Gaze Behaviors For Virtual Crowd Characters

Helena Grillon[1], Barbara Yersin[1], Jonathan Maïm[1], and Daniel Thalmann[1]

EPFL, Lausanne, Switzerland,
{helena.grillon, barbara.yersin, jonathan.maim, daniel.thalmann}@epfl.ch

Abstract. Nowadays, crowds of virtual characters are used in many domains such as neurosciences, psychology, and computer sciences. Since as human beings, we are natural experts in human being representation and movement, it makes it that much harder to correctly model and animate virtual characters. This becomes even more challenging when considering crowds of virtual characters. Indeed, in addition to the representation and animation, there is the mandatory trade-off between rich, realistic behaviors and computational costs. In this paper, we present a crowd engine, to which we introduce and extra layer which allows its characters to produce gaze behaviors. We thus enhance crowd realism by allowing the characters composing it to be aware of their environment and other characters and/or a user.

1 Introduction

Real-time crowd simulation has become a topic of increasingly high interest. Virtual crowds are used in many domains ranging from the movie or gaming industries to neurology and psychology. Simulating real-time crowds is a very challenging topic. First, as humans, we are natural experts when it comes to human representation and animation. It makes it that much harder to create realistic virtual characters, both in aspect and movement. In addition, dealing with crowds of characters rises a mandatory trade-off between rich and realistic representations and behaviors, and computational costs. Individual character animation may provide realistic results but is computationally expensive. Conversely, global crowd behavior design is much faster but results in a loss of character individuality.

We believe that an important aspect which can greatly enhance crowd simulation realism is for the characters to be aware of their environment, other characters and/or a user. This can partly be achieved with the navigation and path planning algorithms present in our crowd engine.

In this paper, we propose a method, integrated in the crowd simulation pipeline, to obtain more advanced behaviors than what navigation can provide. To add attentional behaviors to crowds, we are confronted to two issues. The first one is to detect the points of interest for characters to look at. The second one is to edit the character motions for them to perform the gaze behavior.

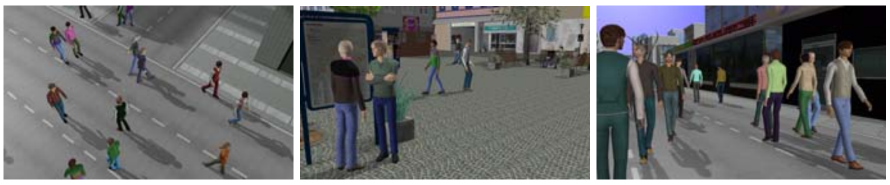

Fig. 1. Examples of gaze behaviors for virtual crowd characters.

2 Related Work

2.1 Real-Time Crowd Simulation

Simulating crowds in real time is a challenging task that is usually divided into the following topics: rendering, navigation, and behavior.

To render large crowds, the best approach is to use a level-of-detail approach [1]: high quality rendering techniques are used for virtual humans close to the camera, while faster, less accurate methods are exploited at farther distances. Many solutions use impostors for their efficiency. Impostors are 2D animated images, mapped onto a quad formed by 2 triangles. One pioneer work in this domain was achieved by Dobbyn et al. [2]: they used pre-computed rigid meshes in front of the camera, and switched seamlessly to impostors for more distant characters. Later, Millan and Rudomin [3] presented a solution combining impostors and instanced geometries, maximizing the use of the GPU. More recently, Kavan et al. [4], presented a virtual human new level of detail, called *polypostor*: they are composed of a small set of 2D polygons, and animated by displacing their vertices. To avoid crowds of cloned entities, much work has been achieved on color variety, or how to modulate the colors of body parts for each instance of a same characters [5, 2, 6]. A recent study [7] has demonstrated that variety at the color level has much more impact on the user than varying the animations solely.

In the domain of navigation and behavior, many approaches have been studied to efficiently simulate large crowds. The pioneer work of Reynolds [8, 9] presented simple rules to easily steer individuals for performing basic actions, such as "seek", "flee", etc. Other important branches of study on crowd navigation were introduced: approaches can be Physics-based [10, 11], sociology/psychology-based[12, 13], example-based [14–16], and hybrid [13, 17].

