In Proceedings of the ACM SIGGRAPH Conference on Motion in Games (MIG), 2016. This is the manuscript of the authors.

Full-Body Behavioral Path Planning in Cluttered Environments

Alain Juarez-Perez*
University of California, Merced

Marcelo Kallmann
University of California, Merced

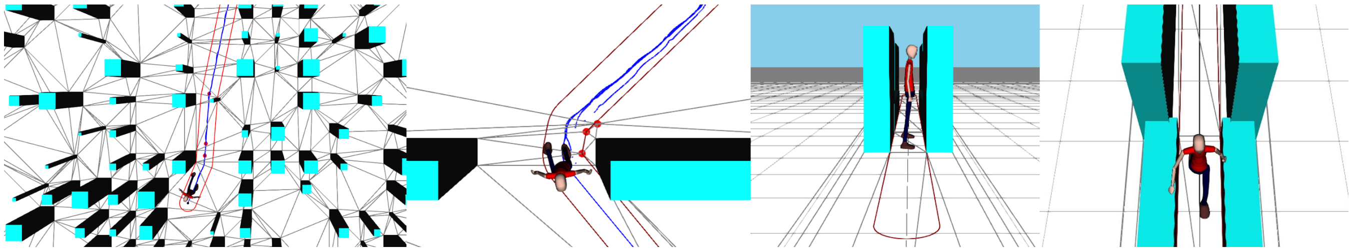

Figure 1: *From left to right: The computed paths are guaranteed to generate collision-free motions and they address the trade-off between longer paths suitable for regular walking and paths through narrow passages which require motion adaptation. Point-obstacles are inserted in narrow passages in order to force the path planner to consider alternate feasible paths. When needed the system handles narrow passages with a lateral side-stepping behavior. Full-body collision detection with the environment is performed such that regular walking can occur in narrow passages if no collisions are introduced.*

Abstract

Character navigation in virtual environments is traditionally approached by first planning a path in the free portion of the environment and then employing steering behaviors that reactively adapt to constraints encountered during path following. Unfortunately a shortcoming of this approach is that the path planning stage does not take into account locomotion behavior choices and trade-offs during the path computation. We propose an approach for incorporating the behavioral capabilities of the character in the path planning stage. The produced paths address trade-offs related to path length and navigation behavior for handling narrow passages. The proposed behavioral path planner uses a combination of clearance-based path planning and character geometry collision detection with the 3D environment in order to achieve results suitable for interactive navigation in cluttered environments. The resulted paths address the natural behavior of preferring paths with enough clearance for regular walking when possible, while also considering shorter paths which need a combination of collision avoidance and lateral steps to be executed.

Keywords: path planning, motion planning, path following, navigation, real-time graphics applications.

Concepts: •**Computing methodologies** → *Motion path planning; Motion processing;*

1 Introduction

Character navigation and locomotion in interactive virtual environments are often addressed by decoupling the overall problem in two independent stages: first, a path is planned in the free portion of the environment with a given clearance value, and second, steering behaviors are used to control a path following behavior while reactively adapting the behavior and trajectory of the character according to constraints detected during locomotion.

One considerable shortcoming of this approach is that the path planning stage does not take into consideration possible locomotion behavioral choices and trade-offs during the path computation. Therefore, behavior and trajectory adaptation are only decided reactively during path execution. When environments are cluttered with obstacles, this traditional approach often leads to characters employing difficult maneuvers in narrow passages while much more comfortable paths just slightly longer may be available.

We propose in this paper a new approach that incorporates behavioral capabilities of the character in a path planning stage efficiently executed at the navigation mesh level. The produced paths address trade-offs related to path length and navigation behavior for handling narrow passages. While several alternate paths can be quickly evaluated at the navigation mesh level, full character-environment collision checking is also performed when needed in order to explore arbitrary body movements that can be employed to overcome 3D narrow passages in the environment.

Our method relies on a character equipped with a collection of parameterized locomotion behaviors capable of performing path following. The output solution is an annotated path specifying which behavior to be used for each step along the path, according to the encountered environment constraints.

In order to remain efficient, our approach is based on planning over a customizable horizon in a greedy fashion, such that a first feasible path quickly found is continuously improved according to the defined behavioral preferences and rules, and at any given computational time threshold the current path solution can be returned.

