plausible movements [2], making it more adapted (performance wise) to real-time applications than the physics-based ones.

Procedural techniques use mathematical formulas and algorithms to generate the locomotion without any real motion data. They do so using empirical and biomechanics concepts. These techniques offer high level of controllability over data-driven ones: practically no morphological constraints, good adaptability to the environment, *etc.* These techniques take

less computation time per character, making it possible of animating several characters at the same time in real-time. However, they may suffer from the lack of naturality in the movement, a disadvantage that we try to overcome in this paper by integrating basic biomechanics principles. We took inspiration from the work of Boulic *et al.* in [13] on a human walking model and the work of Singh *et al.* [14] that show how avoidance in dynamic crowds can be improved using biomechanics-based footstep planning.

In parallel to the locomotion animation, a path and footprints planning are needed in order to navigate through the environment. In [15] they search valid footprints using probabilistic navigation graphs that take into account posture transition. Then, these footprints are substituted with corresponding motion clips. Lamarche in [16] generates navigation graphs and ceiling information and uses these information to calculate the best footprints around the optimized trajectory, and to generate the final motion using motion data, allowing for 3D navigation in complex environments. Path planning techniques are also largely studied in robotics. In [17, 18, 19] (H7 Humanoid Robot™ and Boston BigDog™) their planner produce the footprints using feedforward loops, correcting itself after each step for adaptation. In [20] (Honda ASIMO™) they plans a sequence of footstep positions to navigate toward a goal location while avoiding static and dynamic obstacles, their method uses a time-limited planning horizon. Comparing to our purpose of computer animation, robotics have to deal with the complexity related to the mobile robot mechanical constraints and with the real world physics that we do not fully have.

Most of previously cited techniques are **Foot placement driven locomotion**, where the feet drive the locomotion and the overall trajectory of the center of mass. In [21] they generate locomotion by overlaying Motion Capture data on predefined foot placement, while Torkos *et al.* in [22] generate the locomotion of a quadruped using external imposed footprints, and real-time physics. These techniques suffer from the problem that the lower body (the legs) usually do not control the locomotion. On the contrary, the upper body imposes a logical trajectory that the lower body tries to follow [23]. Our system respects this concept: the pelvis adapt its movement based on the feet feedback, but at the same time it imposes the overall trajectory.

When considering locomotion of multi-legged animals, the possibility of using real data are strongly limited due to difficulties to capture the motion of real creatures like dogs, horses, insects, spiders, *etc.* and is sometime impossible like for an imaginary 5-legged character. Thus, Girard *et al.* in [24, 25] propose a fully procedural system to animate multi-legged characters with visually plausible locomotion. Their system is limited to planner terrains. Our work can be considered as an extension into dynamic environments. Johansen in [26] adapts existing motion data (biped and quadruped) to the environment using inverse kinematics and footprints prediction. In [27] they animate a six-legged character (a cockroach) using reduced-linear articulated body dynamics and gait patterns observed in biology. Wampler *et al.* in [28] present a fully automatic method for generating locomotion and gaits for legged animals based on their shape and by the use of an off-line optimization pass that results in believable animations. In [29] they use intelligent retargetting of 2-4-6-legged Motion Capture data without any morphological constraint (the user can design their own creature). The objective of our system is to achieve similar results with the procedural techniques in real-time without using any off-line components, and with much more control in a dynamic complex environment.

Finally, as our method tries to generate plausible and believable locomotion, we created our controllers in a way that reflects many studies: the effect of limb length and running speed on time of contact and step length in [30], the relationship between speed and step length in humans in [31], and animal gaits: galloping, trotting, pacing, *etc.* in [32].

3 Overall Locomotion Controller

Locomotion is the act of moving from one place to another. For most terrestrial animals that means putting one foot in front of the others in a successive way until reaching the designated point of interest (target). During a normal foot movement there are two main phases explained more in [33]: the stance phase where the foot is blocked on the ground and the swing phase

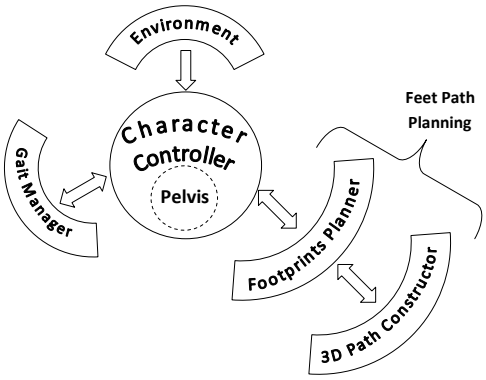

Figure 2: *System Overview*

where the foot flies in a parabolic-like curve toward its target without any ground contact. Locomotion cycle is the act of repeating these feet movements, based on a certain rhythm or tempo. Our system follows these principles when generating -procedurally and in real-time- the locomotion of the multi-legged characters. The main advantage of the procedural approach is the reactivity as it adapts to the environment, and the controllability (a list of all controllable parameters is detailed in Section 3.1). Starting from an environment defined by a height map and several obstacles (moving or static), the overall locomotion process is computed by four blocks as shown on Figure 2.

