Natural user interfaces for virtual character full body and facial animation in immersive virtual worlds

Konstantinos C. Apostolakis, and Petros Daras

Information Technologies Institute, Centre for Research and Technology Hellas, Thessaloniki, Greece
{kapostol, daras}@iti.gr

Abstract. In recent years, networked virtual environments have steadily grown to become a frontier in social computing. Such virtual cyberspaces are usually accessed by multiple users through their 3D avatars. Recent scientific activity has resulted in the release of both hardware and software components that enable users at home to interact with their virtual persona through natural body and facial activity performance. Based on 3D computer graphics methods and vision-based motion tracking algorithms, these techniques aspire to reinforce the sense of autonomy and telepresence within the virtual world. In this paper we present two distinct frameworks for avatar animation through user natural motion input. We specifically target the full body avatar control case using a Kinect sensor via a simple, networked skeletal joint retargeting pipeline, as well as an intuitive user facial animation 3D reconstruction pipeline for rendering highly realistic user facial puppets. Furthermore, we present a common networked architecture to enable multiple remote clients to capture and render any number of 3D animated characters within a shared virtual environment.

Keywords: virtual character animation, markerless performance capture, face animation, Kinect-based interfaces

1 Introduction

In recent years, the introduction of the Microsoft Kinect ignited the avatar full body motion control paradigm based on the concept of avateering. *Avateering* or *puppetting* a virtual 3D avatar refers to the process of mapping a user's natural motoring activity and live performance to a virtual human's deforming control elements in order to faithfully reproduce the user's activity during rendering cycles. Already, a multitude of different schemes for full body avatar control exist in the scientific literature based on the skeleton tracking capabilities offered by software development kits and application

adfa, p. 1, 2011.

programming interfaces plugging into the Kinect sensor [2] [11, 12, 13]. Similarly, avatar facial animation through vision-based methods has been explored following a similar approach in which facial features on the user's face are tracked via a Kinect [15] or single image acquisition methods [3, 4] [10] to generate animation via detailed face rigs or pre-defined blendshapes. In this paper we present two distinct, real-time user avatar control interfaces specifically tailored for use in tele-immersive virtual worlds connecting remote users within a shared virtual environment. We present two similarly built frameworks for remote avatar full body and facial animation through the use of consumer-grade hardware such as the Microsoft Kinect sensor and standard HD webcams. Our Natural User Interface (NUI) frameworks have been developed for use in real-time, tele-immersive shared virtual environments and are designed to enable both user direct and responsive mapping of body movements to avatar characters in the virtual world as well as a means of reconstructing and animating user lookalike, highly realistic 3D virtual facial avatars.

The remainder of this paper is organized as follows: Section 2 presents a brief overview of the common networked architecture of each NUI framework. Sections 3 and 4 better elaborate on each framework individually, outlining the methods used to obtain avatar animation data from raw Kinect, and standard webcam output respectively. Section 5 then concludes with a discussion on the authors' final thoughts and future work.

2 Common framework architecture

Each framework described in this paper, is comprised of both a remote capturing and a rendering component. These are responsible for the acquisition of the proper user motion estimated data originating from the module's targeted hardware input, processing of the latter raw form into sensible avatar animation data and updating of the rendering pipelines in order to output the 3D character in a user motion-mimicking posture. Both frameworks' capturing component is deployed whenever a user is connecting to the shared virtual world via an *application layer multicast* (ALM) server/client network