2.2 Simulation of Attentional Behaviors

Attention Models The synthesis of human vision and perception is a complex problem which has been tackled in many different ways. Models of synthetic vision based on memory have been developed for the navigation of characters [18, 19]. Other proposed models relied on bottom-up, stimulus-driven rules [20, 21] and saliency maps [25–28]. Similarly, much work has been conducted in the simulation of visual attention and gaze in Embodied Conversational Agents [22–24]. On a different note, [29] described an eye movement model based on saccade empirical models and statistical models of eye-tracking data, and [30] proposed a head-neck model based on biomechanics. All these methods give very convincing results but are prohibitive in terms of computational times or not applicable to crowd simulations.

Motion Editing A large category of motion editing techiniques use analytic Inverse Kinematics (IK) [31, 32]. In this domain, [33] proposed a method to edit a pre-existing animation to satisfy a set of user defined constraints for human like figures. [34] extended this method to remove footskate from motion captured animations. [35] used Kalman filters and a set of rules to assign varying importance to a set of tasks which they then solved with a dedicated IK solver. [36] proposed a hierarchical Cyclic Coordinate Descent algorithm to deal with spacetime constraints for human-like figures. These analytic methods are dedicated to the positionning of end-effectors, whereas we are interested in controlling the final orientation of the eyes, head, and spinal joints over time. Several other methods use Jacobian based IK solvers to edit motions [37, 38]. While these methods are generic enough to possibly use any kind of constraints, the use of Jacobian inversion causes prohibitive computational costs that are not compatible with our framework.

3 System Overview

Our crowd engine is composed of 3 main elements, illustrated in Figure 2: the variety component, the navigation component, and the real-time component. The variety and navigation components are processed off-line, before launching the simulation, while the real-time component represents our pipeline, which is run at each frame of the simulation.

The variety component takes care of the visual appearance of pedestrians composing the crowd: it generates the human instances, vary their color, shape, and height. This component is also responsible for generating all the animations that will be used at runtime. Many locomotion animations are generated at various speeds with a locomotion engine (see next section), and some quieter actions, such as talking, standing, sitting, are hand-designed. Variations are applied to these animations, *e.g.*, hand on the hip, in the pocket, and finally, they are all stored in a dedicated database that is later uploaded at runtime.

The navigation component is responsible for analyzing and structuring the environment. From the geometry of an environment, we create a navigation graph [39, 40], which is then used to compute many paths for pedestrians to follow. Semantic information can also be input in the navigation graph to locally trigger special behaviors in the crowd.

Finally, the real-time component is our main pipeline. It is composed of 4 steps, which are successively processed as follows:

– The Scaler applies scores to each vertex of our navigation graph in order to know which level of detail is applied to which humans.
– The Simulator navigates each individual towards its next waypoint, using different algorithms, depending on the score obtained in the Scaler.
– The Animator has to update the posture of each virtual human. Once again, depending on the score obtained by each virtual human, the animation process is different.
– The Renderer first takes care of computing the shadows for the characters as well as the scene. In a second pass, it displays the crowds and environment.