- The character controller is the central main structure that manages the overall locomotion of the multi-legged character (See Section 3.2). It relies on the three other structures to compute the motion of the feet and the overall pelvis displacement.
- The gait manager regulates the feet tempo according to the movement patterns defined by the user (See Section 3.3).
- The 3D path constructor is a utility structure that construct a 3D trajectory of a foot using an efficient discrete representation of the environment that can be easily maintained and updated with dynamic objects (See Section 4).
- The footprints planner evaluates for each foot all the possible targets and trajectories in real-time, and chooses the best couple of footprint target and 3D trajectory (See Section 5).

These four blocks work together in order to generate the movement of the feet and the pelvis in 3D. Now, to obtain the final lower body locomotion movement we apply an inverse kinematics (IK) algorithm. We use a system called the Cyclic Coordinate Descent (CCD) algorithm [34, 35] which computes the position and orientation of the legs joints that connect the pelvis with its feet. We chose the CCD because of its performance and simplicity: it minimizes the error between the actual end effector position and the desired one by iteratively modifying the angle of the joints.

3.1 System Parameters

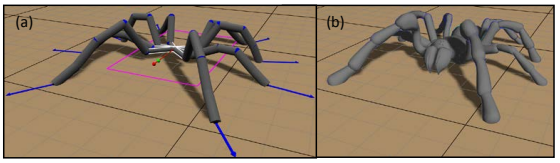

Figure 3: *Spider Model: (a) internal representation of the spider model with its IK system, in pink is the projected bounding box, in blue the feet orientation. (b) the actual spider 3D model.*

Our system generates the locomotion animations using several input parameters, that the user provides or controls in real-time. All the parameters (except the morphology parameters) can be edited at any time by the user or by any automatic character controller when the focus is to animate a crowd of creatures. An overview of all the parameters follows.

- **Multi-legged character morphology**: the user provides an initial static skeleton that can be associated with a skinned mesh. To automatically map the morphology, the user provides the system with the name of each leg hip joint and the name of its end effector (*e.g.* the joint before the foot). Out of these inputs our system detects the number of feet, leg sections, the relative hip positions and the initial feet relative positions. The user also needs to provide the system with the desired leg joint limits (for our CCD algorithm), real body center, projected body bounding box, body thickness, and finally the projected foot shape (if it exists) an example is shown on Figure 3.
- **Gait/Tempo**: using our interface, the user designs for each foot the stance and swing

phase cycle. These cycles describe the pattern of the feet movement. The final gait can be symmetrical or asymmetrical.

- **Locomotion speed**: speed of the movement in $meter\ per\ second$.
- **Step height**: the preferred foot step height in $meter$.
- **Feet spacing**: the preferred position of each foot, relative to the pelvis.

3.2 Character Controller

The character controller is the coordinator of the overall system. It is in charge of two main tasks: managing the movement of the feet and computing the pelvis 3D movement based on the user needs and on the final positions of the feet.

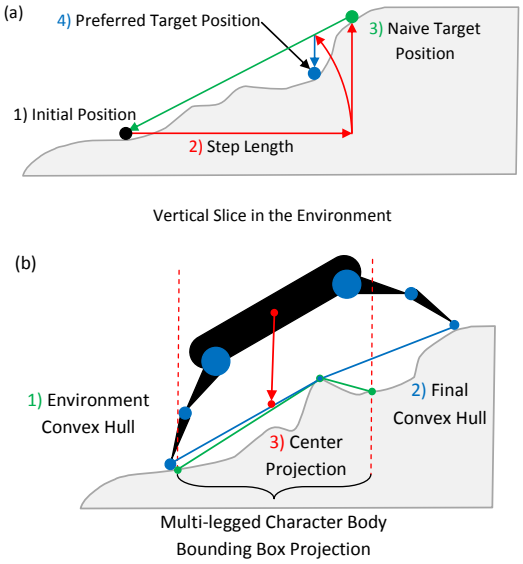

Figure 4: *The vertical slice in the environment can contain objects and obstacles, (a) computation of the preferred footprint target for each foot. The naive target position is calculated by querying the environment about the elevation of the 2D point resulted from the initial position + the step length (b) Computation of the pelvis' height using several convex hulls that covers the environment slice that interests us*