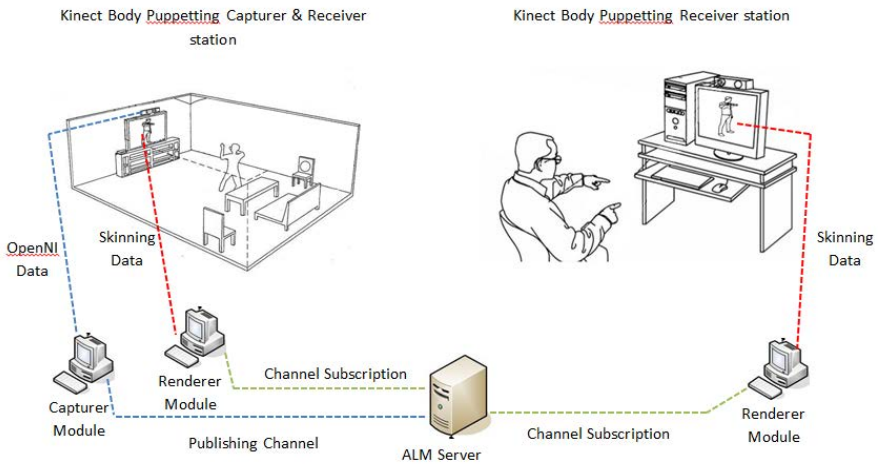

Fig. 1. Kinect full body avateering framework component architecture network.

architecture [16]. Each capturing component (one per user) is responsible for creating a single, new ALM channel and publishing data to it. The rendering component of each framework is similarly setup to serve a three-fold purpose: a) it generates a number of ALM channel subscribers which receive and reconstruct the user animation data published by any number of respective capturers; b) it loads the appropriate avatar assets; and c) it handles the real-time rendering and animation of all avatars in the system.

Through this scheme, multiple users can connect to a shared virtual environment and view any number of 3D animated avatars by simply subscribing to each users' capturing component publishing channel. In the case of the Kinect full body avateering framework, the ALM capturer channel is publishing user skeleton data, which the subscribing rendering components turn into avatar skinning data, as is demonstrated in the framework architecture diagram shown in Figure 1. Elaborate details on the depicted data flow are presented in the following Section. Similarly, the webcam facial animation framework features a tracking pipeline for turning raw camera input into user facial landmark animation data at the capturing site, which is posted on the capturer's publishing channel. The latter is in turn translated to 3D vertex

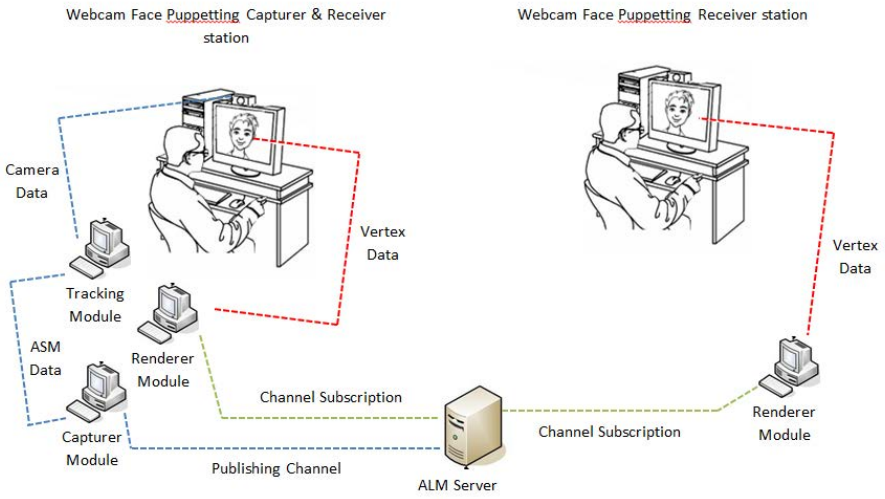

Fig. 2. Webcam avatar facial animation framework component architecture network.

buffer information at the receiving rendering components. The process is described in more detail in Section 4. A diagram of the architecture is depicted in Figure 2.

3 Full body avatar control using Kinect-based interface

Our Kinect full-body avateering framework enables users to transfer their physical body motion into a shared virtual environment through the use of markerless capturing via the skeleton tracking algorithms implemented for the Microsoft Kinect sensor. The process requires that the avatar 3D mesh is parented to an articulated structure of control elements called *bones*. Bones can be viewed as oriented 3D line segments that connect transformable *joints* (such as a knee or a shoulder). These joints usually offer a three-to-six Degrees-Of-Freedom deformation control of the avatar's mesh geometry, with respect to translation and rotation transformations. In order to provide a 1-by-1 mapping of Kinect trackable joints to avatar control elements, the