The main contributions of our approach are two-fold. First, we propose a new method for planning navigation motions for interactive virtual characters in cluttered environments that require full-body adaptation to overcome narrow passages. Second, we introduce the concept of relying on fast navigation mesh techniques that dynamically update the navigation mesh in order to gradually impose behavioral and path constraints during the process of searching for the most suitable path.

*e-mail:ajuarez-perez@ucmerced.edu

MiG '16, October 10-12, 2016, Burlingame, CA, USA
ISBN: 978-1-4503-4592-7/16/10
DOI: http://dx.doi.org/10.1145/2994258.2994281

2 Related Work

A typical classification of locomotion synthesis is to group the methods as either physics-based or data-based. Methods that have explored the integration of planning capabilities with full-body motion synthesis are mostly data-based; however, efficient path planning techniques based on navigation meshes have been mostly developed independently from full-body motion synthesis techniques. This section makes an overview of relevant previous work in these areas.

Physics-based methods provide the possibility of achieving realistic motion adaptation to different events in the environment. For instance, a recent work demonstrates motion adaptability to multiple constraints while executing difficult tasks [Zimmermann et al. 2015]. Another relevant example is the work of Bai et al. [2012], which employs physics-based controllers to handle coordination between concurrent upper-body motions. However, this methods are computationally intensive and they are usually avoided when speed of computation is a priority.

The first methods designed to re-use data from a motion capture database [Lee et al. 2002; Arikan and Forsyth 2002; Kovar et al. 2002] gave rise to *motion graphs*. Many variants were developed to improve different aspects of the method. Interpolation of motion paths explored the solution space of the approach [Safonova and Hodgins 2007] but at the expense of increasing the size of the motion graph. Further improvements were proposed [Zhao and Safonova 2008; Zhao et al. 2009] but mostly to construct motion graphs with improved connectivity. Approaches such as the work of Lau and Kuffner [2006] have relied on search pre-computation of navigation behaviors in order to obtain interactive search performances. Similarly, motion maps [Mahmudi and Kallmann 2012] were pre-computed in order to achieve interactive motion synthesis from an unstructured motion capture database. While these represent powerful methods, they require pre-computation procedures that are time-intensive and memory-consuming.

A number of planning-based methods have been developed with the purpose of generating character locomotion among obstacles. One of the first approaches was based on deforming animation data to cope with constraints while searching for paths using a probabilistic roadmap [Choi et al. 2003]. Another approach was also developed for supporting cooperation between characters [Esteves et al. 2006]. A more recent approach uses specialized deformation techniques achieving a powerful mechanism that can be explored during navigation [Choi et al. 2011]. While these methods are able to achieve impressive results, their focus is on the motion synthesis generation and not on improving the underlying trajectory planning method that is employed. Our proposed method focuses on this problem and proposes an improved path planner that considers constraints and trade-offs that are defined from the motion controllers.

Our proposed motion planning methodology can be applied to any given motion controller as long as it can follow paths with different navigation behaviors. While our motion controller has the advantage of relying on only one example animation per behavior, there are a number of possible data-based approaches based on larger motion databases [Heck and Gleicher 2007; Lee et al. 2010], which have the potential to generate better-quality results. Motion controllers can also be based on learning methods, such as reinforcement learning. For instance, Treuille et al. [2007] and Levine et al. [2011] have employed learning for achieving powerful real-time character controllers. This methods are time-consuming during learning and controllers have to remain in a very low dimensional space. In any case, these methods could also be controlled by our proposed multi-behavior path planner.

Our multi-behavior path planner relies on path queries performed on an efficient navigation mesh representation called a Local Clearance Triangulation (LCT) [Kallmann 2014]. It allows us to efficiently perform path queries and dynamic mesh changes during the process of searching for a path suitable for multi-behavior execution. Alternative navigation meshes can be employed as long as the employed operations are supported. Texts discussing approaches for navigation meshes are available [Kallmann and Kapadia 2016].

In summary, while powerful behavioral planning approaches have been proposed in the past for addressing multi-behavior navigation in cluttered environments, previous planning-oriented approaches quickly become too expensive for interactive applications. On the other hand efficient methods based on path planning with navigation meshes have not been extended to take into account constraints or trade-offs emerging from a multi-behavior full-body motion controller, which is exactly the approach introduced in this paper.

3 Method

Our method relies on the availability of a multi-behavior character navigation controller and an efficient clearance-based navigation mesh path planner. Our proposed approach is to associate the available navigation behaviors with their clearance requirements and then employ the path planner with the different requirements such that the solution path will allow collision-free multi-behavior execution while remaining close to the shortest feasible path.

