Avatar Locomotion in Crowd Simulation

N. Pelechano
U. Politècnica de Catalunya
abeacco@lsi.upc.edu

B. Spanlang
U. de Barcelona
bspanlang@ub.edu

A. Beacco
U. Politècnica de Catalunya
npelechano@lsi.upc.edu

Abstract

This paper presents an Animation Planning Mediator (APM) designed to synthesize animations efficiently for virtual characters in real time crowd simulation. From a set of animation clips, the APM selects the most appropriate and modifies the skeletal configuration of each character to satisfy desired constraints (e.g. eliminating foot-sliding or restricting upper body torsion), while still providing natural looking animations. We use a hardware accelerated character animation library to blend animations increasing the number of possible locomotion types. The APM allows the crowd simulation module to maintain control of path planning, collision avoidance and response. A key advantage of our approach is that the APM can be integrated with any crowd simulator working in continuous space. We show visual results achieved in real time for several hundreds of agents, as well as the quantitative accuracy.

Keywords: Crowd animation, Foot sliding.

1. Introduction

Crowd simulation in real time for computer graphics applications, such as training and video games, requires algorithms for not only agent navigation in large virtual environments while avoiding obstacles and other agents, but also rendering highly complex scenes with avatars to enhance realism. To achieve that, either of these steps can become a major bottleneck for real time simulation. Therefore, trade-offs between accuracy and speed are necessary. Simulating human motion accurately whilst keeping within constraints is not an easy task, and although many techniques have been developed for synthesizing the motion of one agent, they are not easily extended to large numbers of agents simulated in real time.

To achieve natural looking crowds, most of the current work in this area uses avatars with a limited number of animation clips. In most cases, problems arise when the crowd simulator module updates the position of each agent's root without taking into account movement of the feet. This yields unnatural simulations where the virtual humans appear to be 'skating' in the environment, which is known as 'foot sliding'. Other problems emerge from the lack of coherence between the orientation of the geometric representation of the agents and direction of movement. As we increase model sophistication through enhanced path selection, avoidance, rendering, and inclusion of more behaviors, these artifacts become more noticeable.

In order to avoid these problems, most approaches either perform inverse kinematics, which implies a high computational cost that does not allow large crowd simulations in real time, or adopt approaches where the animation clip being used drives the root position which limits movements and speeds.

In this paper we present an Animation Planning Mediator; a highly efficient animation synthesizer that can be seamlessly integrated with a crowd simulation system. The APM selects the parameters to feed a motion synthesizer while it feeds back to the crowd simulation module the required updates to guarantee consistency. Even when we only have a small set of animation clips, our technique allows a large and continuous variety of movement. It can be used with any crowd simulation software, since it is the crowd simulation module which drives the movement of the virtual agents and our module limits its work to adjusting the root displacement and skeletal state.

2. Related Work

We can classify crowd simulation models into two approaches. The first encompasses those models that calculate the movement of agents in a virtual environment without taking into consideration the underlying animation. The second is defined by those that have a core set of animation clips that they play back, or blend between, to move from one point to another.

The first group suffers from an artifact known as foot sliding, since the position of the characters is updated without considering the foot position. Notice this is not exactly the same problem addressed by other works removing motion capture noise [5]. In this group we have many examples using different crowd simulation models, such as social forces [4], rule-based models [13], cellular automata [17], flow tiles [1], roadmaps [16], continuum dynamics [18] and forces models parameterized by psychological and geometrical rules [12].

The second approach puts effort into avoiding foot-sliding while limiting the number of possible movements for each agent. Lau and Kuffner [8] introduced pre-computed search trees for planning interactive goal-driven animation. These models do not perform motion blending, are often limited to a small graph of animations and play one animation clip after another, thus moving the agent according to the root movement in each animation clip. As such they do not perform well in interactive, dynamic environments, especially with dense crowds where collision response forces have an impact.

Recent trends in character animation include driving physical simulations by motion capture data or using machine learning to parameterize motion for simplified interactive control. Examples are inverse kinematics and motion blending based on gaussian process statistics or geostatistics of motion capture data [2][11]. Such techniques avoid foot sliding but are computationally more expensive.