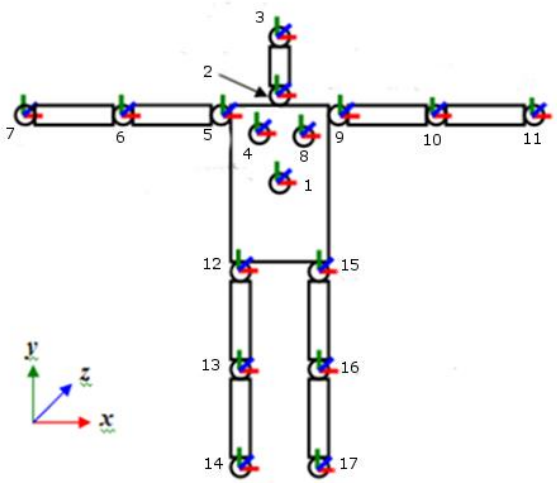

Fig. 3. 17-joint, OpenNI-based hierarchy of control structures defined for REVERIE Kinect Puppetted characters. Joints are identified as follows: 1) Torso; 2) Neck; 3) Head; 4) Left Collar; 5) Left Shoulder; 6) Left Elbow; 7) Left Wrist; 8) Right Collar; 9) Right Shoulder; 10) Right Elbow; 11) Right Wrist; 12) Left Hip; 13) Left Knee; 14) Left Foot; 15) Right Hip; 16) Right Knee; 17) Right Foot.

avatars are required to be rigged with a pre-defined 17-joints hierarchy defined by the OpenNI and NiTe joint tracking structure depicted in Figure 3[1].

As described in the previous Section, the framework consists of an end-to-end capturing and rendering module that generates character 3D animation based on skeleton data, received over the network from a Kinect capturing station. Both the remote capturing and rendering components of the module address skeleton joint data using a pre-defined path traversing all joints starting at the root of the hierarchy (the torso) and moving towards the end effectors (head, wrists and feet) in a left-to-right manner. This way, a 1-to-1 correspondence of Kinect user tracked joint data to avatar virtual joint information is ensured.

[1] PrimeSense, who was founding member of the OpenNI, shutdown the original OpenNI project on which our modules are linking to when it was acquired by Apple on November 24, 2013. The module retains its operability through the latest legacy version of the library (1.5.4.0 as of May 7, 2012).

For each animation frame, the remote *capturing* component of the full-body avateering framework generates joint position and rotation data with respect to the Kinect camera's world coordinate system. This data is represented by two sequences of floating point numbers, one referring to the XYZ-positions of the 17 joints while the other accumulates all 3x3 orientation matrices describing the rotation of each joint in camera world space. However, avatar mesh assets loaded and rendered by the remote *rendering* component of the framework, are usually stored with skinning information that constitutes a hierarchical bone structure requiring any affine transformations to be applied to each joint in its local axis space, in additional relation to its parent. Furthermore, avatar bones are defined with a static length, which should remain constant throughout the animating session regardless of varying user anthropometric measurements acquired by the skeleton tracking module, as multiple users of different ethnic backgrounds and physical attributes should be able to control avatars that share a common skeleton animation template basis. Therefore, a one-by-one direct copying of the tracked skeleton data to the avatar's joints would result in unrealistic rendered characters, as the remote capturing component would, in all, neglect any constraints applied to the animated 3D model skeleton during the characters' design in the 3D modelers' workshop. To achieve a plausible and constraint-bound 3D animation flow, the data obtained by the remote capturing component of the module is received and translated to joint-specific local-coordinate orientation quaternions in the remote rendering component before the joint matrices are updated and applied in the skinning calculations taking place in the hardware-accelerated avatar rendering pipeline. The process is described in detail in Algorithm 1 [2]. The root joint's quaternion can safely be obtained by the 3x3 global orientation matrix sent by the capturer.

