The Visual Computer manuscript No.
(will be inserted by the editor)

Parag Chaudhuri · George Papagiannakis · Nadia Magnenat-Thalmann

Self Adaptive Animation based on User Perspective

Abstract In this paper we present a new character animation technique in which the animation adapts itself based on the change in the user's perspective, i.e., when the user moves, so that their point of viewing the animation changes, then the character animation adapts itself, in response to that change. The resulting animation, generated in real-time, is a blend of key animations provided a priori by the animator. The blending is done with the help of efficient dual-quaternion transformation blending. The user's point of view is tracked using either, computer vision techniques or a simple user-controlled input modality, such as mouse-based input. This tracked point of view is then used to suitably select the blend of animations. We show a way to author and use such animations in both Virtual as well as Augmented Reality scenarios and demonstrate that it significantly heightens the sense of presence for the users when they interact with such self adaptive animations of virtual characters.

Keywords self-adaptive character animation · animation blending · augmented and virtual reality

1 Introduction

Over the years, character animation has developed into a very mature art form. However, virtual characters are still, far less convincing when they have to interact with the user, especially in real-time. They are often inert to the user's presence or do not react appropriately. This is a very important aspect in computer games and interactive virtual and mixed reality applications. Therefore, a lot of effort is being invested in creating expressive virtual characters [33].

We present, in this paper, a simple and fast method to author *self adaptive character animations* that respond automatically to changes in the user's perspective or point of view in real-time. The animator creates a set of example key animations for the characters assuming the user is viewing the animation from different key viewpoints in the world. When the user actually interacts with the character, the user's actual point of view is tracked in real-time by using computer vision techniques or by simple user controlled input methods. The tracked position of the user's viewpoint, with respect to the key viewpoints, is then used to blend the example key animations, in real-time. Thus, the animation of the character adapts itself in response to the changes in the user's viewpoint.

We will demonstrate that our method is simpler and more efficient than other techniques that can be used to obtain similar results [27], [10]. We will also show a working, prototype implementation of our method with simple examples in Virtual and Augmented Reality.

We begin by providing the related background work in Section 2. Next, we present our method for authoring self adaptive animations in Section 3. This is followed by a discussion of some lessons learnt during implementation of the introduced technique and an analysis of its performance in Section 4. Section 5 concludes with a discussion of future work that can be done.

MIRALab, University of Geneva
Switzerland.
E-mail: parag, papagiannakis, thalmann @miralab.unige.ch

2 Background

In this section, we explore related prior work on control and blending of character animation, followed by related background work on presence and interaction in Augmented Reality.

2.1 Control and blending of character animation

Many existing works represent the set of plausible character poses as an abstract space and use it to generate the animation [18], [23]. Igarashi et al. [14] present spatial keyframing where the user controls a character by adjusting the position of a control cursor in the 3D space and the pose of the character is given as a blend of nearby key poses. However, there is no concept of a view or camera involved in this work. The first work to use the camera to control the pose of a character was View-Dependent Geometry [27]. In this

work, the animator manually matches the view direction and the shape of a base mesh model with the sketched poses of the character and creates a view-sphere. Tracing any camera path on this view-sphere generates the appropriate animation with view-dependent deformations. Chaudhuri et al. [10] extend the above work to a generalized framework, for doing view-dependent animation from sketches and video. These methods, however, do not work in real-time and also involve a significant amount of manual work in generating the space of poses. Also, these methods do not associate animations to camera viewpoints, like we do, but only use a static pose-camera association. This imposes a serious restriction on this method that the animation is generated only when the camera moves. Therefore, none of these techniques can be used to generate real-time self adaptive character animation.

We blend between a set of example key animations provided a priori by the animator. Example based animation research has taken many directions, like motion recovery from video with/without additional human motion databases [6], [28]; resequencing motion capture data [19], [17], [2] and performance driven animation [30], [7]. We use a dual quaternion based transformation blending technique, as introduced by [16] for performing character skinning, to blend between multiple example animations in real-time.

The blending is controlled by tracking the position of the rendering camera in a subspace of key cameras. We can use simple mouse based input to control the user's viewpoint to create self adaptive animations in Virtual Reality (VR) or use real-time camera tracking using computer vision techniques, to create such animations in Augmented Reality (AR). We discuss the related work pertaining to animated virtual characters in AR in the next section.