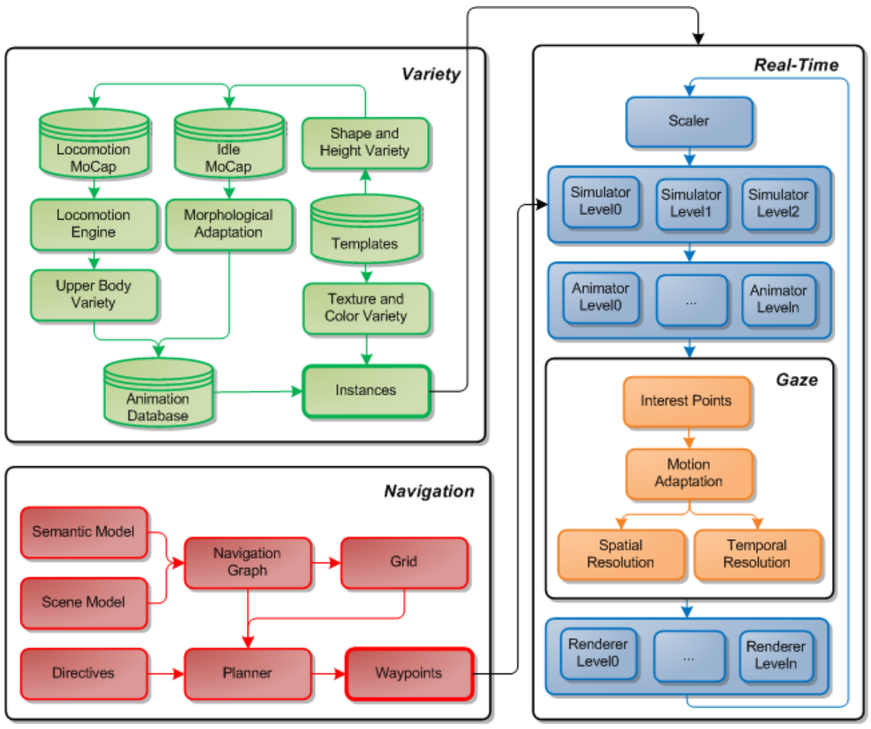

Fig. 2. system overview

The attentional behaviors are added to the walking character motion after the Animator part of the pipeline, in the Gaze module depicted in Figure 2. The first step is to define the interest points, *i.e.* the points in space which we consider interesting and which therefore attract the characters' attention. We use several different methods to do this depending on the result we want to obtain:

– The interest points can be defined as regions in space which have been described as interesting (see Section 4). In this case, they will be static.

– They can be defined as characters evolving in space. All characters may then potentially attract the attention of other characters as long as they are in their field of view. In this case, we have dynamic constraints, since the characters move around.
– They can be defined as a user if we track a user interacting with the system. A coupled head- and eye-tracking setup allows us to define the position of the user in the 3D space as depicted in Figure 3. Characters may then look at the user.

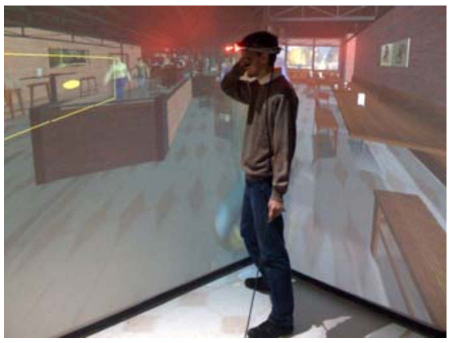
Fig. 3. A user is tracked in a CAVE environment. His position and orientation is used to interact with the characters walking in the virtual environment.

The second step to obtain the desired attentional behaviors consists in computing the displacement map which allows for the current character posture to achieve the gaze posture, *i.e.* to satisfy the gaze constraints. Once the displacement map has been computed, it is dispatched to the various joints composing the eyes, head, and spine in order for each to contribute to the final posture. Finally, this displacement is propagated in time in order for the looking or looking away motions to be smooth, natural, and human-like.

4 Crowd Engine

As introduced in Section 3, our crowd engine is divided into 3 important elements: the variety and navigation components, that are pre-processed, and the real-time pipeline, which is run at each frame of the simulation.

Variety. To generate thousands of individuals, a naive approach is to design as many humans as there are people in the crowd. Obviously, such an approach is impossible, since it would require armies of designers, and an infinite memory. The common and more reasonable approach is to use *human templates*. A human template is a virtual human defined by its skeleton, its mesh, which is skinned to the skeleton, and its set of textures. To create large crowds, a small group of human templates are instantiated several times. For each instance, one texture is randomly chosen within the template's available set. Then, color and shape variety techniques are applied, so that instances of a same template and using the same texture are still different [6]. To generate many locomotion animations for each human template, we use a motion-capture-based locomotion engine [41, 42]. This allows us to obtain adapted locomotion cycles that will be used at runtime to animate the characters. Using an IK system, upper body variations are also applied to further vary the crowd: the characters are thus able to have a hand in their pocket, or next to their ear to make a phone call. Note that the generated animations are skeletal, *i.e.*, they can be applied to meshes that are deformed at runtime. In our crowd engine, we use such meshes only at the forefront of the camera, for the animation process is very costly. The second and third levels of detail we use, *i.e.*, static meshes and impostors, are pre-computed and saved in the database too.