Pelvis default movement: the 2D movement of the pelvis on the ZX plan (assuming the Y-axis is up) is calculated based on the speed and orientation. While the computation of the pelvis height is more complex, as shown on Figure 4(b): we first construct a convex hull based on the ground height underneath it and the actual position of the feet. By projecting the multi-legged character center on this convex hull we get the needed pelvis height and by adding the body thickness we get the actual pelvis height. At the same time the character controller adds an oscillating (sinusoidal) movement on this pelvis height, based on the feet phases. The pitch angle of the multi-legged character body is calculated directly from the convex hull. It is the slope of the line segment that contains the projected center of the multi-legged character.

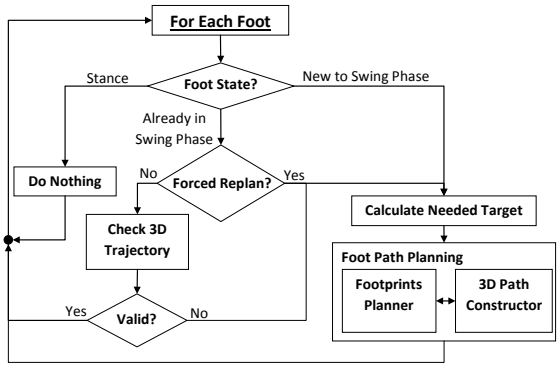

Figure 5: *Main job of the character controller per foot: based on the current foot state it can take different decisions*

Feet movement: Figure 5 explains how the character controller manages the movement of the feet (first task). At each time step, the gait manager informs the character controller about the feet that are going to enter swing phase as detailed in Section 3.3. For each of these feet a preferred footprint target is calculated. As shown on Figure 4(a), the foot step length is calculated based on the current speed, multi-legged character morphology and the foot relative position that the user can impose as illustrated and explained directly on Figure 6. After calculating this foot step length, we use the initial position of the foot to calculate the preferred 3D target position in the environment (See Figure 4(a)).

Finally, the character controller calls the foot path planner to compute the 3D trajectory that this foot will follow, as described in Section 5. Feet that were already in swing phase may need a new a 3D trajectory (path) for several reasons: a new pelvis orientation (when turning), a new desired relative position of the foot (Figure 6), a new overall speed, or something changed in the environment thus invalidating its current 3D trajectory or its current target. In these cases the character controller processes this foot as if it

just started its swing phase, and therefore it is redirected to the previously explained foot path planning phase. Stance feet are blocked on the ground.

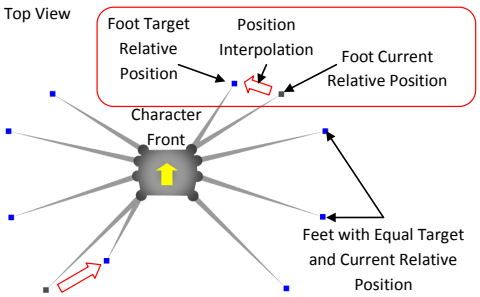

Figure 6: *8-Legged Character Feet Spacing Interface*

Pelvis movement refining: After moving the feet, the character controller refines the pelvis movement based on the feet feedback (shown on Figure 7). Each foot can choose a better target that adapts better to the context (see Section 5). In that case, the character controller averages the offset between the preferred footprint targets (Figure 7 in blue) and the effectively chosen one (in green). The controller incorporates this offset when it is quite significant, based on the multi-legged character morphology, otherwise it discards it.

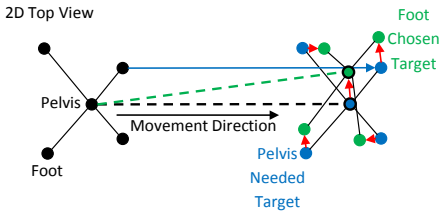

Figure 7: *Feet feedback: During the computation of the pelvis displacement, we use the feet feedback.*

3.3 Gait Manager

The role of the gait manager is to organize and visualize the pattern of the feet's cycle and to perform the transition between patterns. Since a locomotion is cyclic, it seemed natural to represent the gait/tempo with circles. As illustrated on Figure 8, each circle represents a foot, with the colored sectors representing the swing phase portion of the foot movement. The feet needle activates sectors and deactivates others based on its current position while turning. Activating a sector means that the corresponding foot should enter in the swing phase.