2.2 Animated virtual characters in augmented reality

Virtual characters have been used to enrich AR experiences in a lot of different ways, for e.g., as collaborative game partners [3], [26]; as training assistants for repairing machinery [5] or to bring ancient civilizations back to life [25]. Some virtual characters are designed to function as autonomous software agents using the belief, desire and intention model for agents [5], while others respond to changes in the real world, done by the user [4]. However, these examples do not allow for *mutual persistent presence* between real users and virtual characters, since the virtual characters are not capable of sensing the real users, looking at them and thus establishing a more advanced conscious relationship [12]. It is known that non-verbal communication (for e.g., via gaze or gesture) between the virtual character and the user enhances the expressiveness of the virtual character and improves the sense of presence for the user in the AR environment [33], [34]. Self adaptable animation is able to adapt the character animation in real-time and react to the user appropriately.

Interactive, autonomous virtual characters have also been used in storytelling [9] or for modeling individual behaviour such as pedestrian behaviour in multi-human simulations [29]. In these works, a procedural system is used for modeling the cognitive and motor behaviour of the character based on pre-recorded data or artistic intent. These methods do not extend directly to AR or mixed reality environments. Our method is complimentary to such methods and can be used in conjunction with them to create more interactive characters.

Though it may be argued by some that similar looking effects can already be seen in games, we have not found any principled approach to creating such self-adaptive characters in existing literature. Since we do not know the algorithms or techniques employed in these games, other than those that can be obtained by modifying some of the methods mentioned in prior reported work, we can claim that our method is simpler, more efficient and robust.

Contributions: We present a novel character animation technique that uses fast real-time animation blending based on the user's point of view, to create animation that automatically adapts itself in response to tracked changes in the user's perspective. This enhances the user's sense of presence when interacting with such characters, both in virtual and augmented reality environments.

3 Self Adaptive Animations

We start by explaining the notation we use to explain our method of self adaptive animation.

3.1 Notation

A character pose can be defined as a hierarchical tree of rigid transformations. If we linearize this tree by doing a fixed traversal on it, we get a list of transformations. Rigid transformations can be represented as unit dual quaternions [11].

A dual quaternion, $\hat{\mathbf{q}}$ can be thought of as a sum of two normal quaternions, $\hat{\mathbf{q}} = \mathbf{q_0} + \varepsilon\mathbf{q}_\varepsilon$, where $\mathbf{q_0}$ is the real part and $\mathbf{q}_\varepsilon$ is the dual part. ε is the dual unit, satisfying $\varepsilon^2 = 0$. The set of unit dual quaternions, $\mathbf{Q_1}$, is a manifold in 8-dimensional Euclidean space (called an image-space of dual quaternions [22]). A unit dual quaternion with the dual part as zero represents a pure rotation. Also, a unit dual quaternion, $\hat{\mathbf{t}} = 1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)$, represents a translation by the vector (t_0, t_1, t_2) with i, j, k being the usual quaternion units. Hence, if the rotation is given by a quaternion, $\mathbf{q_0}$, then the complete rigid transformation can be written down as a composition (i.e., multiplication) of the two dual quaternions as

$$(1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k))\mathbf{q_0} = \mathbf{q_0} + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)\mathbf{q_0} \qquad (1)$$

It can be proved that the composition always yields a valid unit dual quaternion. The rotation and the translation can thus be recovered from a given unit dual quaternion.

One of the advantages of using the dual quaternion notation is that we have a single representation for rigid transformations. So while blending animations, there are no special

cases to be handled for joints that only rotate versus joints that can rotate and translate, including the root joint.

We define a character pose, $P = \{\hat{\mathbf{q}}_m\}$, as a list of dual quaternions, with m as the index of the list. An animation is a time varying sequence of poses and hence can be defined as, $P(t)$. An animator creates a set of $1 \leq k \leq K$ *key animations*, $P^k(t)$, that represent the way the character should react when the user approaches or looks at the character from different directions. We refer to these directions as *key viewpoints* or *key cameras* and represent them as v^k. We will now describe, with the help of an example, how we author such self adaptive character animations.

3.2 Authoring self adaptive animations

In this example, the animator decides that when the user approaches the character from different directions the character should turn around, look at the user and gesture toward the user. So the animator provides us with three (i.e., $K = 3$) different key animations in which the character is looking in different directions while performing some actions (see Figure 1). We associate one key camera with every animation (see Figure 2). In Figure 2, the small green sphere represents the key camera associated with the corresponding pose shown. It is always in the center of view in the figure because when the key camera is added, the current rendering camera is the same as the key camera.