After all local joint coordinate rotation quaternions are computed, avatar animation applies to standard skinning equations, with calculations being shared between the CPU and the GPU through dedicated skinning GLSL shaders loaded by the remote rendering module for the sake of real-time performance. The 3D mesh geometry is calculated per vertex by passing appropriate skinning matrices and per-vertex joint indices and weights along

Algorithm 1. *Kinect Avatar Animation Data*

Input: User's tracked joint positions P_i

Output: Avatar joint rotation quaternions q_i

1. (off-line step) For each avatar rig joint $J_i, i = 1, 2, \dots, 17$, the bone lengths $L_i^c, c \in CH\{i\}$ to its first-level child joints $CH\{i\}$ are calculated in the avatar rig hierarchy along with the corresponding unit-vector directions $d_i^c = \frac{(J_c - J_i)}{\|J_c - J_i\|}$. For notational simplicity, the *c* superscript referring to a child joint of J_i is dropped, wherever it's implied.
2. For each tracking frame, the user's tracked joint positions P_i with respect to the camera's world coordinate system are retrieved by the OpenNI and NiTe skeleton tracking libraries.
3. The direction of each user-tracked bone is determined by the normalized unit vector $s_i = \frac{P_c - P_i}{\|P_c - P_i\|}$, similar to the calculation made in step (1). The result is then multiplied by its corresponding length L_i, to ensure that all tracked joints are within the predefined avatar bone length distances from each other.
4. The axis of rotation m_i is determined by calculating the cross product $d_i \times s_i$, while the angle of rotation is calculated by $\theta_i = \cos^{-1}(d_i \cdot s_i)$, i.e. from the inner product of d_i and s_i.
5. The avatar's new joint position is set by traversing the joint hierarchy from the parent joint and adding the offset $L_i \cdot s_i$. The local rotation quaternion q_i^{lo}, is calculated from the axis m_i and the eigen-angle θ_i.
6. The final rotation quaternion q_i is calculated by multiplying the inverse rotation quaternion of the parent q_i^{pa} with q_i^{lo}, i.e. $q_i = (q_i^{pa})^{-1} \cdot q_i^{lo}$.

with the standard 3D vertex, normal and texture coordinates attributes to the vertex shader. The process requires the calculation of these skinning matrices by *posing* the virtual skeleton in the CPU according to the following scheme:

1. Setup the avatar skeleton. Traverse the avatar joint hierarchy from the root and calculate final 4x4 joint skinning matrix M_i by first calculating local rotation matrix M'_i from previous rotation quaternion q_i and multiplying with the joint's parent skinning matrix M_i^{pa}, i.e. $M_i = M_i^{pa} \cdot M'_i$. For the root joint, $M_i = M'_i$.
2. Flatten bone matrices to an array of floating point numbers for passing to the GPU via uniform bindings.

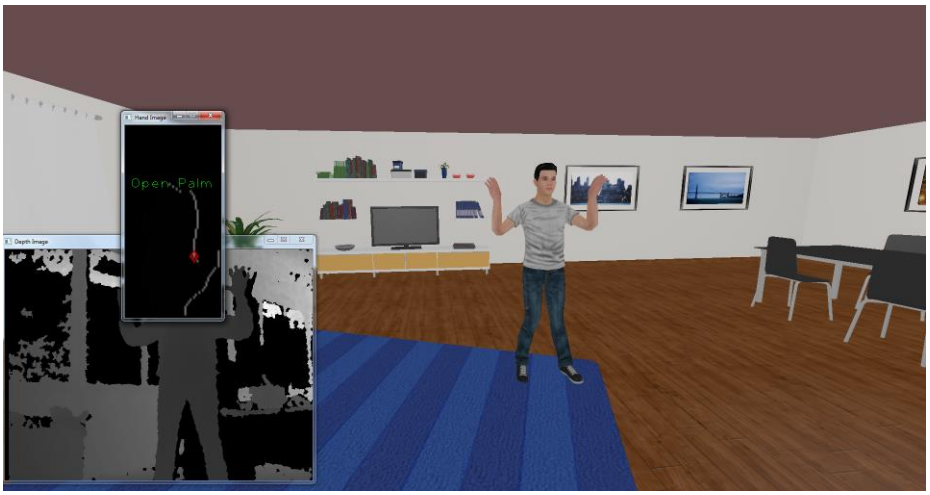

Fig. 4. Screenshot of an animated 3D avatar rendered by our Kinect full body avateering framework inside a virtual environment.

3. Normalize skinning weights using Manhattan distance metric and pass the appropriate buffers to the shader as vertex attributes.

