KEYWORDS
Human Animation, Real-time Motion Synthesis, Path Planning, Virtual Environments, Video Games.

ABSTRACT
We present a framework for path planning and character animation with interactive objects in large environments. Our work extends the motion patch algorithm to allow dynamic environments to be crafted from a set of small building blocks with embedded animation data. We develop a set of data structures and path planning mechanisms that support real-time interaction, avoidance, and traversal of dynamic objects in the environment, as well as methods for expanding the types of locomotion available to a character.

1 INTRODUCTION
As the complexity of virtual environments video games grows, so does the need for expressive characters that can interact with their surroundings in varied and subtle ways. As the number of actions that a character can perform and the number of behaviors that a character can express grows, so too does the complexity of handling the dramatically increasing interrelationships of a character’s animations.

A wealth of research has been devoted to realistic character locomotion, physically-responsive characters, and to the efficient synthesis of novel motions and transitions from a pre-existing set of animations. One problem in the field of animation that is particularly relevant to video games is the direct interaction of characters with their environment. Although video game environments have grown vast and intricate, providing a rich set of interactions and a framework that allows seamless transitions from one action to another remains a difficult problem, especially for dynamic environments.

The motion patch algorithm, developed by Lee et al (2006) provides a framework for efficiently allowing realistic character interaction with a virtual environment. In the motion patch algorithm, animations are not held in a graph or state machine internal to the character. Instead, they are embedded directly into the environment, encapsulated in small building blocks, the aforementioned motion patches. When an environment is crafted from these patches, their animations are connected in a process called “stitching,” resulting in a structure that supports both a rich and varied character interaction with the environment and efficient planning of the actions available to a character at a given location in the environment.

However, this algorithm is unsuited for some video game applications. The environment constructed from motion patches must be static at run-time even as more and more video games allow partially or fully dynamic environments. Also, motion patches are designed to hold a limited range of character locomotion speeds and cannot easily encapsulate motions of different paces.

We present a set of adaptations and extensions to the motion patch algorithm to leverage its strengths in efficient and realistic motion synthesis in complex environments while ameliorating some of the issues that make the algorithm less suitable for many video game settings. In this paper, we first describe how the motion patch algorithm can be adapted to efficiently support dynamic motion patches. Second, we supply a set of robust path planning mechanisms to support goal-oriented autonomous characters and efficient interaction with dynamic motion patches. Third, we elaborate a multi-layer approach to motion patches to support the varied gaits and character speeds common in many video games. In addition, we describe how tilable motion patches can be generated from a minimal set of animations, rather than a large corpus of motion capture data.
2 RELATED WORK

One of the most important structures for synthesizing realistic motion from small, potentially disparate, clips is the graph. The concept of forming a path of nodes whose edges reflect the costs of connecting a pair of nodes lends itself well to the problem of motion synthesis.

Schödl et al (2000) demonstrate how short video clips can be concatenated into long, smooth animations by identifying correspondences between individual frames and computing the cost of transitioning from one clip to another at a given pair of frames. Kovar et al (2002) apply this technique to motion synthesis. Combined with a branch and bound depth-first search, their algorithm demonstrates that motion graphs can produce not only long, high quality motion segments from short clips, but also segments satisfying a set of user-defined constraints, including character poses, motion types, and path following.

Lee et al (2006) define a spatially-explicit motion graph formulation by embedding motion clips into small 3D objects, then using these objects as building blocks to construct the environment. A special, tilable motion patch is constructed to handle locomotion. The tilable motion patch is a small, square grid that contains a precomputed set of paths from one edge of the grid to another edge. Each entry and exit point on the grid is specified by a node that contains the position, orientation, and pose of the character. Paths through the motion patch are specified as motion segments that connect a pair of nodes. Additional motion patches are constructed from the set of interactive objects.

When an environment is constructed, the tilable motion patches are overlaid across the environment, and the object motion patches are “stitched” into these tiles, providing an efficient representation of the actions and animations available to the character at every location in the environment. Path planning is performed in a two step process. First, a high-level path is generated from tile to tile in the environment. Then, the low-level path is computed from the set of motion segments that connect the nodes in the tiles of the high-level path.