Fig. 1 The input key animations

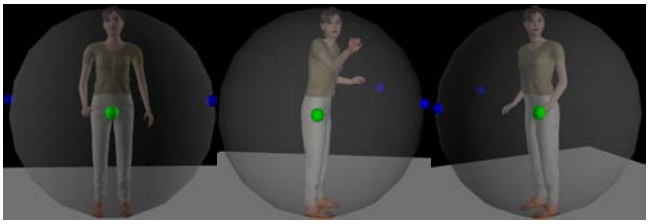

Fig. 2 The input key cameras

This creates a subspace of key cameras, v^k, such that each camera has one animation associated with it. It is a subspace of the space of all the view directions from which the character can be looked at. Since we are only interested in the direction of viewing, this space can be represented by a unit sphere around the character. This is shown as the large transparent sphere in Figures 2 and 3. All the key cameras, shown as small green and blue spheres in Figure 2, lie on this unit sphere. This is similar to (though not the same as) the view-space used in [10]. There the sphere is around each character pose and the complete space is an aggregation of these spheres, while we are only interested in finding out the direction of the user vis-a-vis the character, so one unit sphere around the character is sufficient (see Figure 3). This sphere moves with the character if the character changes position, so that the relative location of the user can always be figured out. Our formulation is simpler and more apt for real-time use.

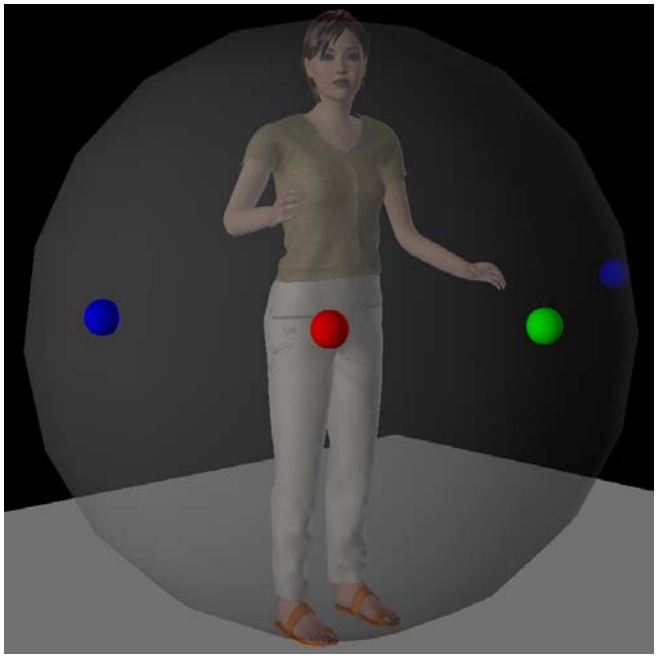

Fig. 3 The key cameras are shown in blue and green. The current camera is shown in red. The large sphere represents the space of all the directions from which the character can be viewed.

Once we have associated the cameras to the animations, then the character is ready to interact with the user. Note that this association has to be done only once and can be stored for future use. When the user moves around the character, we track the user's viewpoint and project it onto the unit sphere. This is referred to as the current camera or viewpoint, v_c. We then compute a set of convex weights, w^k, based on the distances between the current camera and the key cameras (see Equation 2). For the K input key animations, if the distance between v_c and v_k is given by $D(v_c, v_k)$, then

$$w^k = \frac{[1/D(v_c, v_k)]^\alpha}{\sum_{k=1}^{K}[1/D(v_c, v_k)]^\alpha} \tag{2}$$

where α generally ranges from 2 to 4. Lower values of alpha mean that the influence of the respective key camera is spread over a larger area, while larger values of alpha tend to focus the influence in closer vicinity of the key cameras.The

distance function, D, can be the geodesic distance between the cameras computed on the surface of the unit sphere, or it can be approximated by the Euclidean chordal distance between the cameras. Then the current character animation seen by the user is given by

$$P(t) = BLEND_k(P^k(t), w^k) \tag{3}$$

where $BLEND_k$ is a blending function that blends the corresponding dual quaternions of each animation, at that particular time. We can use either dual quaternion linear blending (DLB) or dual quaternion iterative blending (DIB) to blend the actual dual quaternions [16]. Given the list of the K dual quaternions to be blended, $\hat{\mathbf{q}}_m^k$, and the set of convex weights, w^k, the DLB is given by

$$DLB(\hat{\mathbf{q}}_m^k, w^k) = \frac{\hat{\mathbf{q}}_m^1 w^1 + \ldots + \hat{\mathbf{q}}_m^K w^K}{\|\hat{\mathbf{q}}_m^1 w^1 + \ldots + \hat{\mathbf{q}}_m^K w^K\|} \tag{4}$$

The linear blend is fast because it is a closed-form solution, however, it is not precise. For a more precise solution, we employ the DIB algorithm (as given in Algorithm 1).