3.1 Locomotion Behaviors

The motion generation relies on a set of deformable motion capture clips. In our case the clips define three locomotion behaviors: regular frontal walking, arm-constrained frontal walking and lateral walking. The motion capture set contains annotated clips of each of the behaviors as well as the necessary transition clips between each behavior. The arm-constrained frontal walking is implemented by modifying the trajectory of the arms during the regular frontal walking so that they remain close to the body. Examples of these behaviors being used in our system are shown in Figures 1 and 6.

The first layer of our controller is based on the techniques presented in [Juarez-Perez et al. 2014; Juarez-Perez and Kallmann 2016]. The controller requires annotated transition points between data-based behaviors and as well the general direction of motion. Motion deformations are employed with small motion modifications applied incrementally at each frame of the motions, allowing us to precisely parameterize behaviors for achieving smooth path following. Whenever we need to detect if a deformed motion introduces collision we test if a limb of the character collides with itself or the environment. For testing path validity with respect to collision we test if there are collisions between limbs or between a limb and the environment, at every frame of the motion that executes the path.

We have evaluated the parameterization space of the locomotion behaviors employed in this work, which are based on incremental rotation and orientation deformations. The evaluation is based on the quality maps as shown in Figure 2. In the maps, red represents the areas that generate self-collisions while blue and green represent areas that generate motions with slightly unreachable goals for the employed Inverse Kinematics (IK) corrections which control feet constraints enforcement during the motion parameterization. The blue and green errors are considered imperceptible and represent regions that still produce acceptable motions. If we limit our deformations to lie inside the the green and blue region of the quality maps, we can control the locomotion behaviors with flexibility beyond the needs of the path following capabilities that are encountered in this work. In this way our parameterization is guaranteed to

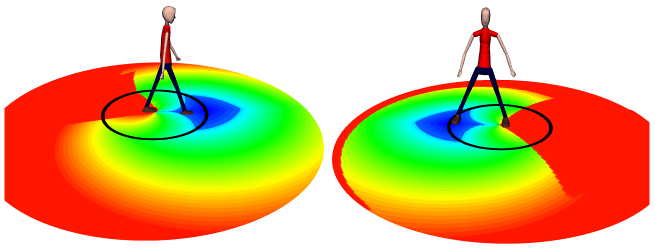

Figure 2: *Parameterization coverage maps for the frontal regular walking (left) and lateral walking (right) locomotion behaviors. The maps are on polar coordinates. The angle coordinate represents motion deformation controlling the direction of the motion. The radial coordinate represents the orientation deformation that is applied to specify the final orientation of the character's body, at the end of one step cycle. The black boundary delimits the radius representing no deformation on the final character orientation.*

precisely follow the produced paths with good quality and without generating self-collisions, and such that leg-leg collisions during sharp turns do not occur.

At any point in time the controller can receive a new behavior to be executed and the necessary transitions will be automatically applied. Motion blending is applied in order to smooth out any possible misalignment introduced by the deformations. At each concatenation point it is possible to generate a new path starting at the current position and orientation or to switch to a different behavior, such that a multi-behavior path can be executed, as it is required by the planner described below.

3.2 Behavioral Path Planner

Our behavioral path planner receives as input a goal position, and it then computes an annotated path from the current position of the character to the goal position. If the goal position lies beyond a distance threshold h, the planner improves the obtained path up to this given distance h. This distance controls the planning horizon of the overall behavioral planner.

We rely on a the local clearance triangulation (LCT) navigation mesh, which supports computation of paths with arbitrary clearance and dynamic insertion and removal of degenerate obstacles defined as a single point. This feature is used for adding constraints to the navigation mesh that reflect limitations of the locomotion behavior of the character, forcing the LCT path planner to search for alternate paths with new requirements.

We define the available behaviors of the locomotion controller as the set of n behaviors $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$, each with its correspondent clearance value c_i, $i \in \{1, \ldots, n\}$, such that $c_1 < \cdots < c_n$. We define $\mathcal{B}_n$, our last behavior, to be the preferred locomotion behavior. In our case $\mathcal{B}_n$ represents regular frontal walking. Behavior $\mathcal{B}_1$ is defined as the least preferred behavior, usually an uncomfortable behavior to execute, but the most appropriate navigation behavior for narrow passages. In our case $\mathcal{B}_1$ performs lateral walk (or side-stepping). Any behavior in between will represent a trade-off between capabilities of navigation and preference of execution. In our current setup, we only have one intermediate behavior, $\mathcal{B}_2$, which is a regular frontal walk but with the original arms swing motions modified to stay close to the body of the character, achieving a walking with arms collision avoidance behavior. Additional navigation behaviors or styles can be easily added to our framework.