Through interpolation and concatenation of motion clips, new natural looking animations can be created [20]. Kovar et. al. introduced motion graphs [6]. Zhao et. al. extended them to improve connectivity and smooth transitions [23]. These techniques can avoid foot sliding by using the root velocity of the original motion data, but they require a large database of motion capture data to allow for interactive change of walking speed and orientation.

Ménardais et al. were able to synchronize and adapt different clips without motion graphs [10].

Proportional derivative controllers and optimization techniques are used [22] [14] to drive physically simulated characters. Goal-directed steps can perform a controlled navigation [21]. While such techniques show very impressive results for a single character in real time, the computational costs mean they are not suitable for real time large crowd simulations.

Kovar et. al. [7] presented an online algorithm to clean up foot sliding artifacts of motion capture data. The technique focuses on minute details and therefore is computationally not suitable for large real time crowds.

Treuille et. al. [19] generated character animations with near-optimal controllers using low-dimensional basis representation. This approach also uses graphs but, unlike previous models, blending can occur between any two clips of motion. They also avoid foot sliding by re-rooting the skeletons to the feet and specifying constraint frames, but their method requires hundreds of animation clips which is very time consuming to gather.

Recently, Gu and Deng [3] increased the variety realism creating new stylized motions from a small motion capture dataset. Maïm et. al. [9] apply linear blending to animations selected based on the agent's velocity and morphology achieving nice animations for crowds but without eliminating foot sliding.

3. The Framework

The framework employed for this work performs a feedback loop where the APM acts as the communication channel between a crowd simulation module and a character animation and rendering module. The outline of this framework is shown in Figure 1.

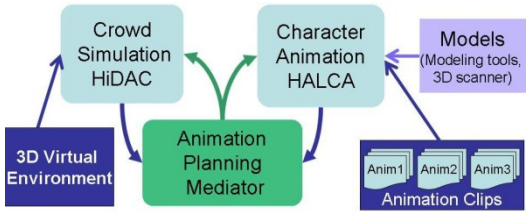

Figure 1 : Framework

For each frame, the crowd simulation module, CS, calculates the position p, velocity v, and desired orientation of the torso θ. This informa-

tion is then passed to the APM in order to select the parameters to be sent to the character animation module, CA, which will provide the next character's pose, $\mathcal{P}$. Each pose is a vector specifying all joint positions in a kinematic skeleton.

The APM calculates the next synthesized animation S_i which is described by the tuple $\{A_i, dt, b, \mathcal{P}, \mathbf{v}, \theta\}$ (see Table 1).

p	Agent root position.
$\mathbf{v}$	Agent velocity.
θ	Angle indicating torso orientation in x-z plane.
A_i	Animation clip for agent i.
dt	Differential of time for blending animation clip A_i
b	Blending factor when changing animation clip, i.e. when $A_i(t) \neq A_i(t-1)$, where t indicates time.
$\mathcal{P}$	Pose given by the skeletal state of the agent.

Table 1 : Input/Output variables of the APM

The APM may need to slightly adjust the agent's position and velocity direction in order to guarantee that the animations rendered are smooth and continuous. It is thus essential that the crowd simulation model employed works in continuous space and allows for updates of the position and velocity of each agent at any given time in order to guarantee consistency with the requirements dictated by the animation module. We have used the crowd simulation (CS) model HiDAC [12].

The torso orientation at time t, $\mathbf{w}_t$, is obtained from the velocity vector $\mathbf{v}_t$ after applying a filter so that it will not be unnaturally affected by abrupt changes.

$$\mathbf{w}_t = \omega_o \mathbf{w}_{t-1} + \mathbf{v}_t$$

where ω_o is the orientation weight introduced by the user and $\mathbf{w}_{t-1}$ is the direction of the orientation vector at time $t-1$. The orientation angle θ of the vector $\mathbf{w}$ is measured relative to the positive x-axis (since either the angle or the vector can be calculated from the other).