Navigation. Given an environment geometry, a navigation graph can be computed [39, 40]: the vertices of the graph represent circular zones where a pedestrian can walk freely without the risk of colliding with any static object composing the environment. Graph edges represent connections between vertices. In the environment, they are viewed as intersections (or gates) between two circular zones (vertices). From a navigation graph, path requests can be issued from one vertex to another. Using an algorithm based on Dijkstra's, we are able to devise many different paths that join one point of the environment to another one. It is possible to provide the navigation graph with a second model of the environment, which usually has a much more simple geometry, annotated with information. This second model is automatically analyzed, its meta-information retrieved, and associated to the corresponding vertices. Using this technique allows to make virtual humans look at specific points in the environment, for instance. We further detail this technique in Section 5.

Real-Time Pipeline. The first stage of our pipeline is the Scaler. The Scaler receives in input the navigation graph, and the view frustum of the camera. Based on this data, each vertex of the graph, and thus each pedestrian situated inside this vertex, is provided with two scores. The first one determines which navigation algorithm should be used for simulating the characters, but is beyond the scope of this paper. The second score determines which level of detail (LOD) will be used to render the characters in the vertex: deformable mesh (skeletal animation skinned at runtime), static mesh (pre-computed animations of the same mesh), or impostors (2D images representing frames of animation of the character). The second stage, the simulator, employs the first score to make each virtual human progress on its path. The Animator uses the second score to correctly update each pedestrian's position:

– If the LOD is the highest, the Animator needs to update the skeleton posture of human instances. The skinning of the mesh is later achieved in a shader, at the Renderer phase. With this high-level representation, it is also possible to add more detailed animations, that can be either pre-computed, or procedural. Procedural gaze variations are inserted in the Gaze part of the pipeline, after the Animator, but only to those pedestrians having the highest LOD. We further detail how this is done in the next Section.

– If the LOD is lower (static mesh or impostor), the job of the Animator is much different: it consists in retrieving the next frame of animation of the LOD, *i.e.*, a static and already deformed mesh, or a 2D image of the same posture, and send it to the Renderer. Note that with such LOD, it is impossible to provide humans with procedural animations. The gaze system presented below is thus limited to deformable meshes. This limitation is however not too troubling, because characters using these representations are farther from the camera, and thus, won't be visible since they will be either hidden by other characters or too far away for the user to be able to see their eyes. Moreover, it is possible to define the range from the camera at which the highest LOD will remain.

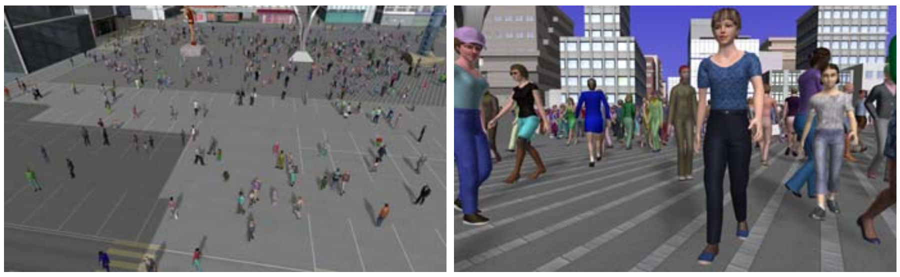
Fig. 4. Various examples depicting the crowd simulation engine at work.