Each behavior is then associated with its clearance requirements. Let c_1 be the minimal path clearance that a character behavior can handle. In our case this corresponds to behavior $\mathcal{B}_1$ and its required clearance is equivalent to the radius of a cylinder enclosing only the body of the character in a straight posture. While this clearance is enough for passing narrow passages with confidence, we want to avoid using it as much as possible. Clearance value c_2 is the clearance required for $\mathcal{B}_2$, which executes frontal walking with arms collision avoidance. Finally, c_3 is the clearance required for the preferred unconstrained regular frontal walking behavior. Throughout this paper we may refer to it as the normal behavior. The clearance values are expected to satisfy $c_3 > c_2 > c_1$, which implies from the assumption that the behaviors are ordered with respect to their capabilities of handling narrow passages. This order is also used as the inverse of the behavior preference order.

Algorithm 1 and 2 summarize the main steps of the overall behavioral path planner. They rely on several calls to the LCT path planner FINDPATH$(c; \boldsymbol{a}, \boldsymbol{b})$, which returns a free path π of clearance c between points $\boldsymbol{a}$ and $\boldsymbol{b}$. Paths are computed according to the needed clearance of each behavior and composed such that the solution path is defined as a sequence of path sections $\Pi = \{\pi_1, \ldots, \pi_k\}$, where each section π_i has an independent clearance value.

The overall algorithm is designed to minimize path length and at the same time maximize the employment of the preferred behaviors. These two objectives are contradictory because the preferred behaviors require more clearance than the non-preferred ones, and paths with high clearance will always be longer than ones with low clearance.

Algorithm 1 starts by computing a path with the minimum accepted clearance c_1. The path is shortened according to the planning horizon h, and then Algorithm 2 is iteratively called in order to evaluate and consider longer path sections that can be executed with preferred behaviors, until the entire path is processed.

Algorithm 1 - Behavioral Path Planner Entry Point

Input: current position $\boldsymbol{s}$, goal position $\boldsymbol{g}$, and the number of available behaviors n (3 in our current work.)
Output: path from $\boldsymbol{s}$ to $\boldsymbol{g}$ with annotated behavior, or null path if a feasible path does not exist.

```
1:  procedure BHPATHMAIN( $s, g, n$ )
2:      $\Pi_{cur} =$ FINDPATH($c_1; s, g$);
3:      if ( $\Pi_{cur} == null$ ) return null path;
4:      if ( $length(\Pi_{cur}) > h$ ) $\Pi_{cur}$ is shortened to length $h$;
5:      $g_h$ = final point of $\Pi_{cur}$;
6:      $\Pi_{sol} = \{s\}$;
7:      $\hat{s} = s$;
8:      while ( character not at $g_h$ ) do
9:          Append BHPATH($\hat{s}, g_h, \Pi_{cur}, n$) to $\Pi_{sol}$
10:         if ( $interactive\ mode$ ) then
11:             Update $\Pi_{sol}$ on the character execution queue;
12:         $\hat{s} =$ last point on $\Pi_{sol}$
13:         $\Pi_{cur} = \Pi_{sol}$
14:     return $\Pi_{sol}$;
```

Algorithm 1 is designed to operate in two modes: interactive character control, in which case path sections are sent for execution as they are computed, or full path processing, where the entire multi-behavior path is composed and returned, if one exists.

Algorithm 2 relies on the following 3 additional procedures:

- TESTCOLLISION$(\Pi, \boldsymbol{p}, \mathcal{B})$ starts testing for collision points from point $\boldsymbol{p}$ onwards, along path Π, and employing behavior $\mathcal{B}$. The function returns the first collision point encountered.