Although motion patches very effectively encapsulate the rich set of object interaction available to a character in a complex environment, environments constructed with motion patches must remain static at run-time. The tilable motion patches also restrict the types of motions available to a character. The locomotion encapsulated within a tilable patch must be nearly uniform in pace for optimal balance between connectivity and memory footprint. Finally, motion patches do not provide an optimal structure for object traversal and goal-oriented behavior and path planning but are, instead, optimized for crowd simulation with local, wandering behavior simulation.
3 OVERVIEW

The search and stitching procedures that are used to manipulate dynamic object patches and synthesize character animation with path planning, obstacle avoidance, object traversal, and object interaction are outlined in Section 4 in the following order:

- Section 4.1 describes coarse high-level path planning that is used to select the set of tiles that a character will pass through.
- Section 4.2 defines the costs and heuristics for low-level path planning.
- Section 4.3 discusses how occlusion and stitching are handled with respect to dynamic object patches.
- Section 4.4 describes how a path is updated as object patches move.

Section 5 outlines how different locomotion types can be efficiently managed using multiple layers of tiled locomotion patches, and how multiple layers are integrated into path planning. In Section 6, a method for generating a tilable locomotion patches from a handful of specific motion segments is described. Finally, the results and conclusions are discussed in Sections 7 and 8.

4 PATH PLANNING WITH DYNAMIC OBJECT PATCHES

Two distinct kinds of motion patches are constructed: tilable locomotion patches and object patches.

**Locomotion Patch.** The locomotion patch is a prototype generated for a single type of locomotion, such as walking or running, and encapsulates the complete set of paths that a character can follow within a small square grid of approximately two cycles in length. The paths through the patch are represented as discrete motion segments that connect two nodes on the edge of the grid (Figure 1). By tiling instances of a locomotion patch uniformly across the environment and connecting the overlapping nodes of adjacent tiles, long animations can be efficiently synthesized by concatenating the motion segments from node to node. Although the locomotion patch encapsulates all locomotion data of a particular type, each tile possesses independent occlusion and stitching data that reflects the state of the objects overlapping the tile.

**Object Patch.** An object patch contains the set of animation data of the character interacting with the object. The animation data is specified relative to the object. Each instance an object in the simulation will have a corresponding instance of the object patch. The motion segments of the object patch are divided into two groups: traversal and interaction animations. Any motion segment that depicts the character passing by the object as though it were an obstacle is classified as a traversal animation. These might include vaulting over a wall or ducking under an arch. Other animations, in which the character interacts with the object as a starting or goal state, are classified as interaction animations. The former are incorporated into path planning, so that a character can realistically navigate the environment. The latter can only exist as starting or goal states in the path planning.

At run-time, the dynamic object patches are allowed to move freely across the ground plane and rotate about the vertical axis. In order to allow a character to interact with and traverse an object, the object patch must be stitched into the locomotion patches. Stitching (Section 4.3) occludes the motion segments in the underlying locomotion tiles and connects the motion segments of the object patch to those of the tiles.

To allow object patches to move at run-time, the bulk of the stitching procedure is withheld until the information is needed by the path planner. This form of lazy evaluation is necessary to prevent wasted computation on moving object patches whose stitching and un-stitching have no impact on a character’s navigation. In order to increase the efficiency of path planning, two additional organizational structures are applied to traversal animations. First, the traversal animations are divided into a set of coarse containers, called *density bins*, which surround the object patch. The traversal animations are sorted into bins based on their starting and ending position relative to the object patch. At run-time, these bins provide rapid occlusion detection for traversal animations (Section 4.3). Second, the traversal
animations are divided by layer based on the type of incoming and outgoing locomotion (Section 5).