The last stage of the pipeline is the Renderer. It takes care of rendering the environment and the crowd, as well as their respective shadows. Once again, the rendering process for a human instance varies, depending on the LOD it uses. If the representation is a deformable mesh, then the skeleton posture is sent to the GPU, where the deformation of the mesh will be achieved. If it is a static mesh, the deformation has been pre-computed and the work is limited to rendering the vertices of the mesh. Finally, in the case of impostors, the shader receives the correct coordinates of the texture that should be mapped onto two triangles.

Figure 4 depicts the results obtained with this crowd engine. It enables us to realistically simulate hundreds of characters in real-time.

5 Gaze Behaviors For Crowds

In the previous section, we explained the overall crowd engine pipeline. In the present section, we explain how we adapt the character motions to obtain the desired attentional behaviors. As mentioned, this motion adaptation takes place at the end of the Animator part of the pipeline, in the Gaze module depicted in Figure 2.

5.1 Interest Points

The first step is to define where and when each character should look. As mentioned in Section 3, the method to do this varies depending on the setup and the results we want to obtain.

The first is to use the meta-information present in the environment model as described in Section 4. This meta-information allows to describe the various environmental elements as "interesting" to look at. If this is the case, they will attract character attention. Characters will perform the gaze behavior in proximity of these elements and as long as they are in their field of view. These types of interest points are always at the same position in the scene; they are static elements.

The second method we use to define interest points is by assigning them to other characters in the scene. We may assign different levels of interest to different characters. In this way, some of them will attract attention more than others. As for the environmental elements, characters will attract the attention of other characters when in their proximity and as long as they remain in their field of view. These types of interest points are moving entities; they are therefore dynamic.

Finally, the third type of interest point is the user. A person using our system may be tracked using a coupled head- and eye-tracker. In this way, we can track the user position and gaze in the crowd engine 3D space. In this case, the user will be the interest point. Moreover, since we can track the user's gaze, the user's interest points (where he looks at) may become interest points for the characters in the environment. They will thus seem to unconsciously imitate the user. They will seem to try and find out what the user is looking at.

5.2 Motion adaptation

In order to obtain convincing results, we then have to adapt the character motions in order for them to look at the interest points. Since the characters and the interest points may be dynamic, we have to compute the joint displacements to be applied to the base motion at each timestep.

The skeletons we use are composed of 86 joints. Our method adjusts 10 of them: 5 spinal cord joints, 2 cervical joints, 1 head joint and 2 eye joints, in order for the characters to align their gaze to the interest points. By considering only this subset of the full skeleton, we greatly reduce the complexity of our algorithm. This allows us to have very small computational times and thus to animate a large number of characters.

Our method consists of two distinc phases. The first one computes the displacement maps to be applied to the various joints in order to satisfy the gaze constraints. We name this *spatial resolution*. The second component is the *temporal resolution*. This allows to propagate the displacement maps over an automatically defined number of frames onto the original motion. We thus ensure a smooth and continuous final movement.

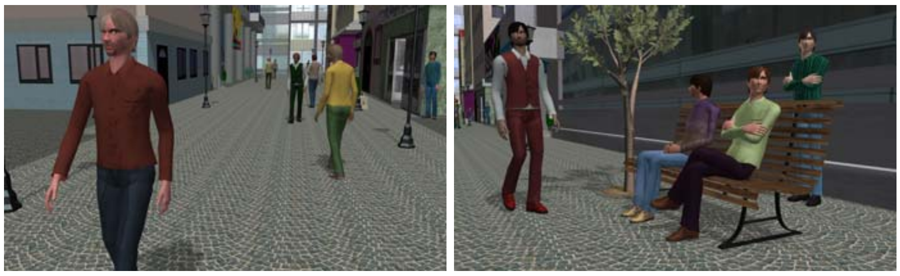

Fig. 5. Various examples depicting the types of possible gaze behaviors.

Spatial Resolution The purpose of the spatial resolution is to find a displacement map that modifies the initial motion in order to satisfy a given gaze contraint At each timestep, and for each deformable mesh in the crowd, if there is an active constraint, we launch an iterative loop starting with the bottom of the kinematic chain (lumbar verterbrae) and ending with the top of the kinematic chain (the eyes). At each iteration, we calculate the total remaining rotation to be done by the average eyes position (global eye) to satisfy the constraint. However, since the eyes are not the only joints to adjust, we dispatch this rotation to the other recruited joints.