Algorithm 1 Dual Quaternion Iterative Blending

Require: The input list of dual quaternions , $\hat{\mathbf{q}}_m^k$ and the convex weights, w^k.
Require: The desired precision, p.
1: Compute $\hat{\mathbf{b}} = DLB(\hat{\mathbf{q}}_m^k, w^k)$
2: **repeat**
3: $\hat{\mathbf{x}} = \sum_{k=1}^{K} w^k \log(\hat{\mathbf{b}}^* \hat{\mathbf{q}}_m^k)$ //$\hat{\mathbf{b}}^*$ is the conjugate of $\hat{\mathbf{b}}$
4: $\hat{\mathbf{b}} = \hat{\mathbf{b}} \exp(\hat{\mathbf{x}})$
5: **until** $\|\hat{\mathbf{x}}\| < p$
6: The output blended quaternion is $\hat{\mathbf{b}}$

In practice, both methods can be used for real-time scenarios. Though the DLB is a very good approximation to the precise blend solution, it has been shown [16] that the iterative DIB algorithm is the best choice when blending more than two rigid transformations. It is so because the DIB preserves rigidity, it shows coordinate system invariance, the blending is constant speed i.e., the derivatives of both the interpolated rotation and translation are constant and it computes the blend along shortest path. For pure rotations and/or translations, the methods can be degraded gracefully to simpler solutions.

Blending rigid transformations has been a long studied problem, however, blending multiple (more than two) rigid transformations has very few solutions that can work in real-time scenarios. SLERP or Spherical Linear Interpolation [31] can be used to blend between only two quaternions. Other solutions that can handle multiple transformations, require decomposition into component translations and rotations that make the method coordinate system dependent [8] or they are computationally less efficient [1]. If the component translation vectors and rotation quaternions are already available then the dual quaternion blending is very easy to do and more efficient in cases when both of these transformations exist. Hence, dual quaternion blending is a very good choice for blending multiple animations. In our method, it performs very well for creating the self-adaptive character animations. A more detailed description of the dual quaternion algebra and blending techniques is out of the scope of this paper, however, the interested reader can find the details in [11], [22] and [16].

Fig. 4 Self adaptive animation in VR : the character reacts to the user's movement

Therefore, as the user moves, the animation is generated instantaneously based on the position of the user's viewpoint. In this particular example, when the user changes the view direction using the mouse, the character turns to look at and gesture toward the user (see Figure 4). This is a typical VR scenario, and we see that the blending works very nicely to adapt the animation to react to the user's movement. An obvious advantage we have over other gaze tracking solutions, that can also be used to perform such tasks, is that the jitter in the movement of the user is completely damped during the blend. So the character does not jerk unexpectedly. Also, all the poses of the character are automatically, kinematically correct as long as the input key animations are kinematically correct (i.e., all joint transformations are valid and within limits). Since we blend the transformation hierarchies of the character (i.e., kinematic or skeleton level blending), then if the skinning of the character is derived from the skeleton joint positions, it does not have to be blended separately. This is a big advantage in terms of efficiency. Earlier methods, [27] and [10], blend the mesh vertices directly and hence are much slower. They also cannot guarantee the kinematic validity of the resulting blended characters.

We can also associate multiple key cameras with one key animation. This is done using a many-to-one map from the cameras to the animations. The animator can thus, use very few input key animations to create the self adaptive animations. We can also blend static poses in addition to animations.

4 Implementation

Dual quaternions are not the de facto standard to represent transformations in current graphics libraries. Therefore, it is not obvious how to combine dual quaternion based blending with an existing real-time graphics architecture. We discuss in this section how we efficiently implement self adaptive animations in a real-time setting.

4.1 Self adaptive animation in VR

We use OpenSceneGraph (OSG) [24] to manage our characters and the scene. OSG represents the characters as a hierarchical tree of transform nodes. Animations are represented as a sequence of rigid transformations, called animation paths, attached to every transform node of the character. The transform changes, as per the animation paths, to generate the animations. We first convert the key animations, to lists of dual quaternions by parsing all their animation paths. This involves a one-time conversion of all the transformation matrices, representing the input key animations, to dual quaternions. Then for the character in the final scene, we attach an update callback to every transform node. OSG performs an update pass, when all update callbacks are processed, before performing a render pass for every frame of the animation. The update callback updates every transform node of the character (see Algorithm 2).