4.1 High-Level Path Planning

Path planning occurs in two stages (Figure 2). A high-level path is constructed as a list of tiles that will be traversed on the path to the goal. At the lower level, a path of motion segments is computed from node to node through each tile. The high-level graph structure has nodes formed by the individual tiles and edges formed by the adjacency between tiles. Although the cost computation is somewhat more involved, the search heuristic at a given tile is simply the distance of the tile from the goal position:

\[ h_H(t) = \| G - t_c \| \]  \hspace{1cm} (1)

where \( G \) is the position of the goal and \( t_c \) is the position of the center of tile \( t \).

Because dynamic object patches are not stitched into the environment, the complete set of valid paths through a tile containing one or more dynamic object patches is not known. Thus, in order to prevent dead-ends and to properly allow object avoidance and traversal to occur, the high-level path planner must be provided with knowledge of the state of the tiles and overlapping object patches. Rather than simply using passable/impassable costs for the edges of the high-level path, the cost reflecting the size of the tile is combined with two cost overpassable/impassable costs for the edges of the high-level path, overlapping object patches. Rather than simply using passable/impassable costs for the edges of the high-level path, the cost reflecting the size of the tile is combined with two cost overpassable/impassable costs for the edges of the high-level path, overlapping object patches. The cost of a transition between two cells \((x,y)\) is computed as the weighted proportion of non-occluded paths created by stitching \( i \) at location \((x,y)\) and \( O_{(x,y)} \) is the number of paths occluded by \( i \). The accessibility of an object patch reflects the impact of the object patch on path planning through a tile. An object patch with accessibility close to 1 will create a corresponding traversal path for almost every path that it occludes while an object patch with a low accessibility will offer few if any traversal paths. While the accessibility describes the likelihood that a single object may be traversed, density defines the negative impact that groups of object patches have on one another. For example, a character may be able to hurdle a chair that is in his path, but if a number of chairs are grouped closely together, the character may not be able to hurdle a single chair without landing on another chair. Although this generalization does not apply to all types of object patches, it acts as a simplifying assumption to avoid expensive iterative stitching procedures.

Avoidance Probability. The avoidance probability utilizes the coarse occlusion data of a tile to approximate the proportion of non-occluded motion segments connecting the entry nodes of one edge to the exit nodes of another edge. In each tile, the avoidance probability, \( \psi_{jk} \), is computed from each incoming edge \( j \) to each outgoing edge \( k \), resulting sixteen potential probability values. When the avoidance probability of a tile is computed, cell-level occlusion is performed for each of the overlapping dynamic object patches. Each tile stores a bit field with an index for each cell. Cells occluded by object patches are marked with a 0 while non-occluded cells are marked with a 1. For efficiency reasons, the individual motion segments in the tile are not checked for occlusion. The avoidance probability is computed as the weighted proportion of non-occluded paths based on evenly distributed samples from one edge to another.

Traversal Probability. Whereas the avoidance probability describes the likelihood that a path can be found that does not pass through any of the occluded cells of the tile, the traversal probability describes the likelihood that new paths have been created by the presence of object patches (Figure 3). For example, consider an object patch consisting of a low wall that covers the breadth of a tile but contains traversal animations (i.e. animations of the character vaulting the wall). The avoidance probability describes whether the character can go around the wall without leaving the wall while the traversal probability describes whether the character can go over the wall. The traversal probability has two major components: accessibility and density. The accessibility, \( \psi_i \), of an object patch, \( i \), is a precomputed value defined as the ratio of the number traversal paths created by the presence of \( i \) to the number of motion segments occluded by \( i \), taken as an average sampled at a number locations in the locomotion patch:

\[ \psi_i = \{ \text{avg}(T_{i(x,y)}) \mid x \in [0:w], y \in [0:h] \} \]  \hspace{1cm} (2)

where \( T_{i(x,y)} \) is the number of traversal paths created by stitching \( i \) at location \((x,y)\) and \( O_{(x,y)} \) is the number of paths occluded by \( i \).