To determine the contribution of each joint to the complete rotation we take inspiration from the work of [43]. The authors proposed a set of formulae to define the angle proportions to be assigned to the spine joints depending on the type of rotation to be done (pitch, yaw or roll). We use the formula they propose for the linearly increasing rotation distribution around the vertical axis. In our model, the rotations around the sagittal and frontal axes are very small in comparison to those in the vertical axis. We therefore keep the same formula for all types of rotations:

$$c_i = (-i - n)(\frac{2}{n(n-1)}) \quad i = 1...9 \tag{1}$$

where n is total number of joints through which to iterate and i is the joint index with $i = 1$ the lowest lumbar vertebra and $i = 9$ the global eye. At each step, c_i determines the percentage of *remaining* rotation to be assigned to joint i. The total rotation to be done by each joint for the character to satisfy the constraint may then by calculated by spherical linear interpolation (slerp) using these contribution values.

The remaining rotation to be done by each eye joint is then computed in order for them to converge on the interest point. Moreover, for interest points in the 30° composing the central foveal area, only the eye joints are recruited. For the 15° farther on each side composing the central vision area, only the eye, head, and cervical joints are recruited. Small movements therefore do not recruit the heavier joints.

Temporal Resolution The time it takes to look at something or someone depends, amongst other things, on where it is placed in the field of view. It will take less time to perform the gaze movement to look at something just in front than at something in the periphery.

When initiating a gaze movement, we therefore define how long it will take to perform it. We first define the upper and lower bounds for the movement duration. The lower bound is set to 0, which corresponds to no movement, i.e. the character already facing the interest point. The upper bound is set to 2 seconds, which corresponds to a 180° movement. We then simply interpolate between the two to obtain the duration of the gaze movement.

Moreover, this duration is different if considering the eye joints, the head and cervical joints, or the joints composing the remainder of the spine. The duration for the head to move is 2 times smaller than for the spine. Similarly, the duration for the eyes to converge is 5 times smaller than for the head. In this way, we allow for the lighter joints to move more rapidly than the others. The eyes therefore converge on the interest point well before any of the other joints attain their final posture. Our final movement therefore allows for the eyes to converge on the interest point and then recenter with respect to the head as the remainder of the joints move to satisfy the constraint.

As previously mentioned, some of the gaze constraints may be dynamic, i.e. in the case where the constraint is either another moving character or the system user. We therefore recompute the displacement map to satisfy the constraint at each timestep. We can assume that the constraint's position from one frame to the next one does not change much. We therefore recompute the rotation to be done at each frame but maintain the total contribution c_i to apply which we calculated before the initiation of the gaze motion. However, we reset the contributions to 0 if the gaze constraint changes. More specifically, when the current constraint location is farther than a pre-determined threshold from the constraint location at the previous frame. The newly calculated rotations to be performed by the joints to attain the new constraint's position are then distributed over the appropriate number of frames.

Finally, if the gaze behavior is deactivated, either because the character has been looking at a point of interest for too long or if there are no more interest points, the character will look back in front of him.

The added gaze motions strongly increase the human-like behaviors of crowd characters. By adding this functionality to the crowd simulation engine, we obtain a crowd of characters which seem to be aware of their environment and of other characters, as depicted in Figure 5.

6 Conclusion

In this paper, we have described how to improve the realism of a crowd simulation by allowing its pedestrians to be aware of their environment and of the other characters present in this environment. They can even seem to be aware of a user interacting with this environment. We have explained the various setups which allow for crowd characters to gaze at environment objects, other characters or even a user. Finally, we have described a method to add these attentional behaviors in order for crowd characters to seem more individual.

Acknowledgements

The authors would like to thank Mireille Clavien for the design of characters and environments. They would also like to thank the Swiss National Research Foundation for sponsoring this research.