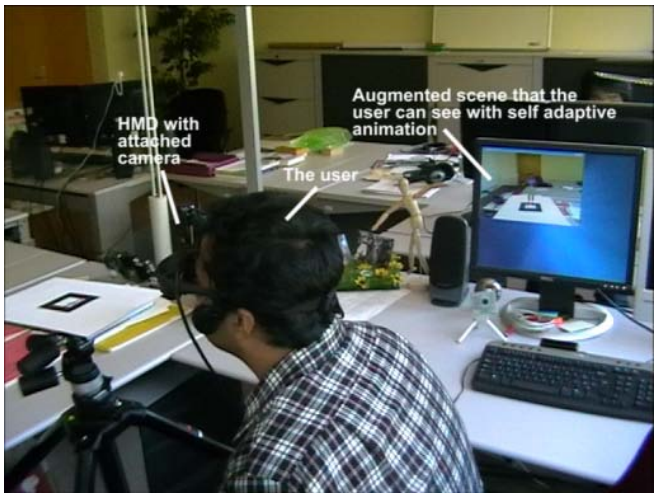

Fig. 5 The user with a camera mounted on the HMD looks at the scene with the marker.

Algorithm 2 Algorithm for the OSG update callback

Require: Input key cameras, v^k and associated key animations, $P^k(t)$.
Require: A camera manipulator that controls the current camera.
1: Get the position of the current camera, v_c, from a camera manipulator.
2: Based on v_c and the set of key viewpoints, v^k, compute the set of convex weights, w^k, for all $1 \leq k \leq K$ (using Equation 2).
3: Compute a blend of the respective dual quaternions from the input key animations using Equation 3, to get the blended dual quaternion.
4: Use the blended dual quaternion to update the transformation node to which this update callback is attached.

The last step of the update callback involves the only conversion from dual quaternion to matrices (the conversion uses Equation 1) in this algorithm. This algorithm thus ensures that the conversion happens only once for each transformation node of the character. This also abstracts out the mechanism of controlling the current camera position to a camera manipulator. For a VR scenario, a standard OSG trackball camera manipulator can be used to control the current camera with a mouse. In the next section, we describe how we can use input from real-time camera tracking to control the current camera used for blending in an AR scenario.

4.2 Self adaptive animation in AR

For a simple AR scenario, we use the ARToolkit [15] to track the current camera position with respect to the character. We recover the current transform for a tracked marker and place our character at that position. The pose of the marker is used to infer the position of the camera looking at the scene. This functionality is then encapsulated in a camera manipulator. The user wears a head mounted display (HMD) with a camera attached to it (see Figure 5).

Then, when the user moves looking at the marker, the orientation of the camera is obtained in real-time. The user can see the character in the scene, at the position of the marker. This is then used to blend the key animations, to generate the self adaptive animation. So the character responds to the movement of the user, looks at the user and gestures in the direction of the user. Note, that the camera position also changes due to relative movement between the user and the character, so rotating the marker is the same as the user moving and looking at the character from various directions (see Figure 6).

We can also do life size augmentation of a scene, where we use a markerless tracking solution (see Figure 7). We first record a video of the scene and extract feature point correspondences from the video. We create a database of these tracked features. Then in real-time, we track feature points in the scene by matching them to the database of stored feature points. A stratified, metric reconstruction from the computed feature correspondences gives the camera matrix, **P**, for the camera mounted on the HMD [13] [32]. The 3×4 projective camera matrix, **P**, is decomposable as $\mathbf{K}[\mathbf{R}|\mathbf{t}]$, where **K** is a 3×3 matrix containing the focal length of the camera, **R** is a 3×3 submatrix controlling the view direction, and **t** is a 3×1 submatrix governing the viewpoint distance. The camera-to-world transformation is given by the **R** and **t** matrices. These can be used to place the character in the scene. The camera center, **C** can be recovered as the right null space of **P** by solving $\mathbf{PC} = 0$. This camera center is then used to control the self adaptive animation. As the user moves around the character, she turns and looks at the user in response. In addition, the character also gestures toward the user. Hence, the complete body animation of the character can be adapted according to the user's perspective.

4.3 Performance Analysis

In this section, we present some performance indicators for our algorithm. Our hardware configuration is a 2.4Ghz Core