Algorithm 2 - Behavioral Path Planner Main Iterations

Input: initial position $\hat{\boldsymbol{s}}$, goal position $\boldsymbol{g}_h$, current path Π_{cur} and the number of available behaviors n (3 in our current work.)
Output: path section starting at $\hat{\boldsymbol{s}}$ with annotated behavior.

```
 1: procedure BHPATH($\hat{s}, g_h, \Pi_{cur}, n$)
 2:     // Evaluate if preferred behavior is feasible:
 3:     $p$ = TESTCOLLISION($\Pi_{cur}, \hat{s}, \mathcal{B}_n$);
 4:     if ( $p == null$ ) return; // Done.
 5:     $p_{obs}$ = point between $p$ and character;
 6:     Insert $p_{obs}$ as point-obstacle in LCT;
 7:     $\Pi_{new}$ = FINDPATH($c_1; \hat{s}, g_h$);
 8:     if ( $|length(\Pi_{new}) - length(\Pi_{cur})| > l_t$ ) then
 9:         // $\Pi_{new}$ is too long
10:         // Find the preferred navigation behaviors of $\Pi_{cur}$:
11:         Remove $p_{obs}$ from LCT;
12:         $q$ = FURTHEST($\Pi_{cur}, \hat{s}, \mathcal{B}_n$);
13:         for ( $i = n-1$ to 1 ) do
14:             $r$ = CLOSEST($\Pi_{cur}, q, \mathcal{B}_i$);
15:             if ( $r$ exists ) then
16:                 Append FINDPATH($c_n; \hat{s}, q$) to $\Pi_{sol}$;
17:                 Append FINDPATH($c_i; q, r$) to $\Pi_{sol}$;
18:                 break; // exit for loop
19:     else
20:         Append $\Pi_{new}$ to $\Pi_{sol}$;
21:     return $\Pi_{sol}$;
```

- **FURTHEST**$(\Pi, \boldsymbol{p}, \mathcal{B})$ first calls **TESTCOLLISION**$(\Pi, \boldsymbol{p}, \mathcal{B})$ to find the first collision point. The function will find the last step along the path such that there are no collisions when following with behavior $\mathcal{B}$.
- **CLOSEST**$(\Pi, \boldsymbol{p}, \mathcal{B})$ finds the closest point $\boldsymbol{r}$ in the path Π beyond $\boldsymbol{p}$ such that $\pi(c_i; \boldsymbol{q}, \boldsymbol{r})$ is a feasible subpath of Π and the character can continue with the normal behavior $\mathcal{B}_n$ afterwards. The function is called in a moment in which we are sure that a behavior different than the preferred one has to be used, so any attempted behavior switch can cause collisions and we are interested in detecting the first time that this does not happen.

The procedure greedily explores the possible behaviors that can navigate the path. It begins by evaluating the path traversal until a collision is found. We want to avoid this collision, but at the same time, we do not want to vary too much from the original path. To do this, the system adds a 2D point-obstacle to the navigation mesh environment so that when a new path is found, this point is avoided, and thus the previous collision is avoided. The point-obstacle location is based on the collision point projected on the floor and the root position of the character. The collision point cannot be added as an obstacle, because it lies inside one already. If the root position is used the path may change too much, specially in narrow passages. We therefore compute the point-obstacle as a point between these two places. An example of this path modification procedure is shown in Figure 3.

When a collision cannot be prevented by path modification, the algorithm will eventually find a path that exceeds the length threshold l_t, at which point it will start to test different behaviors to overcome the existing obstacle. Once a behavior is found, it annotates that section of the path and iterates back to the frontal walking behavior in order to continue the behavior search.

An example of a solution path obtained is shown in Figure 4. In this image, the first time **FURTHEST** is called, the method returns the last point along the first blue trajectory. Then it switches to $\mathcal{B}_2$ and calls **CLOSEST**, it then returns the last point along the green

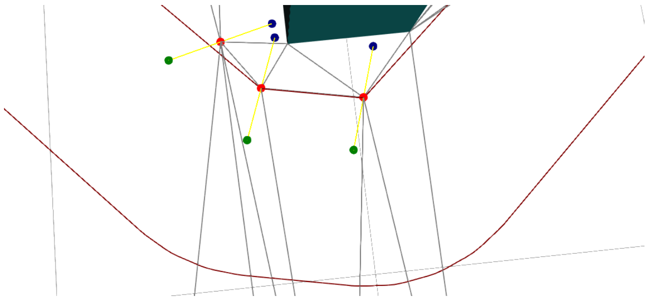

Figure 3: *The blue positions represent the projected joint position of the limbs that collided with obstacles. The green positions are the projected character root positions on the floor at the moment of each collision. The red positions are the point-obstacles that were inserted in the LCT navigation mesh as additional constraints to be considered in subsequent path queries. The final path (marked by the dark red corridor) takes into account the new point-obstacles that were inserted.*

trajectory on that path and the green path section will be annotated with its corresponding behavior.