This orientation filter is applied as we want the position of the character to be able to react quickly to changes in the environment such as moving agents and obstacles, but we need the torso to exhibit only smooth changes, as without this the result will be unnatural animations where the rendered characters appear to twist constantly. Through filtering we can simulate an agent that moves with a slight zigzag effect, while the torso of the rendered character moves in a constant direction.

For the animation and visualization of avatars (CA) we are using a hardware accelerated library for character animation (HALCA)[15]. The core consists of a motion mixer and an avatar visualization engine. Direct access to properties such as the duration, frame rate, and the skeletal states of an animation are provided to the hosting application. Such information can be useful to compute, for example, the actual walking speed of a character when animated. Among the functionalities provided are: blending, morphing, and efficient access and manipulation of the whole skeletal state.

The CA contains a motion synthesizer which can provide a large variety of continuous motion from a small set of animations by allowing direct manipulation of the skeleton configuration. To create the library of animations, we decided to use hand created animations, although motion capture data could also be used after some pre-processing.

4. Animation Planning Mediator

To achieve realistic animation for large crowds from a small set of animation clips, we need to synthesize new motions.

The APM is used to find the best animation available while satisfying a set of constraints. On the one hand it needs to guarantee that the next pose of the animation will reflect as closely as possible the parameters given by the crowd simulation module (p, $\mathbf{v}$, θ), and on the other hand, it needs to guarantee smooth and continuous animations. Therefore, the selection of the best next character's pose needs to take into account the current skeletal state, the available animations, the maximum rotation physically possible for the upper body of a human, and whether there are any contact points to respect between the limbs of the skeleton and the environment (such as contact between a foot and the floor). Once the APM determines the best set of parameters for the next pose and passes this information to the CA for animation and rendering, it will also provide feedback to the CS in the cases where the parameters sent needed to be slightly adjusted to guarantee natural looking animations with the available set of animation clips and transitions.

During pre-processing the APM will calculate, for each animation clip average velocity $\mathbf{v}_{anim}$ in m/s by computing the total distance traveled by the character through the animation clip divided by the total duration of the animation clip, T, as well as the angle α between the torso orientation, θ_{anim}, and the velocity vector, $\mathbf{v}_{anim}$, in the animation clip. The i^{th} animation clip A_i is defined by the tuple $\{\mathbf{v}_{anim,i}, \alpha_i, T_i\}$.

During the simulation the APM takes the input parameters from the CS and proceeds through the five steps shown in Figure 2 to obtain the output tuple $\{Ai, dt, b, \mathcal{P}', p', \theta\}$ that will be sent to the CA. We explain this in detail below.

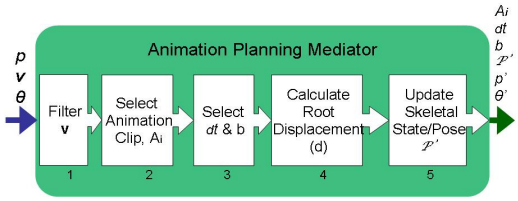

Figure 2 : Animator Planning Mediator

4.1. Animation Clip Selection

Instead of achieving different walking speeds by using hundreds of different walk animations, the algorithm can be effective with a limited number of animation clips by blending within an animation. This is given by the parameter *dt* which defines the time elapsed between two consecutive frames. Each animation clip covers a subset of speeds going from the minimum speed of 0 m/s (*dt*=0) to the original speed of the animation clip (*dt*=1).

An animation clip of walking forward can also be used to have the agent turn, as we can reorient the foot on the floor, thus reorienting the entire figure. This provides natural looking results for high walking velocities, but for slower velocities, using several turning animation clips and blending between them results in more natural looking motion.

If we consider α being the angle between the direction of movement v_{anim} and the torso orientation θ_{anim}, of an animation A_i, we can classify animations based on α and the velocity. For example, for an animation of walking sideways $\alpha = 90$ degrees and for walking forwards $\alpha = 0$ degrees.