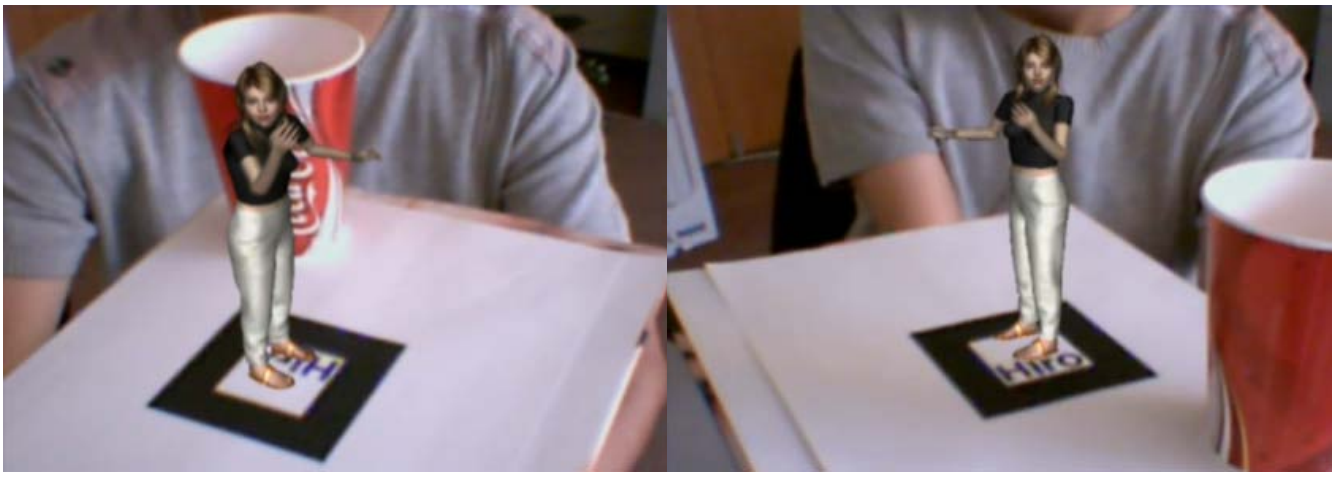

Fig. 6 Self adaptive animation in AR.

Fig. 7 Self adaptive animation in markerless, life size AR - the character turns her head to look at the user.

2 Duo PC with a NVidia 8800 GTX card, Logitech QuickCam Express and Notebook Pro webcams, and an i-Glasses SVGA Pro HMD. In Table 1, we can see that the frames per second (*FPS*) for the animation without blending and that with blending in the VR scenario do not differ significantly. The *FPS* for the AR case is lesser because additional processing needs to be done for the video frames. It can also be seen that while the DIB shows a lesser frame rate than the DLB in both VR and AR, it is still suitable for real-time interaction with the character. The DIB precision used was 0.001. Character 1 is the character shown in Figure 6 with 25000 triangles in the skinning mesh and 27 joints in the articulation skeleton. Character 2 is a simpler mesh with 8009 triangles and 27 joints. Character 3 is the character shown in Figure 7 with 30236 triangles and 27 joints. The numbers shown for the AR case are for the marker based tracking case using ARToolkit. Though it is possible to implement dual quaternion blending in the GPU, we are not using any shaders in this prototype implementation.

The *FPS* statistics have been recorded using the OSG statistics counters. OSG performs update, cull and render passes for each frame of the animation. The update callbacks are called during the update pass. We can see in Figure 8 the update pass statistics of the no blending, VR and AR cases. We see that the time taken for the update pass are almost similar in all three cases, i.e., the self adaptive animation is only slightly more costly, in terms of efficiency, than normal animation. Since everything else is the same in all the cases, measuring the performance of the update pass is equivalent to measuring the performance of our technique, as that is where the adaptation occurs.

The solution is as scalable as any example based animation technique, as it requires the example key animations to create the self adaptive characters. Thus, our algorithms are efficient and help create characters that are more responsive and give the user an enhanced sense of presence in an immersive VR/AR scenario.

5 Conclusion

We have presented a simple and efficient method to create self adaptive character animation, in which the animation of the character automatically adapts itself to react to the changes in the user's perspective. We have used fast dual quaternion blending between example key animations based on position of the current camera, to achieve this in real-time. As a result, we can create more believable characters in VR and AR that can react, as required by the animator, to the changes in the real user's perspective.

An obvious extension to the current implementation is to implement the dual quaternion blending on the GPU and also, use the same blending algorithms for character skinning. The current technique uses the camera position to control the blend. It is possible to use other camera parameters, like distance from the character to control other characteristics of the blend, for e.g., levels of articulation detail. We can even think of using this simple method in conjunction with more complex methods like those presented for animation of conversational characters in [21] or for scripted interactive characters in [20], in order to create even more expressive characters in the future.