The accessibility of an object patch reflects the impact of the object patch on path planning through a tile. An object patch with accessibility close to 1 will create a corresponding traversal path for almost every path that it occludes while an object patch with a low accessibility will offer few if any traversal paths. While the accessibility describes the likelihood that a single object may be traversed, density defines the negative impact that groups of object patches have on one another. For example, a character may be able to hurdle a chair that is in his path, but if a number of chairs are grouped closely together, the character may not be able to hurdle a single chair without landing on another chair. Although this generalization does not apply to all types of object patches, it acts as a simplifying assumption to avoid expensive iterative stitching procedures.

Figure 3: An example run locomotion tile is shown with a horizontal line of chairs. The avoidance probability for this tile will be low in the vertical direction because the chairs occlude most of the paths between the top and bottom edges. On the other hand, given that the character possesses a chair hurdling animation, the traversal probability will be high despite the proximity of the chair because the traversal animations in the direction of motion will be mostly non-occluded.

To compute the density, the collision bounds of the object are extended to contain the entry and exit points of the traversal animations of the object patch. The bounds are then subdivided into a set of coarse bins that surround the object patch. The traversal animations in the object patch are subdivided into the density bins. During the collision detection phase of the
simulation, the extended bounds of the object are used in an additional broad-phase collision check. The position of each collision is computed and mapped to one of the bins. That bin is then marked as occluded. During low-level path planning, these bins will be used to quickly culled occluded traversal animations. The density, $\rho_{jk}$, of an object patch with respect to entry and exit edges $j$ and $k$ of the overlapping tiles is computed as the weighted average of the number of bins that are occluded. Each bin is weighted according to its proximity to the entry and exit edges. The traversal probability $\tau_{jk}$ of an object patch can then be defined as:

$$\tau_{jk} = \psi(1-\rho_{jk})$$

With the traversal probability computed for pair of edges in the tiles containing the object patch, the traversal probability for the tile can be computed as the maximum traversal probability of the tile’s object patches weighted by proportion of cells in the non-occluded tile. Finally, using the avoidance probability, $\nu_{jk}$, and the traversal probability, $\tau_{jk}$, the high-level cost for the tile in each entry and exit direction can be defined as:

$$g_H(t_{jk}) = \frac{1}{\max(\nu_{jk}, \tau_{jk})^\omega}$$

where $\ell$ is a constant reflecting the length of the tile and $\omega$ weights the influence of the avoidance and traversal probabilities on the cost. The maximum of the two probabilities is used because a high probability in either avoidance or traversal indicates that the tile can be incorporated into the high-level path with confidence even if the alternative probability is low.

### 4.2 Low-Level Path Planning

In Section 4.3 the impact of dynamic object patches on search is discussed. In this section, the costs and heuristics for low-level path planning are outlined. The low-level search is based on minimization of three criteria: distance traveled, change in orientation, and effort. In minimizing these criteria, the shortest, straightest, and easiest path is sought. In each motion patch (both locomotion and object patches), the cost of each path is precomputed as the weighted sum of the length, total curvature, and approximated effort per unit time.

$$g_L(P) = \alpha \int ds + \beta \int d\theta + \gamma T$$

In Equation 5, $s$ is the distance metric, $\theta$ is the orientation of the root, and $T$ is the approximated effort, while $\alpha$, $\beta$, and $\gamma$ are used to weight these criteria respectively where $\alpha$ is the weight per unit meter, $\beta$ is the weight per unit radian, and $\gamma$ simply weights the unit-less value $T$. A user-supplied $T$ value is used to approximate the effort although physically-based computation of $T$ could be used to automatically generate $T$ values for each motion. The integrals of Equation 5 are approximated with the summations for each keyframe in the motion segment. Although the summations are pre-computed, the weights are applied at run-time to vary the cost according to the setting. A hurried character will weight distance and orientation more than effort, and thus, be more amenable to leaping or climbing over obstacles, while an unhurried character will prefer a longer, but less strenuous, route. Within each entry node to a tile, the set of exit nodes is stored, along with the costs of the connecting path.