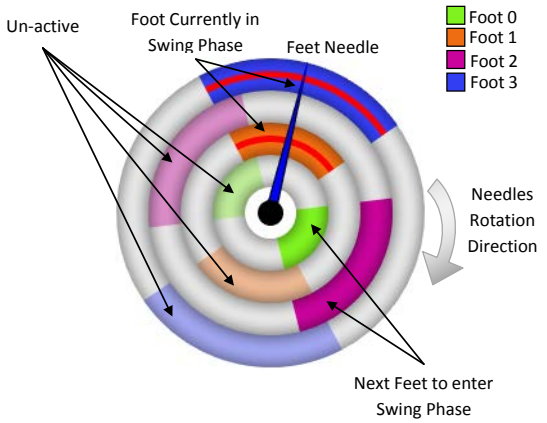

Figure 8: *An example of a four feet's gait as shown by the gait manager. With this interface, the user can edit the pattern.*

Gaits interpolation/transition During a locomotion cycle any creature changes its gait constantly in order to adapt to the environment. To accommodate for this changes in the gait and to introduce more variety in the locomotion styles, our gait manager allows the transition/interpolation between any needed gaits on the fly. Each swing phase sector has a start and a finish, we compute the transition to another (destination) gait by simply interpolating the start of the source sector toward the start of the destination sector in the direction that does not pass by the finish of the destination sector. On Figure 9(a) we chose the clockwise direction while on Figure 9(b) we chose the counterclockwise one. By doing so, the interpolation does not pass by any area of the circle that the user does not want to. The interpolation of the duration of the swing phase (the size of the sector) is quite straightforward, and the speed of all these processes can be fixed in advance by the user. So, on each step and when the foot is not active (in stance phase), we replace its disk with the new disk calculated in the interpolation process, resulting in seamless and logical interpolation.

4 3D Path Construction

During the swing phase, each foot has a current position (source) and a desired footprint target. The role of the 3D path construction is to construct the foot 3D trajectory that navigates through the environment, from this current position toward the footprint target without colliding with any obstacle. This 3D path construction is

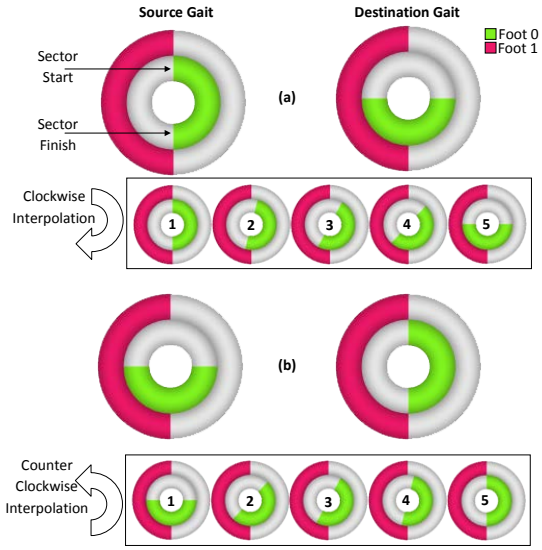

Figure 9: *Interpolations Options: (a) deciding to do a clockwise position interpolation based on the actual disposition of the gait disks (b) deciding to do a counter-clockwise position interpolation*

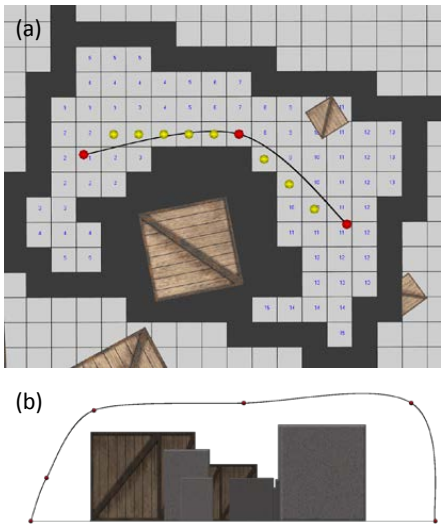

Figure 10: *(a) We firstly construct the path in a 2D voxel-based representation of the environment (yellow dots) and then generate the final curve (in black) with a selection of waypoints (red dots). (b) Using the elevations map and the previous 2D trajectory we construct the 3D trajectory.*

requested several times with different footprint targets by the foot path planner as it will be explained in Section 5. The environment itself can be very complex, and it can include several objects represented by many triangles. That is why we have oriented this path construction process toward a discrete grid-based approach. Indeed, once the environment is converted and represented by our grid, the path computation becomes independent of the object's complexity. We also have chosen this over a vectorial based path planning because discrete algorithms are easier to implement.