	# of Triangles, Joints	*FPS* (without blending)	VR Scenario		AR Scenario	
			FPS (DLB)	*FPS* (DIB)	*FPS* (DLB)	*FPS* (DIB)
Character 1	25000, 27	84	82	79	55	50
Character 2	8009, 27	135	132	130	80	77
Character 3	30236, 27	58	56	53	45	43

Table 1 Frames per second for the character animation

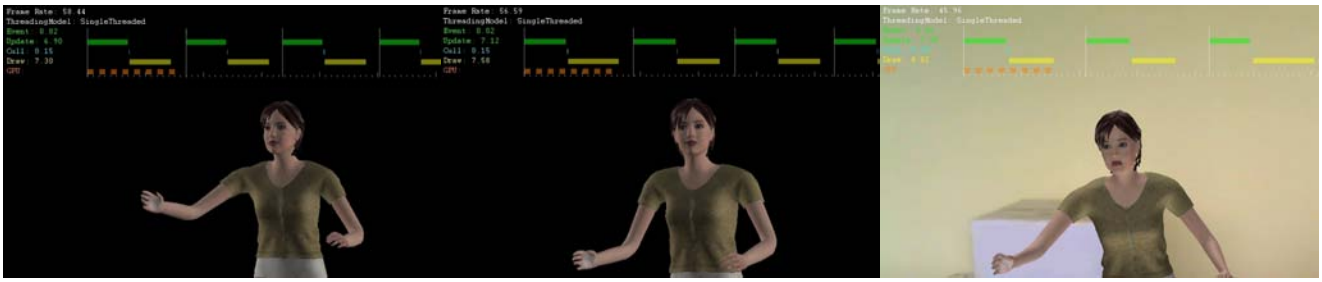

Fig. 8 From left to right: OSG update pass times for the no blending (6.90s), DLB blending in VR (7.12s) and DLB blending in AR cases (7.35s)

Acknowledgements We would like to thank Nedjma-Cadi Yazli for creating the example animations that were crucial to this work. The work done in this project is funded by the European research project INDIGO (IST-045388).

References

1. Alexa, M.: Linear combination of transformations. ACM Transactions on Graphics **21**(3), 380–387 (2002)
2. Arikan, O., Forsyth, D.A., O'Brien, J.F.: Motion synthesis from annotations. ACM Transactions on Graphics **22**(3), 402–408 (2003)
3. Balcisoy, S., Kallmann, M., Torre, R., Fua, P., Thalmann, D.: Interaction techniques with virtual humans in mixed environments. In: International Symposium on Mixed and Augmented Reality (ISMAR'01) (2001)
4. Barakonyi, I., Schmalstieg, D.: Augmented reality in the character animation pipeline. SIGGRAPH 2006 Sketches and Applications (2006)
5. Barakonyi, I., Schmalstieg, D.: Ubiquitous animated agents for augmented reality. In: ISMAR 2006 - IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 22–25 (2006)
6. Bregler, C., Malik, J.: Tracking people with twists and exponential maps. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society (1998)
7. Buck, I., Finkelstein, A., Jacobs, C., Klein, A., Salesin, D.H., Seims, J., Szeliski, R., Toyama, K.: Performance-driven hand-drawn animation. In: NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering, pp. 101–108 (2000)
8. Buss, S.R., Fullmore, J.P.: Spherical averages and applications to spherical splines and interpolation. ACM Transactions on Graphics **20**(2), 95–126 (2001)
9. Cavazza, M., Charles, F., Mead, S.J.: Character-based interactive storytelling. IEEE Intelligent Systems **17**(4), 17–24 (2002)
10. Chaudhuri, P., Kalra, P., Banerjee, S.: View-Dependent Character Animation. Springer-Verlag, London (2007)
11. Clifford, W.: Mathematical Papers. Macmillan (1882)
12. Encarnação, J., Gross, M., Reiner, M., Slater, M., Stork, A., Stricker, D., de Velde, W.V.: Presence and interaction in mixed reality environments. FET Proactive initiative (2004). Ftp://ftp.cordis.lu/pub/ist/docs/fet/pr2-37.pdf
13. Fitzgibbon, A., Zisserman, A.: Automatic camera tracking. In: M. Shah, R. Kumar (eds.) Video Registration, pp. 18–35. Kluwer Academic (2003)
14. Igarashi, T., Moscovich, T., Hughes, J.F.: Spatial keyframing for performance-driven animation. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 107–115. ACM Press (2005)
15. Kato, H., Human Interface Technology Laboratory: ARToolkit (2007). Http://artoolkit.sourceforge.net
16. Kavan, L., Collins, S., Zara, J., O'Sullivan, C.: Skinning with dual quaternions. In: 2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 39–46. ACM Press (2007)
17. Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. In: SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, pp. 473–482. ACM Press (2002)
18. Lewis, J.P., Cordner, M., Fong, N.: Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In: SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 165–172. ACM Press/Addison-Wesley Publishing Co. (2000)
19. Li, Y., Wang, T., Shum, H.Y.: Motion texture: A two-level statistical model for character motion synthesis. In: SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, pp. 465–472. ACM Press (2002)
20. Loyall, A.B., Reilly, W.S.N., Bates, J., Weyhrauch, P.: System for authoring highly interactive, personality-rich interactive characters. In: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 59–68. Eurographics Association (2004)
21. Masuko, S., Hoshino, J.: Head-eye animation corresponding to a conversation for cg characters. Computer Graphics Forum (Proceedings of Eurographics 2007) **26**(3), 303–312 (2007)
22. Mccarthy, J.M.: Introduction to theoretical kinematics. MIT Press, Cambridge, MA, USA (1990)
23. Ngo, T., Cutrell, D., Dana, J., Donald, B., Loeb, L., Zhu, S.: Accessible animation and customizable graphics via simplicial configuration modeling. In: SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 403–410. ACM Press/Addison-Wesley Publishing Co. (2000)
24. OSG: OpenSceneGraph 2.0 (2007). http:// www. openscenegraph. org
25. Papagiannakis, G., Schertenleib, S., O'Kennedy, B., Arevalo-Poizat, M., Magnenat-Thalmann, N., Stoddart, A., Thalmann, D.: Mixing virtual and real scenes in the site of ancient pompeii. Computer Animation and Virtual Worlds **16**(1), 11–24 (2005)
26. Piekarski, W., Thomas, B.: ARQuake: The outdoor augmented reality gaming system. Communications of the ACM (1), 36–38 (2002)