The heuristics for the search estimate the cost to the goal based on the state of the character at the exit node of the path.

$$h_L(P) = \alpha \| \vec{G} \| + \beta \cos^{-1}((\vec{G} \cdot \vec{\theta}) / \| \vec{G} \|)$$

The vector $\vec{G}$ is the vector from the end position of the path to the goal position. The vector $\vec{\theta}$ is the normalized orientation vector of the body root at the end of the path. The distance heuristic defines the minimum distance to the goal from the end of the path while the orientation heuristic defines the minimum change in orientation that must occur to reach the goal.

### 4.3 Occlusion and Stitching with Dynamic Object Patches

Although the costs and heuristics of low-level path planning are unaltered by the presence of dynamic object patches, the path planning algorithm must efficiently handle the occlusion of paths by dynamic patches and the dynamic stitching and pruning of object patch animations.

When occlusion is performed on static object patches, the bounds of the object occlude a set of cells within one or more tiles. Each of these cells stores a list of the motion segments that pass through. When a cell is occluded, each of these motion segments is disabled. One of the interesting ramifications of this method is the reduction of the search space as the amount of occlusion increases, leading to faster low-level searching in more crowded environments.

The occlusion procedure is altered with dynamic object patches. First, each motion segment in the locomotion patch has a precomputed bit field, which stores the list of cells that the motion segment passes through. Rather than disabling motion segments, this bit field is used to quickly assess whether the motion segment is occluded. As object patches move throughout the environment, no occlusion is performed. However, each tile stores the list of currently overlapping object patches. It is during high-level path planning that occlusion is performed. When the high-level path planner expands a node of its graph, which correspond to individual tiles, a bit field with an entry for each cell in the tile is reset, such that each bit stores a ‘1,’ meaning that cell is currently non-occluded. Then, for each overlapping object patch, the set of occluded cells in the tile is computed, and their corresponding bit entries are set to ‘0.’ As discussed in Section 4.1, the resulting tile bit field is used to compute the avoidance probability. The motion segments within the tile are not disabled by dynamic occlusion. When the low-level path planner expands one of the individual nodes within a tile, the set of motion segments that lead to the next tile in the high-level path are identified and checked for occlusion. A bit-masking technique is used to determine whether a motion segment is occluded.

$$(B_p \& B_r \neq B_p) \Rightarrow P \text{ is occluded}$$

Using the bitwise & operator, the cells containing motion segment $P$ are checked against the occluded cells of the tile $t$. If $B_p$ is unaltered by the operation, each cell that $P$ passes through is non-occluded. If one of the cells containing $P$ is occluded, the
corresponding bit in $B_p$ is changed from ‘1’ to ‘0,’ and the integer value of $B_p$ is altered. Using the A* search optimizations described in Cain (2002), the results of the node expansion are stored, and the occlusion is computed only once. Furthermore, using the costs and heuristics outlined in the previous section, the low-level path planner will, in most cases, only need to expand a small subset of the nodes in each tile of the high-level path, meaning that dynamic occlusion will not need to be performed on the majority of the motion segments that form the low-level search space.

Like dynamic occlusion, dynamic object traversal reduces the amount of precomputation performed when an object patch is placed. As object patches move about the environment, the density bins of each object patch are updated as described in Section 4.1. When the high-level path is computed, each object patch overlapping a tile the high-level path updates its traversal probability. When the high-level path is complete, an intermediate step is performed before low-level planning to identify the best traversal paths through each tile and estimate the costs of those traversal paths. For each object patch in the tiles of the high-level path whose traversal probability is above the minimum threshold, the best traversal animation is computed using the low-level cost and heuristic defined in Section 4.2 and maintained in a high-level observer of the path planner. As the low-level path planner computes the paths from tile to tile, the observer records the new cost and heuristic for each tile based on the individual motion segments. If the cost and heuristic of the best path through a tile exceeds the estimated cost and heuristic of the best traversal animation through the tile, the traversal animation is stitched (Figure 4) and the traversal motion segments are added to the low-level graph, and the path planning resumes as before. To reiterate, the best traversal animations for each object patch are computed and stored in a high-level structure corresponding to the list of tiles in the high-level path. Traversal animations are ignored until the cost and heuristic of the path through an individual tile exceeds those of the traversal animation. At this point, the traversal animation is stitched, and the new paths created by the traversal are added to the low-level graph.