Grid-based representation of the environment. We convert the 3D environment near the animated multi-legged character into two 2D grids: the *obstacles' map* and the *elevations map*. The *obstacles' map* describes the areas of the environment where the feet are allowed to pass, as illustrated on Figure 10(a). The black cells represent the obstacles, and we call them the forbidden cells. The *elevations map* contains the elevation of the highest obstacle in each cell as shown on Figure 11. These two maps are computed using the terrain heightmap and the objects: the bounding box of each object is voxelized onto the maps. Concave objects are subdivided into convex ones. We pre-compute the map's representation of static objects. While we compute the map's representation of dynamic objects when they move near our multi-legged character. In order to avoid legs crossing, we add for each foot the projection of the other legs into the *obstacles' map* as forbidden cells.

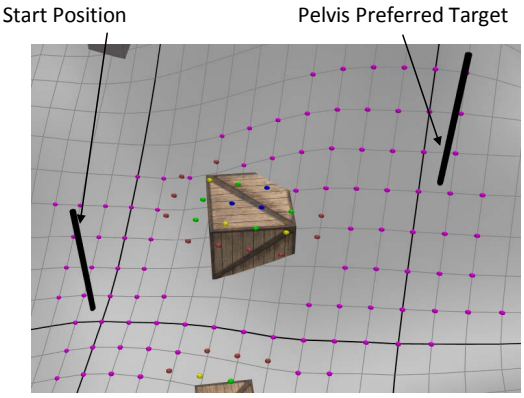

Figure 11: *An example of our environment discretization: pink spheres represents the ground, yellow spheres contain the corner of an obstacle, green spheres contain an obstacle edge, blue spheres are an obstacle interior and red spheres are an obstacle size increase.*

These maps are relatively small as they are only computed around our animated multi-legged character, speeding up calculations. Since a foot could not be punctual, we use the bounding box of the foot shape to increase the size of the obstacles in the opposite direction

of the foot, as shown on Figure 12. Doing so guarantees the non penetration of the foot with the obstacles.

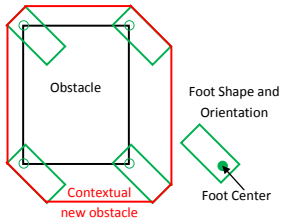

Figure 12: *We increase each obstacle size using the bounding box of the foot and its orientation in order to better avoid collision.*

Path construction. Always with a performance concern, we firstly compute the trajectory from the actual foot position toward the targeted footprint in the ZX plan using the projections of the source and the target on the ZX plan (in black on Figure 11). Our system uses a shortest path computation that processes a Wavefront algorithm combined with the potential field method [36] for the obstacle avoidance. This method is quite similar to the well-known Dijkstra's shortest path algorithm.

The resulting path off the wavefront algorithm is constructed out of several cells that we call the naive cells of the plan (in yellow on Figure 10). We refine these naive cells, using queries of line of sight (no intersection with a forbidden cell) to eliminate unwanted cells, and we obtain the final cells of the the plan (in red). These cells are used to build the 2D path represented by a parametric Hermite curve as shown on Figure 10)(a).

We sample this 2D curve and elevate it in 3D using the *elevations map*. Resulting waypoints are used as control points to define a 3D Hermite curve: this curve represents our 3D trajectory through the environment as shown on Figure 10(b).

5 Footprints and Feet Path Planning

In Section 3.2, we explained that starting from the initial foot position the character controller calculates a preferred footprint target according to the locomotion parameters and the surrounding environment (Figure 11). But these calculations do not take into consideration the actual state of the foot and its preferences (*e.g.* the preferred target calculated by the character controller can be too close to an obstacle). That is why in this step we calculate and assign to each foot the best trajectory toward the best target in the current environment.

5.1 Potential Footprints

Our algorithm evaluates several footprint targets by exploring the potential cells around the preferred footprint (the one computed by the character controller in Figure 11). Several operations are done to do that. In both maps (the *obstacles' map* and the *elevations map*), we firstly increase the size of the obstacles using the foot shape (See Figure 12 and Figure 11 in red). Secondly, to eliminate possible intersection with other feet we fill the *obstacles' map* with the position of these other feet. Thirdly, we identify all the -potential- cells that may accept a footprint: all the non-forbidden cells that are not a size increase ones (Figure 13). For each potential target, we calculate a score based on several criteria's: distance to the preferred footprint, difference of elevation between the footprint and the surrounding cells, leading or not to a feet crossing, *etc.* The combination of all these criteria's provides the final cell's score (footprint score) that we normalize between 0 and 1. Figure 13 shows these scores.