References

1. Ryder,G. and Day,A. M.: Survey of Real-Time Rendering Techniques for Crowds. Computer Graphics Forum, vol. 24 no. 2, pp. 203–215, 2005.
2. Simon Dobbyn, John Hamill, Keith O'Conor, and Carol O'Sullivan: Geopostors: a real-time geometry / impostor crowd rendering system. In: SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games, pp. 95–102, 2005.
3. Erik Millan and Isaac Rudomin: Impostors and pseudo-instancing for GPU crowd rendering. In: GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, pp. 49–55, 2006.
4. L. Kavan, S. Dobbyn, S. Collins, J. Zara, and C. O'Sullivan: Polypostors: 2D polygonal impostors for 3D crowds. In 2008 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 149–155, 2008.
5. F. Tecchia, C. Loscos, and Y. Chrysanthou: Image-based crowd rendering. IEEE Computer Graphics and Applications, vol. 22, no. 2, pp. 36–43, 2002.
6. J. Maïm, B. Yersin, and D. Thalmann: Unique Instances for Crowds. To Appear in IEEE Computer Graphics and Applications, 2009.
7. R. McDonnell, M. Larkin, S. Dobbyn, S. Collins, and C. O'Sullivan: Clone attack! Perception of crowd variety. ACM Transactions on Graphics, vol. 27, no. 3, pp. 1–8, 2008.
8. C. W. Reynolds: Flocks, Herds, and Schools: A Distributed Behavioral Model. In: SIGGRAPH '87: Proceedings of the 14th International Conference on Computer Graphics and Interactive Techniques, vol. 21, no. 4, pp. 25–34, 1987.
9. C. W. Reynolds: Steering Behaviors for Autonomous Characters. In: Game Developers Conference, 1999.
10. D. Helbing and P. Molnar: Phys. Rev. E51, 4282 (1995).
11. A. Treuille, S. Cooper, and Zoran Popović: Continuum Crowds. ACM Transactions on Graphics, vol. 25, no. 3, pp. 1160–1168, 2006.

12. Musse, S. R., and Thalmann, D.: A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis Computer Animation and Simulation, In: Proc. Workshop of Computer Animation and Simulation of Eurographics97, pp. 39–51, 1997.
13. N. Pelechano, J.M. Allbeck, and N.I. Badler. Controlling individual agents in high-density crowd simulation. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, 2007.
14. Lerner, A., Chrysanthou, Y., and Lischinski, D.: Crowds by example. Computer Graphics Forum (Eurographics 2007) vol. 26, no. 3, pp. 655–664, 2007.
15. S. Paris, J. Pettré, and S. Donikian: Pedestrian steering for crowd simulation: A predictive approach. Computer Graphics Forum, vol. 26, no. 3, pp. 665–675, 2007.
16. Lee, K. H., Choi, M. G., Hong, Q., and Lee, J.: Group behavior from video: a data-driven approach to crowd simulation. In: Proceedings of the 2007 ACM SIGGRAPH Eurographics symposium on Computer animation, pp. 109–118, 2007.
17. B. Yersin, J. Maïm, F. Morini, and D. Thalmann: Real-Time Crowd Motion Planning: Scalable Avoidance and Group Behavior. The Visual Computer Journal, vol. 24, no. 10, pp. 859-870, 2008.
18. Kuffner, J.J. Jr., Latombe, J.-C.: Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans. In: Proceedings of Computer Animation, pp. 118–127, 1999.
19. Peters, C., O'Sullivan, C.: Synthetic Vision and Memory for Autonomous Virtual Humans. Computer Graphics Forum, vol. 21, no. 4, pp. 743–752, 2002.
20. Hill, R.: Modeling Perceptual Attention in Virtual Humans. In: Proceedings of Computer Generated Forces and Behavioral Representation, 1999
21. Chopra Khullar, S., Badler, N.I.: Where to Look? Automating Attending Behaviors of Virtual Human Characters. Autonomous Agents and Multi-Agent Systems, vol. 4. no. 1–2, pp. 9–23, 2001.
22. Peters, C., Pelachaud, C., Bevacqua, E., Mancini, M., Poggi, I.: A Model of Attention and Interest Using Gaze Behavior. Lecture Notes in Computer Science, pp. 229–240, 2005.
23. Gu, E., Badler, N.: Visual Attention and Eye Gaze During Multiparty Conversations with Distractors. Lecture Notes in Computer Science, pp. 193–204, 2006.
24. Lance, B., Marsella, S.: Emotionally Expressive Head and Body Movement During Gaze Shifts. Lecture Notes in Computer Science, pp. 72–85, 2007.
25. Itti, L., Dhavale, N., Pighin, F.: Realistic Avatar Eye and Head Animation Using a Neurobiological Model of Visual Attention. In: Proceedings of the Symposium on Optical Science and Technology, vol. 5200, pp. 64–78, 2003.
26. Peters, C., O'Sullivan, C.: Bottom-up visual attention for virtual human animation. In: Proceedings of Computer Animation and Social Agents, pp. 111–117, 2003.
27. Marchand, E., Courty, N.: Controlling a camera in a virtual environment. The Visual Computer, vol. 18, no. 1, pp. 1–19, 2002.
28. Kim, Y., Hill, R.W. Jr., Traum, D.R.: A Computational Model of Dynamic Perceptual Attention for Virtual Humans. In: Proceedings of Behavior Representation in Modeling and Simulation, 2005.
29. Lee, S.P., Badler, J.B., Badler, N.I.: Eyes alive In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 637–644, 2002.
30. Lee, S.-H., Terzopoulos, D.: Heads up!: biomechanical modeling and neuromuscular control of the neck. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 1188–1198, 2006.