Stitching of traversal animations is performed similarly to the method in Lee et al (2006). The first and last keyframe of the traversal animation are used in two independent stitching procedures. In stitching, the position, orientation, and pose of the character at the stitch keyframe are used to index a single cell in the underlying motion patch. For each motion segment passing through the cell, the error is computed with respect to the stitch keyframe, and a connection is formed between the motion segment and the stitched animation where the error is below the threshold value. For a traversal animation, this results in $n$ motion segments that can transition into the traversal from an entry node, and $m$ motion segments that can transition out of the traversal and proceed to an exit node (not necessarily in the same tile). The entry and exit motion segments are checked for occlusion independently. An additional bit mask is applied to Equation 8 to ignore portions of the incoming and outgoing motion segments that are no longer used.

Although traversal animations may be stitched as necessary, the stitching of other animations occurs only when the initial or goal state of the character lies within an object patch animation. For example, the character may begin or end his path sitting in a chair, but sitting and other interaction animations in the chair object patch are ignored during path planning. When the initial state or goal state lies in an object patch, the cost and heuristic are computed for each motion segment in the object patch that meets the constraints, and the best motion segment is selected, stitched, and the resulting connections are added as nodes in the low-level graph. If the animation reflects the initial state, these nodes become the initial set of unexplored nodes. If the animation reflects the goal state, these nodes become the goal nodes of the low-level path, as well as used to constrain the high-level path. The beginning and ending tiles, additional to the valid exit and entry edges, respectively, to these tiles are specified by the starting and goal nodes of the search algorithm.

4.4 Planning in the Presence of Moving Object Patches

Although dynamic objects patches are free to move at runtime, in many cases, not all will be in motion at any given time. Two physical states are defined for those patches: asleep and awake. An asleep patch has no velocity and the sum of the forces acting on the object patch imparts no acceleration on the object patch. An awake patch has either non-zero velocity or acceleration. During path planning, object patches that are asleep are incorporated in global high- and low-level path planning, while object patches that are awake are handled only in local obstacle avoidance. When an object patch transitions from the awake state to asleep or vice versa, the low-level path is updated. The motion segments are checked for occlusion and all occluded segments are removed and iteratively replaced by the low-level path planner.

Local obstacle avoidance is performed on moving object patches by predicting the short-term future state of the next two tiles in the character’s path. The bounds of nearby object patches are extended in the direction of the velocity according to magnitude of the velocity. Occlusion is then computed in the next two tiles on the path, and the low-level path is updated to incorporate the presence of the moving object patches. By finding the motion segments that are not occluded by the

**Figure 4:** An example of the paths created when a traversal animation is stitched into a tile. The image shows the character jumping over a chair that has been stitched into a tile constructed from run locomotion. The red paths emanate from an entry node to the entry stitch of the traversal animation. The blue paths begin at the exit stitch of the traversal animation and continue to the exit nodes of the tile.
extended bounds of the object patches, paths through the tiles can be found that avoid collision with nearby objects.

5 A MULTI-LAYER APPROACH

When optimizing motion patches for character animations with different paces (e.g. walking and running), it quickly becomes evident that there is no one-size-fits-all for locomotion patches. A motion patch created for walking animations will not support run animations since a single cycle of a run animation will not fit within the bounds of the patch. On the other hand, even the smallest possible motion patch designed to handle run animations will exponentially increase the number of walk paths required to cover the area and will substantially deteriorate the responsiveness of a walking character.

To address this issue, a multi-layered set of locomotion patches is crafted to handle the different paces of animation discretely. The animations are organized into sets representing the different types of locomotion. A locomotion patch is then constructed for each set of animations. These motion patches have dimensions and boundary node spacing that vary according to the pace of the locomotion. At run-time, the sets of tiles are layered independently across the environment.