5.2 Best Pair of Footprint and Path

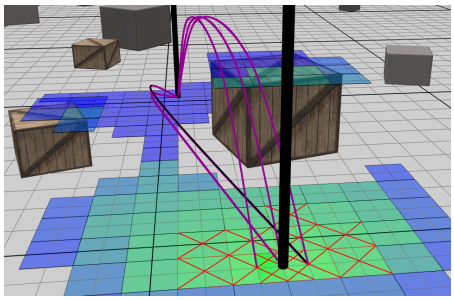

Figure 13: *Potential targets: the color varies from green (best target) ($score = 1$) to blue (worst target) ($score = 0$). Crossed cells are the cells processed by our algorithm, in pink possible trajectories, in black the chosen one.*

Avoidance-wise. During the 3D path construction (See Section 4), the forbidden cells in the *obstacles' map* designate the parts of the environment that the 2D trajectory is going to avoid and go around. This 2D trajectory passes through cells that have an elevation. The 3D path constructor uses this elevation to construct

the final 3D trajectory. The system uses this *obstacles' map* to process an obstacle: either going around it or going over it. If the system wants to go around an obstacle, it just needs to add it to the *obstacles' map* as forbidden cells. In that way, the resulting 2D trajectory will definitely go around it. While if the system wants to go over an obstacle, it does not need to do anything special as the obstacle is already present in the *elevations map*. And the obstacle will be avoided accordingly based on its elevation. In order to achieve this avoidance distinction in an optimal way in a complex environment (height map plus obstacles), our system discretize the *elevations map* on the Y-axis into slices (for instance slices of $10cm$). For each slice, the system fills the *obstacles' map* with the cells that have higher elevation than this slice. In that way, all calculated trajectories will go over all obstacles, objects and pieces of the environment that have an elevation lower or equal to the slice's elevation. At the same time, all calculated trajectories will go around all obstacles, objects and pieces of the environment that have an elevation higher than the chosen slice's elevation.

So each trajectory can go around or over each obstacle, which generates multiple trajectories toward a target. Each one of these trajectories has a score. Our search space is all the possible trajectories that go from the starting point toward all the possible targets. Out of this multitude of possibilities, our main algorithm picks up the best couple (target-trajectory) in an intelligent way.

Our algorithm (result shown on Figure 13) loops on all the possible targets based on their score (in a descending way). For each one of these targets, the algorithm first generates the 2D trajectory and scores it, if the score of the couple (target-trajectory in 2D) is better than the current best couple found till now, it continues. Then it generates the 3D trajectory based on the 2D one and scores it, if the score of the new couple (target-trajectory in 3D) is better than the current best couple, then it tags this new couple as the best one and continues. The algorithm continues evaluating the couples until it reaches a couple (target-trajectory in 3D) with a score worse than the current best one, in this case it stops and the current best couple is the best one. So, at the end of the algorithm we obtain the best footprint target and 3D trajectory that this foot should follow. We calculate the length of this 3D trajectory, and using the needed time given by the character controller, we move the foot on the curve at a constant speed.

Trajectory scoring. We generate for each created 2D/3D trajectory a score based on the application specifications, like the total length of the path, the acceleration and the curve tangents profile, *etc.* For instance, the system shown in the accompanying video prefers trajectories that are more straight (direct) with less curvature, a preference observed in biomechanics as it minimizes the energy cost [37, 38].

6 Level of Details techniques (LOD)

Our main algorithm (Section 5.2) execution time can be controlled easily when simulating many multi-legged characters in order to accelerate the simulation itself. This comes from the ability of controlling the number of evaluated couples (target-trajectory) and from the ability of controlling the size of the slices used when avoiding obstacles. But in order to always have a plausible locomotion, we start applying this LOD after we find the first valid couple (target-trajectory), in this way we ensure that the algorithm will assign to each foot a valid trajectory toward a valid target (although it is not the best couple).