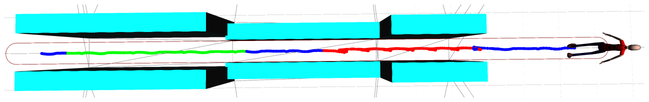

Figure 4: *Example solution path found by our method. The blue path sections can be executed with the preferred behavior $\mathcal{B}_3$. The green section with $\mathcal{B}_2$ and the red sections with $\mathcal{B}_1$.*

After successful completion the overall algorithm will return a path

$$\Pi = \{\pi_1(c_{j_1}; \boldsymbol{a}_1, \boldsymbol{b}_1), \ldots, \pi_n(c_{j_n}; \boldsymbol{a}_n, \boldsymbol{b}_n)\},$$

where $\boldsymbol{b}_i = \boldsymbol{a}_{i+1}, i \in \{1, ..., n-1\}$. The path sequence implicitly defines a solution path with behavior transitions in the following way: if $j_i = 3$ regular walking is used for that section; if $j_i = 2$, constrained walking is used; otherwise lateral walking is used.

In summary, the overall procedure will start by finding the minimum clearance path, which can be followed with the lateral stepping behavior, however since this behavior is slow and often unnecessary, preference is given to path sections of higher clearance whenever path length is not overly increased.

4 Results and Discussion

The proposed method was tested on three different scenarios and we have designed an interactive application that computes at interactive rates path following to arbitrarily given points in a given scenario.

4.1 Scenarios

In the first scenario the character has to find a path that crosses an open door with obstacles on the way. The door has a geometry that makes it impossible to be crossed without using a different behavior than the preferred regular walking. The first area is cleared by adding point-obstacles to modify the path. However, the door obstacle could not be cleared by path modifications and so a behavior switch to the arm avoidance behavior was selected, which was sufficient to avoid collisions with the door. This example is illustrated in Figure 5-right.

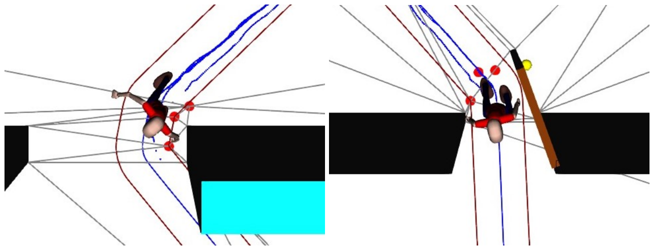

Figure 5: *Left: The obstacle was avoided by adding extra point obstacles to steer away the path. Right: The corridor is narrow so an arm avoidance behavior was needed.*

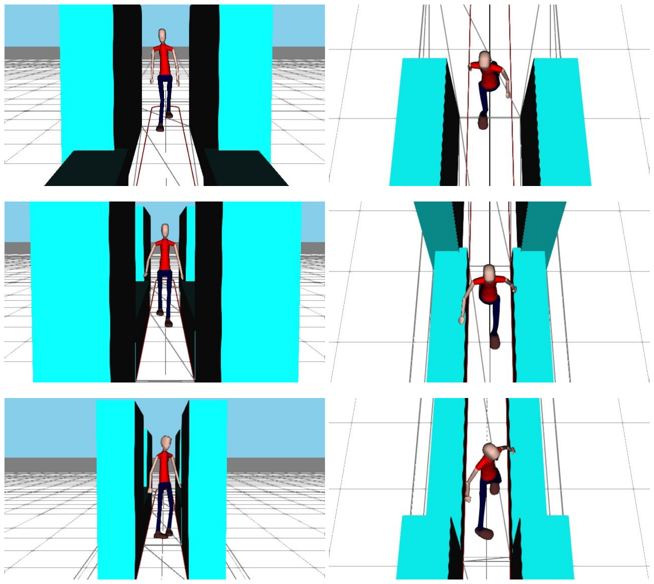

Figure 6: *Behavior transition in a narrow corridor. From top to bottom: arm avoidance behavior employed in the first section. In the middle section regular walking could be used because no collisions were detected even though the path has low clearance. In the final section lateral side-stepping is required.*

The second scenario was created to intentionally trigger our specific behaviors. It consists of a narrow corridor with specific areas designed to be traversed only by each of the available behaviors. The first area of the corridor forces an arm avoidance behavior. The next one has less clearance for the path planner, but because the obstacles don't collide with the arms, the system allows a switch to the normal behavior. Finally, the final section has the minimum accepted clearance and lateral side-stepping is required to be employed. Figure 6 shows the environment and the obtained results.