To determine when to use each animation we classify them in a circle defined by tracks and sectors. A track is the area contained between two concentric circles. Each concentric circle has as radius the velocity of an animation, $\mathbf{v}_{anim}$. Once the tracks are defined we divide them into sectors, each of which maps to a clip.

All the animations used must satisfy the following requirements: (1) must be time aligned, (2) $\mathbf{v}$ and α must be approximately the same throughout the animation clip (within a small threshold defined by the user), and (3) animation clips must be cyclical.

Each animation clip could be used when the velocity of the agent $\mathbf{v} \leq \mathbf{v}_{anim}$, therefore we decide on which animation to assign to each sector depending on the α value. The decision points of when to switch from one animation sector to the next as the angle increases is defined as being halfway between the α values of two neighboring animations. Figure 3 graphically represents the decision framework, where colors are used to represent the animation clips assigned per sector. We have chosen some animation clips with similar velocities or angles ($v_3 \approx v_4$, $v_6 \approx v_7 \approx v_8$, and $\alpha_1 \approx \alpha_2 \approx \alpha_5$) to graphically show the splitting of tracks into sectors. Only half of the circle of animation clips is shown due to symmetry. At any time during the simulation we can run any animation backwards by using a negative *dt* and selecting the clip by flipping vertically over the x axis and mirroring in the y axis.

4.1.1 Classifying Motion Clips

Initially the algorithm starts by dividing the circle into tracks $T_1, T_2, \ldots, T_m$, where each track is defined by the velocity of an animation clip from the library (v_{anim}) and $v_1 > v_2 > \ldots > v_n$. Note that $m \leq n$ as two animations A_i and A_j with similar velocity (i.e. $|v_i - v_j| < \varepsilon_v$) will be assigned to the same track. Each track T_i is defined by its maximum and minimum velocity, $T_i = \{v_i^{\min}, v_i^{\max}\}$.

Starting from the outer track T_1 (highest velocity), the algorithm proceeds by splitting each track into sectors based on the γ values defined as being halfway between the α values of two animations.

Each sector $S_{i,k}$ inside track T_i is defined by the tuple: $\{v_i^{\min}, v_i^{\max}, \gamma_{i,k}^{\min}, \gamma_{i,k}^{\max}\}$ which corresponds to the minimum and maximum velocity, and minimum and maximum decision angles.

For each step of the algorithm, a track T_{i+1} will be split into at least as many sectors as contained within T_i. For each sector $S_{i,k} = \{v_i^{\min}, v_i^{\max}, \gamma_{i,k}^{\min}, \gamma_{i,k}^{\max}\}$ there will be a new sector $S_{i+1,k} = \{v_{i+1}^{\min}, v_{i+1}^{\max}, \gamma_{i,k}^{\min}, \gamma_{i,k}^{\max}\}$ where $v_{i+1}^{\max} = v_i^{\min}$, with

the same animation being assigned. Then further splitting of those sectors will occur for each of the new animations with $|v_{anim} - v_{i+1}^{max}| < \varepsilon_v$ depending on the α values of the remaining animations. If we let α_p be the angle of a new animation A_p and α_q be the angle of the previous animation A_q that is assigned to a new sector $S_{i+1,k}$, then if $|\alpha_p - \alpha_q| < \varepsilon_\alpha$, the new animation A_p replaces A_q for that sector. For any other case a new sector is created and the angle limits γ between sectors are recalculated.

The variety of movements that can be synthesized with this method will depend upon the number of animation clips. The more clips we have, the more sectors we can define.

During run time the APM will select the best animation clip based on the current velocity, **v**, of the agent, and the angle, φ, between the velocity and the torso orientation, θ, provided by the CS. If a change of animation is required, then the APM will determine the weight, b, necessary for blending between animations.