In our simulation and for off-screen characters, the algorithm stops directly when finding this first couple. While for the on-screen characters, we limit the number of couples evaluated after finding this first couple in a linear way based on the distance of that multi-legged character from the camera. In this way for far on-screen characters the algorithm stops also when it finds the first couple. In the same way, for off-screen characters the algorithm tries to go over any obstacle using no slices (not going around any obstacle). While for the on-screen characters, we increase the size of the slices (for example $10cm$, $15cm$...no slices) in a linear way based on the distance of that character from the camera. This LOD can be regarded as: the more the character is far away from the camera (or off-screen) the less our algorithm worries if the chosen couple is comfortable. In next section

Table 1: Average computation time for 100 8-legged characters

	Total for Characters	Maps Preparation	CCD IK Systems	Average FPS
LOD	0.026s	0.004s	0.013s	~30fps
LOD Zoomed	0.028s	0.004s	0.006s	~30fps
No LOD	0.045s	0.004s	0.013s	~18fps

we speak about the actual performance of our real-time system.

7 Results

In our implementation, the environment is represented by two classes of objects: the terrain and the obstacles. A height map defines the terrain and is loaded without any pre-processing analysis (See Figure 11). The obstacles are mainly represented by crates with various sizes and orientations randomly generated on the terrain (See Figure 1 and 14). These crates can be static or can move on the terrain (See the accompanying video). Without changing anything in the system, we have tested it on different kinds of terrain (flat surface, smooth terrain, regular stairs, *etc.*) with different kinds of obstacles (static crates and moving ones). The size of the discretization maps used in our tests is 70×70 with $7cm$ cells, which consumes little memory and is precise enough since the maps describe only the environment close to each animated character. The animated multi-legged characters gaits are inspired from biology studies [39, 40]

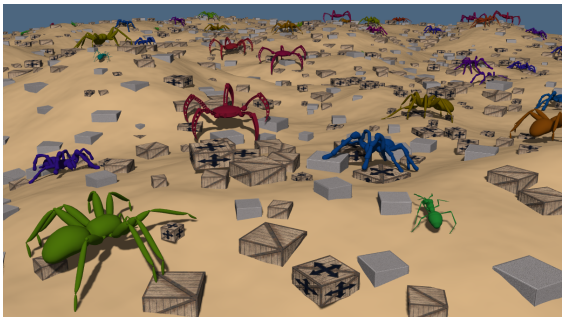

Figure 14: *Simulation snapshot.*

The results show that our system is well adapted for real-time applications. In Figure 14 and in the accompanying video we show many morphologically different characters animated in real-time using our test machine (Core i7 2.7 GHZ, 8 GB RAM, 6870 ATI Radeon HD with 1GB vRam). Table 1 shows average computation time for 100 8-legged characters. In LOD we make sure that all of the characters are in the field of view of the camera, while in LOD Zoomed we zoom on one character and make sure that at least 50% of the characters are shown on the screen. As we can observe, the maps preparation is fixed, as it is pre-computed once, for all characters at the same time. In LOD Zoomed the system loses some computation time per character as there are more characters doing full search for the best couple, while in the same time gains computation time in the IK systems as we do not calculate the IK for the off-screen characters.

8 Conclusions

We presented a system capable of procedurally generating believable locomotion animation of several multi-legged characters (like arachnids, insects, or any imaginary n-legged robots/creatures) in real-time with no *a priori* motion data nor any information about the environment. Our system is quite generic and can be applied on a variety of morphologies. In runtime the user can control many parameters like the the gait, the speed, the direction, *etc.*

Nevertheless, the current system assumes that the feet of the character can be represented by a point. Our model does not include the feet's metatarsus, which has a great impact on the naturalness of the locomotion when considering more complex characters. As a future work, it will be interesting to study this point, specially for bipeds/humans. And to study the addition of a flexible spine-like structure, which is essential when simulating quadrupeds [10]. We are also planning on improving the pelvis behavior to incorporate dynamic-like balance reactions. Our system can serve as input to higher level characters' controllers that would like to provide more animations than only the locomotion, like touching objects for discovering the environment, studying insect's behavior, *etc.*

References

[1] H. Van Welbergen, B. J. H. Van Basten, A. Egges, Zs. M. Ruttkay, and M. H. Overmars. Real time animation of virtual humans: A trade-off between naturalness & control. *Computer Graphics Forum*, 29(8).

[2] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. In *Proceedings of the Eurographics workshop on Computer animation and simulation*, 1996.

[3] Franck Multon, Laure France, Marie-Paule Cani-Gascuel, and Gilles Debunne. Computer animation of human walking: a survey, 1999.

[4] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH, 1998.