Path planning with multiple layers is handled in a straightforward manner. A high-level path is computed for each layer, the layer with the lowest weighted cost is selected, and low-level path planning is performed in that layer (Figure 5). If the initial state of the character belongs to a different layer, the transition from the initial state to the desired layer is appended to the starting path, and the low-level planning begins in the state in which the transition enters the layer. Similarly, if the goal state is not contained within desired layer, the reverse transition from the desired layer to the goal state is computed and the final state of the character in the desired layer is set as the goal for the low-level path planner.

6 CONSTRUCTING MOTION PATCHES WITH MINIMAL ANIMATION SETS

Because of the fluid and unstructured nature of motion capture data, fitting a motion segment precisely to the set of start and end positions, orientations, and poses that form the nodes of a locomotion patch becomes a significant challenge. In the office demo of their tilable motion patch algorithm, Lee et al. (2006) use 40 minutes of motion capture data to construct the locomotion patch, desk patches, and behavior patches. With sampling at 30 frames per second, this results in over 72,000 keyframes. Because the motion patch is generated from unstructured motion data, a large number of motion segments are required to fully specify a tilable patch.

Using a space curve following technique combined with parametric motion blending, a tilable locomotion patch can be constructed with only a handful of animations. The animations are clustered into a small set of poses using k-means clustering (Duda et al. 2000). In space curve following, the relative distance of the root between consecutive keyframes is mapped to a space curve, such that a character’s location is bound to the curve. The orientation of the character is defined by the tangent of the space curve at the character’s location, rather than the accumulation of relative orientation changes from a fixed starting orientation. This method allows a character animated with a straight locomotion animation to make turns while maintaining realistic foot contact.

The disadvantage of this method is that turning along a space curve lacks the nuanced postures that accompany a physically-based turn. To ameliorate this issue, simple parametric motion blending (Kovar and Gleicher 2004) is applied to blend features of the turn animations into the space curve animation based on the curvature of the space curve. The turn animations are parameterized by the relative change in the orientation of the root from keyframe to keyframe. Then, when animation is being synthesized along a space curve, the change in tangent of the curve is computed and the turn animations are blended with the forward animation. The blend is weighted by the parameterized value of the turn animations. Using this method, the nuanced postures of the turn animations are smoothly applied to the character as he follows the space curve.

A space curve is used to synthesize the animation from node to node in the tilable patch. A cubic Hermite curve is applied, in which the positions of the start and end node form the start and end positions of the curve, and the direction of the start and end
tangent vectors of the curve are based on the desired orientation of the character at the start and end nodes. The length of the space curve is optimized to ensure that it is a multiple of the distance covered by one cycle of the locomotion animation. By optimizing the length of the curve, the poses at each node can be regulated. The optimization process scales the magnitude of the start and end tangent vectors of the curve to appropriately shorten or length the curve. The maximum curvature of the Hermite curve is specified to be within the range of the curvature of the turn animations. Curves that do not fall within this threshold are culled. Motion segments are synthesized to connect each pair of nodes in the tilable patch. The entry and exit orientations of the character at each node are limited to 45° increments. This increment provides a balance between flexibility of the paths from node to node and the increase in memory required to handle additional entry and exit orientations.

7 RESULTS

To test our theory, we generated two tilable locomotion patches using the space curve following approach outlined in Section 6. The first motion patch contained walk animations and was generated using only five animations, a straight walk and a slow and fast turn in each direction. The second patch contained run animations using an analogous set of five run animations. Each patch had a side length of approximately the distance covered in two cycles of the underlying straight locomotion.

Figure 6: An example of motion synthesis and path planning in a large scene constructed with dynamic motion patches. In the path shown, the character avoids pillars and chairs in his path and hurdles two lines of chairs that cannot be easily avoided.