27. Rademacher, P.: View-dependent geometry. In: SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 439–446. ACM Press/Addison-Wesley Publishing Co. (1999)
28. Ren, L., Shakhnarovich, G., Hodgins, J.K., Pfister, H., Viola, P.: Learning silhouette features for control of human motion. ACM Transactions on Graphics (2005)
29. Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 19–28. ACM (2005)
30. Shin, H.J., Lee, J., Shin, S.Y., Gleicher, M.: Computer puppetry: An importance-based approach. ACM Transactions on Graphics **20**(2), 67–94 (2001)
31. Shoemake, K.: Animating rotation with quaternion curves. In: SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, pp. 245–254. ACM Press (1985)
32. Simon, G., Fitzgibbon, A., Zisserman, A.: Markerless tracking using planar structures in the scene. In: Proceedings of International Symposium on Augmented Reality, pp. 120–128 (2000)
33. Vinayagamoorthy, V., Gillies, M., Steed, A., Tanguy, E., Pan, X., Loscos, C., Slater, M.: Building expression into virtual characters. Eurographics - State of the Art Reports pp. 21–61 (2006)
34. Wagner, D., Billinghurst, M., Schmalstieg, D.: How real should virtual characters be? In: Conference on Advances in Computer Entertainment Technology (2006)

Parag Chaudhuri is a post-doctoral researcher at MIRALab, University of Geneva. He received his PhD from the Indian Institute of Technology Delhi, India in 2006. He received the outstanding PhD award from IBM IRL for 2006. His primary research interests include all of computer animation. He is also interested in rendering, motion capture, real-time computer graphics, mixed reality and computer vision (geometric and active). He has been involved in the FP6 European projects ENACTIVE and INDIGO.

George Papagiannakis is a computer scientist and senior researcher at MIRALab, University of Geneva. He obtained his PhD in Computer Science at the University of Geneva in 2006, his M.Sc (Hons) in Advanced Computing at the University of Bristol and his B.Eng. (Hons) in Computer Systems Engineering at the University Of Manchester Institute Of Science and Technology (UMIST). His research interests are mostly confined in the areas of mixed reality, illumination models, real-time rendering, virtual cultural heritage and programmable graphics. He has actively been involved and significantly contributed to the CAHRISMA, LIFEPLUS, STAR, ENACTIVE, JUST and ERATO FP5 and FP6 European projects. Currently he is participating in the INTERMEDIA, INDIGO and EPOCH FP6 EU projects. He is a member of ACM and IEEE.

Nadia Magnenat-Thalmann has pioneered research into virtual humans over the last 25 years. She obtained her PhD in Quantum Physics from the University of Geneva. From 1977 to 1989, she was a Professor at the University of Montreal where she founded the research lab MIRALab. She was elected Woman of the Year in the Grand Montreal for her pionnering work on virtual humans and presented Virtual Marilyn at the Modern Art Museum of New York in 1988. Since 1989, she is Professor at the University of Geneva where she recreated the interdisciplinary MIRALab laboratory. With her 30 PhD students, she has authored and coauthored more than 300 research papers and books in the field of modeling virtual humans, interacting with them and living in augmented worlds. She is presently taking part in more than a dozen of European and National Swiss research projects and she is the Coordinator of several European Research Projects such as the Network of Excellence (NoE) INTERMEDIA, the European project HAPTEX and the European Research Marie Curie training network 3D ANATOMICAL HUMANS.