Finally we have implemented a randomized environment with obstacles of varied dimensions (Figure 7). We can control the density of the obstacles in the scene, going from easy to traverse with the normal behavior to highly cluttered and often requiring non-preferred behaviors. We use this environment for performance evaluation of our method, as described in the next sub-section.

4.2 Performance Evaluation

We evaluate the performance of our planner on two different randomized environments, one with a reasonable density of obstacles

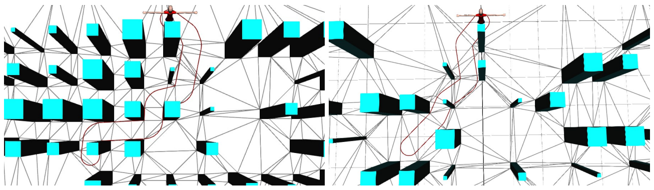

Figure 7: *Left: dense environment. Right: normal environment. Computed solution corridors are shown in both environments for a same goal point.*

considered to be typical of *normal* cases, and the other is a denser environment situation. Figure 7 illustrates the general appearance of these two environments.

In order to test our environments, we defined 100 different goals at the same linear distance from the starting point of the path queries. The length of the planned paths was equivalent to 6 and 10 character steps in the normal behavior. We then ran our behavioral path planner in both environments and measured the computation time taken to reach the solutions.

Our results are presented in Figure 8. The blue bars represent the average time of each tested type of environment, and the black line is the deviation of the results. Each bar is associated with a number and a label specifying the environment type. The number represents the linear distance between the used starting point and the goal point, and the label represents the corresponding environment. The dense environment contained several regions not allowing a direct path, generating longer paths and requiring more computation time. Because of this reason there were some high values observed and the variance in the first evaluation in the graph is considerably higher than in the other tests.

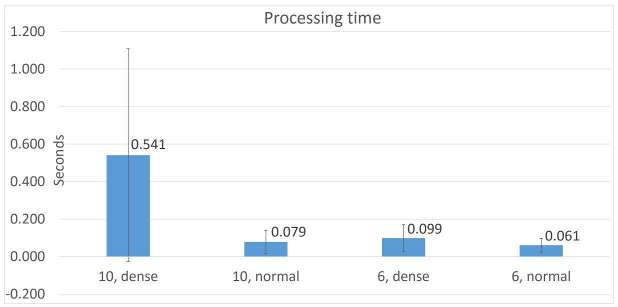

Figure 8: *Performance evaluation. The graph presents the deviation and average times required to compute paths of different lengths on dense and normal environments.*

4.3 Interactive Path Following

In our interactive test application every time the user selects a reachable point in the environment, the system computes the corresponding annotated path from the location at the end of the current motion clip to the newly specified target.

As our evaluation shows, if we limit the horizon distance h of the path that the algorithm is processing it is possible to run the system interactively regardless of the complexity of the environment. We are thus able to reach interactive results suitable for applications that require a mouse-guided target selection for locomotion, as is the case in several computer games. Our interactive application is shown on the accompanying video.

4.4 Discussion and Extensions

The behavioral path planner, although fast under a limited horizon, could be further improved by implementing the path search to occur during motion synthesis, allowing later portions of the path to be upgraded to preferred behaviors while the initial path portion is executed. Our current solution is based on a computation time limit, with the current path being returned when the given time threshold is reached.

Clearly a bottleneck of our approach is the full body collision detection with the environment at the mesh level. Currently we test full collision detection at every frame of the produced motions. Optimizations at the collision detection level are possible and would significantly improve the overall performance of the planner.

It is in principle possible to only rely on clearance parameters in order to avoid inserting point-obstacles in the navigation mesh; however, additional techniques would be needed. The LCT navigation mesh has been recently extended with the concept of extra clearance [Kallmann 2016], which could be used in such an approach. However, such an approach would not by itself allow exploring different corridors of the environment. Our approach of inserting point-obstacles provides a way to solve this problem.