Using the static occlusion and stitching methods described in Lee et al (2006), two to three seconds were required to occlude the overlapping walk and run tiles, and stitch the blended animations into each layer of tiles. Using dynamic occlusion and stitching, the computational cost of placing an object patch instance is constant. This allows many dynamic objects to be placed and moved freely without degrading the performance. Furthermore, in the presence relatively slow-moving (< 1m/s) objects with sparse collisions, local obstacle avoidance can be performed to dynamically update a character’s path and animation.

Table 1: Runtime statistics for 20 random paths generated in a sample scene (Figure 6). Paths were sampled in both the walk and run layers with object interaction (sitting/standing) and object traversal (hurtle). In the “Static” trial, occlusion and stitching were precomputed using the methods outlined by Lee et al (2006). The “Dynamic” trial used only dynamic motion patches whose occlusion and stitching were performed using our methods.

<table>
<thead>
<tr>
<th></th>
<th>Static Scene</th>
<th>Dynamic Scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene Precomputation</td>
<td>214</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>Time (s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Search Time (s)</td>
<td>0.176</td>
<td>0.222</td>
</tr>
<tr>
<td>Average Path Length (s)</td>
<td>14.6</td>
<td></td>
</tr>
<tr>
<td>Average Stitches/Path</td>
<td>4.34</td>
<td></td>
</tr>
</tbody>
</table>

Although our algorithm introduces additional computational costs to path planning with respect to the motion patch algorithm of Lee et al (2006), we found that these costs were within real-time constraints for sequences of animation upwards of ten to twenty seconds in length. The most significant cost introduced by dynamic motion patches is the localized, run-time stitching performed during object traversal interaction. On a 2.2GHz AMD Athlon 64 3700+, each stitching procedure required approximately 10ms computation time. Although this cost can be significant when characters are performing a series of very brief animations involving frequent object interaction, this does not reflect the typical behavior of goal-oriented agents. In the average case, in which 10 or more seconds of motion is synthesized, dynamic motion patches incur an average 25% computational overhead with respect to their static counterparts (Table 1). In these situations, the computational costs of our additional path planning metrics and dynamic occlusion were negligible when compared to the cost of stitching.
8 DISCUSSION

The motion patch algorithm developed by Lee et al (2006) is primarily geared toward crowd simulation in large environments with a wealth of interactive objects. With our adaptations and contributions to motion patches, including support for dynamic objects, robust path planning, and support for multiple locomotion types, our algorithm applies the strengths of motion patches to goal-oriented autonomous agents in large, dynamic environments. Like motion patches, our algorithm very effectively handles complex and realistic interaction with objects in the scene. Our algorithm is best suited to simulations that seek to provide a small number of characters with a rich set of animations and interactivity in a dynamic environment. But, although dynamic motion patches support rich and varied character interaction with the environment, dynamic motion patches do not exhibit the same degree of interconnectivity between objects that can be achieved with static motion patches.

Like precomputed search trees (Lau and Kuffner 2006), our work combines motion synthesis, path planning, obstacle avoidance, and object traversal. By embedding animations into the objects themselves, more realistic interaction with the objects can be achieved and a much larger set of objects can be incorporated with minimal precomputation and little cost to memory.

Finally, dynamic motion patch framework, while flexible in dealing with objects, remains fairly rigid with respect to locomotion. Using multiple layers, a few character gaits can be realistically handled, but the algorithm is not ideal for characters with a wide range of locomotion types. Parametric motion blending could be used to extend the range and types of motion, but in the current framework, any blended motion would need to be expressible as a direct analog of one of the existing locomotion types. For example, sneaking locomotion could be blended into the tilable walk patch, but only if the blend respected both the pace and foot contact of the original locomotion. This would be necessary to ensure that the motion segments and transitions in the patch were not invalidated by the blend. These restrictions on pace and foot contact also limit the range of character morphologies that can be expressed in the motion patch data. The formation of a generalized motion patch that integrates varying locomotion types and character morphologies could significantly add to the flexibility of motion patches.

ACKNOWLEDGEMENTS

We thank Michelle Lu for providing the character model and animation data.

REFERENCES