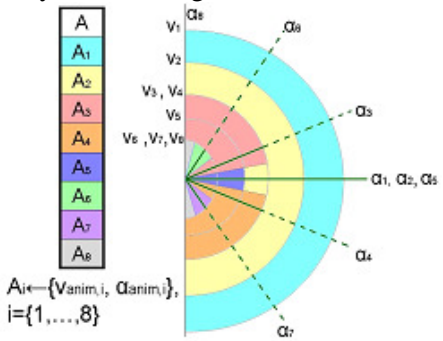

Figure 3 : Circular Decision Framework.

4.2. Blending Factors

At every frame the CS calculates the new position for each agent based on the desired path to move between an initial position to a destination, while interacting with other agents, walls and obstacles. However, since the CS is not aware of the type of animation being used for the rendering, if we take that new position to translate the root of the skinned avatar and the angle θ to reorient it, we will observe that the figures appear to 'skate', and also that there is no coherence between the orientation of the avatar and the actual animation movement.

To avoid foot-sliding and guarantee that the animation satisfies the constraints given by **v** and θ, we need to ensure that the figure appears to move according to **v** while the foot currently in contact with the floor stays in place, and the torso faces the direction given by θ. This could be done using inverse kinematics, but since we are simulating large crowds, we need a method that can be quickly calculated and applied to hundreds of 3D animated figures in real-time. Also, to avoid the time and cost of generating a large number of animation clips, we are interested in a limited number of clips that can be combined to achieve as many realistic animations as possible. We have a trade-off between the accuracy of our animations and the simplicity, and therefore speed, of our calculations.

Knowing the agent's velocity **v** and the animation velocity $\mathbf{v}_{anim}$ from the selected animation A_i, we can calculate the blending factor τ.

$$\tau = \frac{\mathbf{v}}{\mathbf{v}_{anim}} \quad , \qquad \tau \in [0,1]$$

The dt needed by the CA module to blend between poses is:

$$dt = \tau \cdot \Delta t$$

where Δt is the elapsed time between consecutive frames. At this point of our algorithm we have the new pose of the agent and thus can obtain the local coordinates of the root and the feet position.

Knowing that the root movement is driven by the foot that is on the floor during two consecutive poses, we update the root position in the global coordinates of the environment.

4.3. Calculation of Root Displacement

For the current pose $\mathcal{P}_t$, we calculate the vector $\mathbf{u}_t$ that goes from the foot on the floor to the root of the skeleton. Likewise for the pose $\mathcal{P}_{t-1}$ in the previous frame we calculate $\mathbf{u}_{t-1}$ (see figure 4). The vector $\mathbf{u}_t$ contains all the rotations that happen at the ankle and knee level and thus provides sufficient information about the leg movement.

By subtraction of vectors we can calculate root displacement **d** between frames. The vector of displacement **d**, shown in red is:

$$\mathbf{d} = \mathbf{u}_t - \mathbf{u}_{t-1}$$

And the new position is thus:

$$p_t = p_{t-1} + \mathbf{d}$$

The method is efficient enough to allow for fast calculation and extension to 3D is straight forward. From the results shown in Figure 5 we can observe that it avoids foot sliding and preserves the vertical root movement that appears when we walk fast. In the figure we have rendered the root path in black.

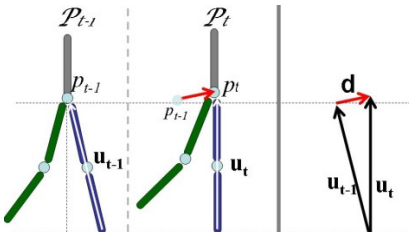

Figure 4 : Root displacement.

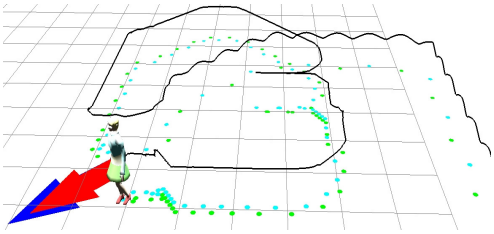

Figure 5 : Path followed by one agent.

4.4. Updating the Skeletal State

Turns in the environment can be achieved by reorienting the figure according to the velocity vector **v** and adjusting the torso orientation according to θ by modifying the skeletal configuration. This will also orient the displacement vector to move the character in the direction indicated by the CS module.