Afterwards, the final vertex transformations are applied in the GPU by applying the following calculations:

1. Retrieve bone matrices M_x^v and M_y^v, corresponding to the bone matrices of joints x, y influencing vertex v according to the latter *skin indices* attribute.
2. Calculate skinned vertex position by applying the appropriate skinning weights w_x^v, w_y^v, retrieved from the vertex' attributes, to the transformation calculation:

$$v^{skin} = M_x^v \cdot v \cdot w_x^v + M_y^v \cdot v \cdot w_y^v \quad (1)$$

3. Apply model-view-projection transformations MVP to the skinned vertex:

$$v^{out} = MVP \cdot v^{skin} \quad (2)$$

An example of the module output is shown in Figure 4. All rendering components are based on plain OpenGL and GLSL shading languages, and are easily portable to both in-house as well as third-party 3D rendering engines providing integrated support for externally written shaders (e.g. Unity).

4 Avatar Facial Animation Control using a Webcam interface

Our webcam-based facial avatar animation framework enables users to display their natural facial activity and expressions through their 3D avatars in the virtual environment, using a marker-less facial landmark tracking scheme based on Active Shape Models (ASMs) [5], applied to frames obtained from a standard HD web camera. Unlike the full body animation framework described in the previous Section, which requires control elements (bones) to re-target animation of the user onto the character's bone structure, and unlike the scientific literature which uses either facial animation controls (rigs) [4] [10] or predefined blendshapes [3] [15] for animating the various 3D facial expressions, our framework works by directly applying changes to the mesh geometry through a one-by-one correspondence of the 3D mesh vertex data to the 2D tracked facial landmarks of the ASM shape fitted onto an instance of the user's face at each consecutive camera frame. In other words, the 3D shape geometry is reconstructed anew in each frame based on the 2D ASM geometry resulting from the fitting process. This 1-on-1 mapping is achieved by using the avatar's face mesh as a template for training an ASM on a large database of hand-annotated images. This way, for each tracked frame, the vertex/landmark index remains constant. The mesh geometry manipulation is achieved by projecting the 2D shape model to the 3D world, keeping the depth coordinate constant and aligned to the template's original size. As the 2D landmarks and camera frame also provide a means to obtain a live texture map and corresponding coordinates per frame, the avateering effect can be further enhanced in terms of realism by continuously updating the 3D model's texture information. This enhanced 3D animated mesh can otherwise be viewed as a time-constant, template-based 3D reconstruction of the user's face: each capturing frame generates a new set of 3D vertex position and texture coordinate attributes which result in a new 3D mesh object, which retains the original structure of the hand-modelled template. The live texturing features can also be dropped without further affecting the process of character animation in order to allow users to avateer characters who share little to no resemblance to their actual users. Furthermore, standard deformation techniques, like blendshape morph targets can be applied

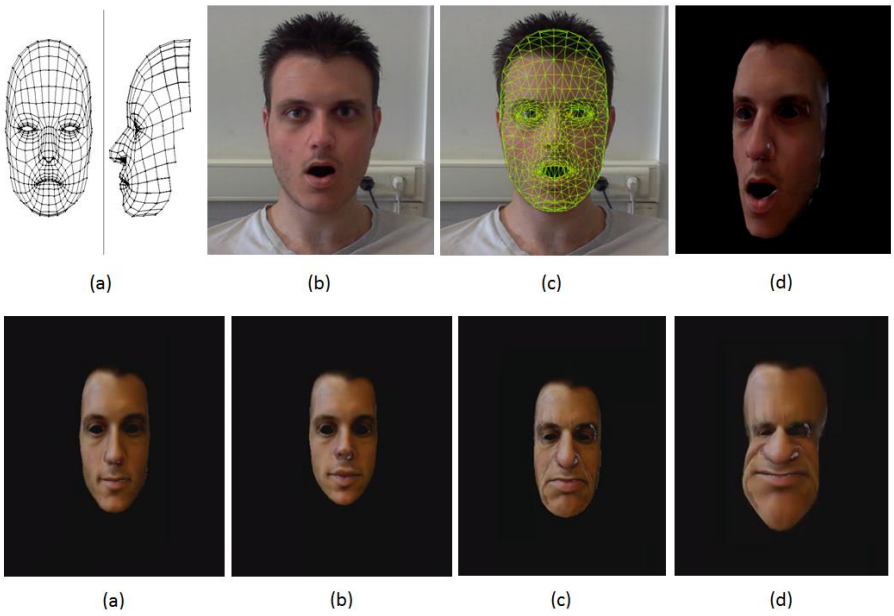

Fig. 5. Example of enhanced functionality possible through the webcam avatar facial animation framework, with both vertex and texture coordinate deform options active. Top row: (a) Template 3D face mesh; (b) webcam input frame; (c) corresponding real-time ASM fit; and (d) 3D reconstructed face with high amount of realism. Second row: original reconstructed face mesh (a) and three examples (b, c, d) of caricature avatars obtained by applying morphs in real time.