[5] Franck Multon, Richard Kulpa, Ludovic Hoyet, and Taku Komura. From motion capture to real-time character animation. In *Motion in Games*, 2008.

[6] Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. Spatial relationship preserving character motion adaptation. *ACM TOG*, 29(4), 2010.

[7] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3d linear inverted pendulum mode: A simple modeling biped walking pattern generation. In *International Conference on Intelligent Robots and Systems*, 2001.

[8] Yao-Yang Tsai, Wen-Chieh Lin, Kuangyou B. Cheng, Jehee Lee, and Tong-Yee Lee. Real-time physics-based 3d biped character animation using an inverted pendulum model. *IEEE Transactions on Visualization and Computer Graphics*, 16(2), 2010.

[9] Taesoo Kwon and Jessica Hodgins. Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, 2010.

[10] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Trans. Graph.*, 2011.

[11] Jia-chi Wu and Zoran Popović. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics*, 29(4):72:1–72:10, Jul. 2010.

[12] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29, 2010.

[13] Ronan Boulic, Nadia Magnenat-Thalmann, and Daniel Thalmann. A global human walking model with real-time kinematic personification. *Vis. Comput.*, 6:344–358, November 1990.

[14] Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. Footstep navigation for dynamic crowds. In *Symposium on Interactive 3D Graphics and Games*, I3D '11. ACM, 2011.

[15] Min Gyu Choi and Jehee Lee. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 2003.

[16] F. Lamarche. Topoplan: A topological path planner for real time human navigation under floor & ceiling constraints. *Computer Graphics Forum*, 2009.

[17] James Kuffner Jr, Jr. Koichi, Nishiwaki Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Footstep planning among obstacles for biped robots. In *Proc. of 2001 IEEE Intl. Conf. on Intelligent Robots and Systems (IROS*, pages 500–505, 2001.

[18] James Kuffner, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Online footstep planning for humanoid robots. In *in Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'03*, pages 932–937, 2003.

[19] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quaduped robot.

[20] Joel Chestnutt, Manfred Lau, German Cheung, James Kuffner, Jessica Hodgins, and Takeo Kanade. Footstep planning for the honda asimo humanoid. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[21] B. J. H. van Basten, P. W. A. M. Peeters, and A. Egges. The step space:example-based footprint-driven motion synthesis. *Comput. Animat. Virtual Worlds*, 21, 2010.

[22] Nick Torkos and Michiel van de Panne. Footprint-based quadruped motion synthesis. In *In Graphics Interface*, pages 151–160, 1998.

[23] Alain Berthoz. *La simplexité*. Odile Jacob, 2009.

[24] Michael Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. In *SIGGRAPH*, 1985.

[25] Michael Girard. Interactive design of 3d computer-animated legged animal motion. *IEEE Comput. Graph. Appl.*, 7, 1987.

[26] Rune Skovbo Johansen. *Dynamic Walking With Semi-Procedural Animation*. PhD thesis, 2009.

[27] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. In *SIGGRAPH*, New York, NY, USA, 1990. ACM.

[28] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Trans. Graph.*, 28, 2009.

[29] Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph.*, 27, 2008.

[30] Donald F. Hoyt, Steven J. Wickler, Edward, A. Cogger, Department Of Biological Sciences, and Department Of Animal. Time of contact and step length: The effect of limb length, running speed, load carrying and incline. *J. Exp. Biol*, 203, 2000.

[31] Arthur D. Kuo. A simple model of bipedal walking predicts the preferred speed-step length relationship. *Journal of Biomechanical Engineering*, 2001.

[32] Biren A. Patel Jesse W. Young and and Nancy J. Stevens. Body mass distribution & gait mechanics in fat-tailed dwarf lemurs (cheirogaleus medius) & patas monkeys (erythrocebus patas). *Journal of Human Evolution*, 2007.

[33] VERNE T. INMAN. Human locomotion. *Canadian Medical Association*, 1966.

[34] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.

[35] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Linear and Nonlinear Programming*. Wiley, 1983.

[36] O. Khatib. *Real-time obstacle avoidance for manipulators and mobile robots*. 1990.

[37] R.M.N. Alexander. *Optima for animals*. Princeton paperbacks. Princeton University Press, 1996.

[38] R.M.N. Alexander. *Principles of animal locomotion*. Princeton University Press, 2003.

[39] Donald M. Wilson. Stepping patterns in tarantula spiders. *The Journal of Experimental Biology*, 1967.

[40] R.W. Blake. *Efficiency And Economy in Animal Physiology*. Cambridge environmental chemistry series. Cambridge University Press, 2005.