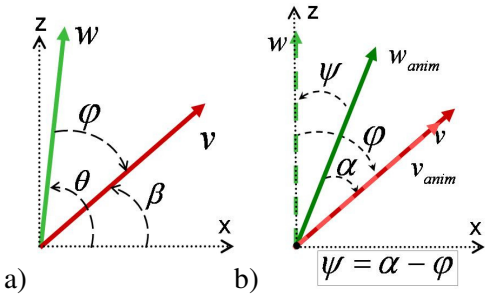

Figure 6 : Angles described in Table.

The visual effect of this is that the foot on the floor will slightly rotate in place. This is almost unnoticeable to the human eye, especially at high velocities, and thus is a trade-off worth considering as we can achieve turns in any direction without the requirement of having a large database of animation clips.
For slower velocities, we can achieve higher realism by having a number of simulations where the torso orientation does not necessarily need to be aligned with **v**. To calculate the rotations we use the following variables:

w	torso orientation vector.
β	angle in the x-z plane of the velocity vector **v** relative to the positive x-axis.
φ	angle of the velocity vector, **v,** relative to the orientation vector, **w**.
α	angle in the x-z plane of the velocity vector $\mathbf{v}_{anim}$ of the animation clip A_i relative to its orientation vector $\mathbf{w}_{anim}$.

Table 2 : Variables required for upper body rotation.

To achieve movement in the direction given by **v**, the avatar is rotated by $(\beta - \alpha)$ so that the velocity vector of the animation $\mathbf{v}_{anim}$ matches **v**. Then the torso is oriented according to **w**: the angle ψ is calculated as the difference between φ and α (Figure 6) and propagated across the spine of the character by modifying the current skeletal state of the character's pose. This allows the agent to move the root in the direction indicated by **v** while the torso is facing the direction indicated by **w**.

4.5. The APM Algorithm

The following table summarizes the APM algorithm with references to the sections where each step was explained:

Algorithm: APM	Section
$\varphi \leftarrow AngleBetween(\mathbf{w},\mathbf{v})$ $A_i \leftarrow SelectAnimation(\mathbf{v},\varphi)$	*4.1*
if $(A_i \neq A_{i-1})$ *then* $b \leftarrow Agent.BlendingFactor()$ $dt \leftarrow CalculateDT(\mathbf{v}, A_i)$	*4.2*
$\mathcal{P}_t \leftarrow getNextPose(A_i, \mathcal{P}_{t-1}, dt)$ $\mathbf{d} \leftarrow CalculateDisplacement(\mathcal{P}_t, \mathcal{P}_{t-1})$	*4.3*
$\alpha \leftarrow GetAngleAnim(Ai)$ $\beta \leftarrow CalculateAngle(\mathbf{v})$ $\psi \leftarrow \alpha - \varphi$ $agentCA.PropagateAngleSpine(\mathcal{P}_t, \psi)$	*4.4*

Algorithm 1 : The APM Algorithm

With the parameters calculated by the APM, we apply an absolute rotation of $(\beta - \alpha)$ and add the displacement, d, to the previous position to render our character satisfying the requirement given by the CS.

5. Results

The method presented only needs a small set of animation clips to obtain visually plausible results. By increasing the number of animations and/or the quality of those animations (i.e. using

motion capture data), we would obtain improved results with no additional cost during simulation time.
The animation library shown in the accompanying videos consists of four walking forward animations, two side-step animations and four "walking on an angle" animations.
Per agent and per frame, on a Intel Dual Core 3GHz with 4GB of RAM, the root displacement computation is less than 0.8µs. The whole APM algorithm requires less than 0.021ms. Therefore, we could incorporate the APM into any crowd simulation module, with an additional linear cost per frame of 0.021 ms times the crowd size. Since typically not all the agents in a crowd are visible simultaneously, we could apply the APM algorithm to only the few hundred agents closest to the camera.