to generate a multitude of 3D geometries based on a single person's avatar face mesh. This allows our framework to easily create user 3D caricatures in real time. Examples of different animated avatars created from a single input camera frame are shown in Figure 5.

Obtaining a shape model of considerable detail (i.e. modern, real-time virtual character face meshes consist of over 500 vertices for enhanced visual detail) requires training on a large dataset of human face images. Such training is done by annotating (usually by hand) each image, keeping landmark annotation continuity consistent throughout the entire database. This process is

almost inconceivable for a human annotator, given the amount of detail required and the close proximity of a large number of landmarks due to the geometry structure of the 3D model meshes. In order to address this problem, an intuitive web-based visual annotation application was created using the *Reverie Avatar Authoring Tool* [1]. The annotation tool was developed specifically for generating ASM files targeted by this framework. The application was designed to accelerate the annotation procedure, by allowing annotators to superimpose an instance of the entire face mesh over the image and make appropriate adjustments to the face 3D pose, its anthropometric features (eye/nose/mouth size, position, rotation etc.) as well as the facial expression visual cues (such as individual Action Units [8] and FAPs [9]) by selecting between a large set of pre-defined blendshapes modelled for the generic template mesh in an external 3D modeling application[2]. Once the annotator has approximated the overlaid face image landmark geometry, an automatic process that projects the superimposed 3D model vertices to the 2D image plane generates an annotation file for the image. For rapidly acquiring twice the amount of data, an automatic, index-invariant flipping algorithm ensures a consistent annotation file is automatically generated for the x-plane flipped image, by keeping the model's vertex indices consistent (i.e., not flipping them) throughout the set. Using this intuitive tool, the annotation of a single image with over 500 landmarks is possible in less than 5 minutes. This efficiency measurement is of considerable note to the authors, seeing how traditional annotation schemes [7] are known take up at least as much time for 10 times less landmarks per model, while the procedure is done by placing each landmark individually on top of the desired image-plane coordinates, making the process delicate to human errors. The eventual ASM files are built using a third-party, pre-compiled C++ OpenCV library *asmlibrary* [14]. Since the landmarks recorded onto the annotation files are merely 2D projections of the 3D model's geometrical structure, a 1-to-1 correspondence of landmarks-to-3D vertices is guaranteed as an added bonus. Two screenshots of the annotation tool used for this module are shown in Figure 6, to better emphasize on the scheme's efficiency.

---

[2] Blender, free, open source 3D Computer Graphics Software http://www.blender.org/.

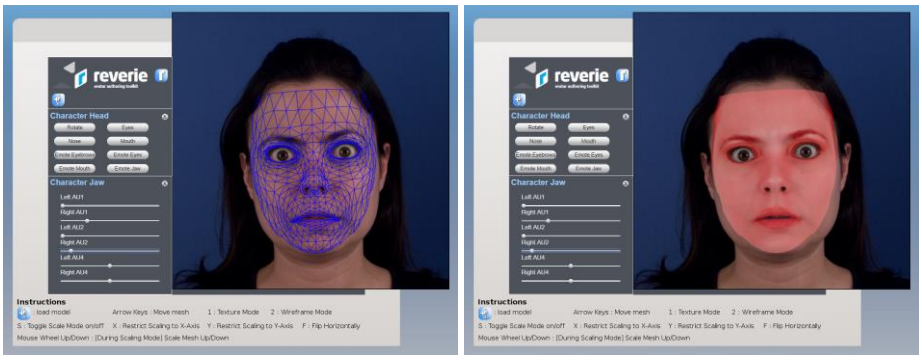

Fig. 6. Screenshots of the *RAAT*-powered web-based annotation tool developed and used for quickly annotating images with high resolution (>500) ASM landmarking schemes such as the one depicted in this image, featuring 511 vertices. The *left* image showcases the ASM model in wireframe mode which greatly helps placing groups of landmarks in their correct positions. The *right* image showcases the ASM model in generic textured view, which helps define facial features such as the eyebrows, lip lining, nostril outline etc.