31. Badler, N. I., Korein, J. D., Korein, J. U., Radack, G. M., Shapiro Brotman, L.: Positioning and animating human figures in a task-oriented environment. The Visual Computer, vol. 1, no. 4, pp. 212–220, 1985.
32. Tolani, D., Goswami, A., Badler, N. I.: Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs. Graphical models, vol. 62, no. 5, pp. 353–388, 2000.
33. Lee, J., Shin, S. Y.: A hierarchical approach to interactive motion editing for human-like figures. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 39–48, 1999.
34. Kovar, L., Schreiner, J., Gleicher, M.: Footskate cleanup for motion capture editing. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 97–104, 2002.
35. Shin, H. J., Lee, J., Shin, S. Y., Gleicher, M.: Computer puppetry: An importance-based approach. ACM Transactions on Graphics, vol. 20, pp. 67–94, 2001.
36. Kulpa, R., Multon, F., Arnaldi, B.: Morphology-independent representation of motions for interactive human-like animation. In: EURORAPHICS 2005, vol. 24, no. 3, pp. 343–352, 2005.
37. Choi, K.-J., Ko, H.-S.: Online motion retargetting. The Journal of Visualization and Computer Animation, vol. 11, no. 5, pp. 223–235, 2000.
38. Le Callennec, B., Boulic, R.: Interactive motion deformation with prioritized constraints. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 163–171, 2004.
39. J. Pettré, P. de Heras Ciechomski, J. Maïm, B. Yersin, J.-P. Laumond, and D. Thalmann: Real-time navigating crowds: scalable simulation and rendering. Computer Animation and Virtual Worlds, vol. 17, no. 34, 445–455, 2006.
40. J. Pettré, H. Grillon, and D. Thalmann: Crowds of Moving Objects: Navigation Planning and Simulation. In: Proceedings of IEEE International Conference on Robotics and Automation, 2007.
41. P. Glardon, R. Boulic, and D. Thalmann: PCA-based walking engine using motion capture data. In: Proc. of Computer Graphics International, 2004.
42. P. Glardon, R. Boulic, and D. Thalmann: A coherent locomotion engine extrapolating beyond experimental data. In: Proc. of Computer Animation and Social Agent, 2004.
43. Boulic, R., Ulicny, B., Thalmann, D.: Versatile Walk Engine. Journal of Game Development, vol. 1, no. 1, pp. 29–43, 2004.