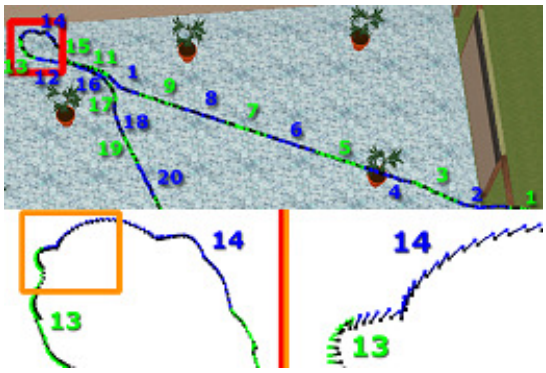

Figure 7 : Path followed by an agent with close-ups of a turn.

In order to satisfy the constraints given by the CA, our APM may need to slightly modify the position of the root given by the CS. Figure 7 shows the path followed by an agent, with a zoomed view of a sharp. Each blue/green segment corresponds to 75 frames of the animation (3 seconds). We have represented the deviation introduced by the algorithm as a segment with a green/blue ending indicating the position given by the CS and a black one indicating the corrected position calculated by the APM.
Deviation is bigger where there are abrupt turns and blending simultaneously. We have calculated the average deviation between the CS position and the APM corrected position in order to determine the impact of our algorithm on the final path. Running at 25fr/s, on average we obtain a deviation of less than 7.78mm per frame. Larger deviations correspond to segments 13 and 14 which are when the agent is turning sharply (Figure 8).

In Figure 9 we can see that deviations are larger at turns and reach maximum values when there are repulsion forces between agents (for example at the doors where agents congregate). Most of the time the deviation stays under 1cm per frame.

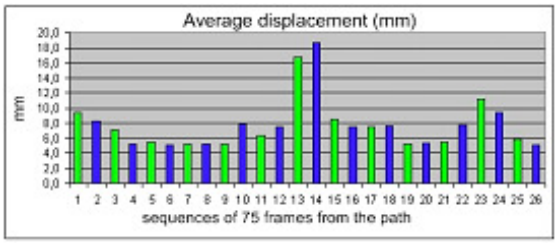

Figure 8 : Deviations in mm for each segment of the path shown in figure 7.

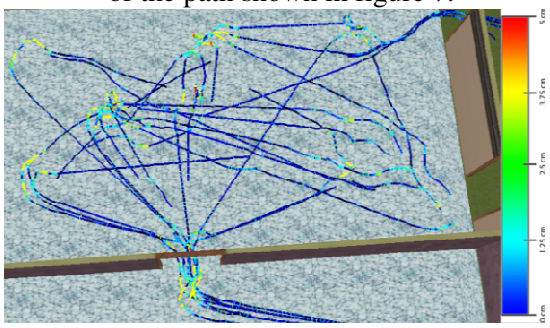

Figure 9 : Paths for 20 agents showing the deviation with a color gradient scale.

6. Conclusions

We present a realistic, yet computationally inexpensive, method to achieve natural animation without foot-sliding for crowds. Our goal was to free the crowd simulation module from the computational work of achieving natural looking animations and focus instead on developing crowd behavior that looks realistic. Natural looking results can be obtained with a minimal library of animation clips and this method can be integrated with any crowd simulation software.
As our model requires a foot on the floor to calculate the root displacement, it has some limitations. Currently we cannot achieve running simulations where both feet are in the air for certain frames. We plan on addressing this problem in future work.
The library employed for this work was hand created, and thus our original animations suffer from rigidity which affects the overall look of the crowd. Since the quality of the final crowd animation depends strongly on the set of animation clips available, having time aligned motion

capture animation clips will achieve more natural looking results.

7. Acknowledgements

This research has been funded by the Spanish Government grant TIN2010-20590-C0-01 and the ERC advanced grant TRAVERSE 227985.
A. Beacco is also supported by the grant FPU-AP2009-2195 (Spanish Ministry of Education).