Our system can be easily adapted to operate with a larger set of behaviors, in particular with reactive behaviors. We have a behavior-associated measure related directly to the behavior selection mechanism, but this can be adapted to assign a priority to each behavior in order to introduce behaviors with dynamic clearances.

5 Conclusion

We have presented a path planning system for priority-based behavior selection and path adaptation in order to achieve interactive multi-behavior navigation in cluttered environments. The system computes paths which can be precisely followed by locomotion behaviors that adapt to the given environment in order to achieve collision-free results. The computed solutions adapt to dense environments with narrow passages and we have shown that our system can run at interactive rates producing solutions that represent natural behavioral choices.

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. *ACM Trans. Graph. 21*, 3 (July), 483–490.

BAI, Y., SIU, K., AND LIU, C. K. 2012. Synthesis of concurrent object manipulation tasks. *ACM Transactions on Graphics 31*, 6 (Nov.), 156:1–156:9.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2 (Apr.), 182–203.

CHOI, M. G., KIM, M., HYUN, K. L., AND LEE, J. 2011. Deformable Motion: Squeezing into Cluttered Environments. *Computer Graphics Forum*.

ESTEVES, C., ARECHAVALETA, G., PETTRÉ, J., AND LAUMOND, J.-P. 2006. Animation planning for virtual characters cooperation. *ACM Transaction on Graphics 25*, 2, 319–339.

HECK, R., AND GLEICHER, M. 2007. Parametric motion graphs. In *Proceedings of the symposium on Interactive 3D graphics and games (I3D)*, ACM Press, New York, NY, USA, 129–136.

JUAREZ-PEREZ, A., AND KALLMANN, M. 2016. Modeling data-based mobility controllers with known coverage and quality properties. In *Digital Human Modeling*.

JUAREZ-PEREZ, A., FENG, A., KALLMANN, M., AND SHAPIRO, A. 2014. Deformation, parameterization and analysis of a single locomotion cycle. In *Proceedings of the Seventh International Conference on Motion in Games*, ACM, New York, NY, USA, MIG '14, 182–182.

KALLMANN, M., AND KAPADIA, M. 2016. *Geometric and Discrete Path Planning for Interactive Virtual Worlds*. Morgan and Claypool Publishers.

KALLMANN, M. 2014. Dynamic and robust local clearance triangulations. *ACM Transactions on Graphics (TOG) 33*, 5.

KALLMANN, M. 2016. Flexible and efficient navigation meshes for virtual worlds. In *Simulating Heterogeneous Crowds with Interactive Behaviors*, N. Pelechano, J. Allbeck, M. Kapadia, and N. Badler, Eds. Taylor & Francis.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics 21*, 3 (jul), 473–482.

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

LEE, J., CHAI, J., REITSMA, P., HODGINS, J. K., AND POLLARD, N. 2002. Interactive control of avatars animated with human motion data. *Proceedings of SIGGRAPH 21*, 3 (July), 491–500.

LEE, Y., WAMPLER, K., BERNSTEIN, G., POPOVIĆ, J., AND POPOVIĆ, Z. 2010. Motion fields for interactive character locomotion. *ACM Transactions on Graphics 29*, 6 (Dec.), 138:1–138:8.

LEVINE, S., LEE, Y., KOLTUN, V., AND POPOVIĆ, Z. 2011. Space-time planning with parameterized locomotion controllers. *ACM Trans. Graph. 30*, 3 (May).

MAHMUDI, M., AND KALLMANN, M. 2012. Precomputed motion maps for unstructured motion capture. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, 127–136.

SAFONOVA, A., AND HODGINS, J. K. 2007. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (Proceedings. of SIGGRAPH) 26*, 3.

TREUILLE, A., LEE, Y., AND POPOVIĆ, Z. 2007. Near-optimal character animation with continuous control. In *Proceedings of ACM SIGGRAPH*, ACM Press.

ZHAO, L., AND SAFONOVA, A. 2008. Achieving good connectivity in motion graphs. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation (SCA)*, 127–136.

ZHAO, L., NORMOYLE, A., KHANNA, S., AND SAFONOVA, A. 2009. Automatic construction of a minimum size motion graph. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

ZIMMERMANN, D., COROS, S., YE, Y., SUMNER, R. W., AND GROSS, M. 2015. Hierarchical planning and control for complex motor tasks. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '15, 73–81.