References

[1] S. Chenney, "Flow Tiles". In. Proc. ACM Symposium on Computer Animation, 233–242. 2004

[2] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović. "Style-based inverse kinematics". In ACM Transactions on Graphics, 23(3), 522–531. 2004

[3] Q. Gu and Z. Deng, "Context-Aware Motion Diversification for Crowd Simulation," In IEEE Computer Graphics and Applications. 2010

[4] D. Helbing, I. Farkas, and T. Vicsek. "Simulating Dynamical Features of Escape Panic", In Nature, 407, 487-490. 2000

[5] L.Ikemoto, O.Arikan, and D. Forsyth, "Knowing when to put your foot down". In ACM Proceedings of the 2006 symposium on Interactive 3D graphics and games (I3D '06). 49-53. 2006

[6] L. Kovar, M. Gleicher, and F. Pighin. "Motion Graphs". ACM Transactions on Graphics 21, 3, 473-482. 2002

[7] L. Kovar, J. Schreiner, and M. Gleicher, "Foot skate cleanup for motion capture editing" ACM Proc. Symp. on Computer Animation. ACM Press. 97-104. 2002

[8] M. Lau and J. Kuffner, "Precomputed Search Tree: Planning for Interactive Goal-Driven Animation". Proc. Symp. on Computer Animation. 299-308. 2006

[9] J. Maïm, B. Yersin, J. Pettré, and D. Thalmann, "YaQ: An Architecture for Real-Time Navigation and Rendering of Varied Crowds". IEEE Computer Graphics and Applications, 29(4), 44-53. 2009

[10] S. Ménardais, R. Kulpa, F. Multon and B. Arnaldi. "Synchronization for dynamic blending of motions". In. Proc. ACM Symposium on Computer Animation, 325–336. 2004

[11] T. Mukai and S. Kuriyama. "Geostatistical motion interpolation". In ACM Transactions on Graphics, 24(3), 1062–1070.2005

[12] N. Pelechano, J.M. Allbeck, and N.I. Badler, "Controlling Individual Agents in High-Density Crowd Simulation", ACM Proc. Symp. on Computer Animation.99-108. 2007.

[13] C. Reynolds. "Flocks, Herds, and Schools: A Distributed Behavior Model", In Proc. of ACM SIGGRAPH, 25-34.1987.

[14] M. da Silva, Y. Abe, and J. Popović. "Simulation of human motion data using short-horizon model-predictive control". In. Computer. Graphics. Forum, 27(2), 371–380. 2008.

[15] B. Spanlang. Halca hardware accelerated library for character animation. Technical report. Event Lab, Universitat de Barcelona. event-lab.org. 2009.

[16] A. Sud, E. Andersen, S. Curtis, M. Lin, and D. Manocha,"Real-time Path Planning for Virtual Agents in Dynamic Environments" In. Proc. IEEE Virtual Reality, 91–98. 2007.

[17] F Tecchia, C. Loscos, R. Conroy, and Y. Chrysanthou. "Agent behavior simulator (ABS): A Platform for Urban Behavior Development", In Proc. of ACM Games Technology Conference, 17–21. 2001.

[18] A. Treuille, S. Cooper, and Z. Popović. "Continuum Crowds" In. Proc. ACM Transactions on Graphics SIGGRAPH, 1160–1168. 2006.

[19] A. Treuille, Y. Lee, and Z. Popović, "Near-optimal Character Animation with Continuous Control", Proc. ACM Siggraph, 26(3), Article 7. 2007.

[20] A. Witkin, and Z. Popović, "Motion Warping", In. Proc. of Siggraph'95. 105-108. 1995.

[21] C.C., Wu, and V. Zordan, "Goal-directed stepping with momentum control", ACM Symp. on Computer Animation. 2010.

[22] K.K.Yin, K. Loken, and M. van de Panne. "Simbicon: Simple Biped Locomotion Control". In ÁCM Transactions on Graphics, 26(3), Article 105, 2007.

[23] L. Zhao and A. Safanova, "Achieving Good Connectivity in Motion Graphs", ACM Proc. Symp. on Computer Animation.139-152. 2008.