After loading the appropriate ASM build model, each frame retrieved by the camera is processed to obtain a reconstructed 3D face model according to the Algorithm 2. After calculating the new 3D vertex coordinates, the model's vertex and texture coordinate buffers are appropriately updated at each drawing frame, along with the diffuse texture which is updated with the current camera frame. At GPU level, no additional calculation takes place to display the deformed vertices other than applying the avatar's model-view projection transformation MVP to get the final output:

$$v^{out} = MVP \cdot v \qquad (3)$$

An example of the rendered output of this framework is depicted in Figure 7.

5 Conclusions

In this paper, we presented two NUI frameworks for remote avateering of 3D characters in shared virtual environments using ALM network architecture. We elaborated on our data flow pipelines and demonstrated how raw sensory input can be translated into sensible animation data for controlling both avatar full body movements, as well as a capturing and reconstruction pipeline of the user's current facial image into a highly realistic representation of

Algorithm 2. *3D vertex reconstruction from ASM tracking data*

Input: set of 2D landmarks χ_i corresponding to the ASM vertices.
Output: Avatar 3D vertex data v_i and (optionally) texture coordinates data c_i

1. (Optional) The current landmarks $\chi_i(x_i^{\chi}, y_i^{\chi})$ coordinates are normalized to retrieve texture coordinates $c_i = (\frac{x_i^{\chi}}{width}, 1.0 - \frac{y_i^{\chi}}{height})$, where $width$ and $height$ correspond to the camera frame XY-resolution.

2. The current shape model is aligned to an orthogonal posture of the same model to nullify the effects of any roll-pitch-yaw rotation of the user's head. This is achieved by minimizing the following weighted sum of squares:

$$E = \sum_{i=1}^{n} \left(\chi'_i - T_t(\chi_i)\right)^T W_i(\chi'_i - T_t(\chi_i))$$

   where χ_i, χ'_i are the two sets of n points corresponding to the original and aligned shape respectively, and $T_t(\chi)$ is a transformation with parameters t [6].

3. The 2D landmark coordinates for each point $\chi'_i(x'^{\chi}_i, y'^{\chi}_i)$ in the aligned shape are projected to a pseudo-3D space centered at (0, 0, 0) (with the z-coordinate being kept constant all through the algorithm and retrieved by the 3D mesh model file loaded by the module) through the following equation:

$$v_i'(x, y, z) = (x_i', y_i', z_i) = (\left(\frac{x'^{\chi}_i}{width}\right) - 0.5, 0.5 - \left(\frac{y'^{\chi}_i}{height}\right), z_i)$$

   where z_i is the z-coordinate of the 3D model vertex at the same index i.

4. The XY coordinates of each vertex obtained in step 2 are multiplied by an appropriate scaling factor for each dimension dim calculated using the ASM model scale s_{dim}^{asm} in comparison to the 3D mesh model scale s_{dim}^{model} loaded onto the module:

$$v_i''(x, y, z) = (x_i'', y_i'', z_i) = (x_i' \cdot \frac{s_x^{model}}{s_x^{asm}}, y_i' \cdot \frac{s_y^{model}}{s_y^{asm}}, z_i)$$

5. Finally the coordinates are translated to an offline designated anchor point $v^{anchor}(x^{anchor}, y^{anchor}, z^{anchor})$ (in our implementation, we use the 3D model's chin vertex as the anchor point) to obtain the final coordinates:

$$v_i(x, y, z) = (x_i'' + x^{anchor}, y_i'' + y^{anchor}, z_i)$$

the user in a shared 3D space to enhance the element of telepresence. Towards this latter approach we demonstrated how a simple ASM face fitting process can be expanded to train and build high resolution (>500 landmarks)

Fig. 7. Front and side view of real-time 3D facial animation avatar anchored at chin with body mesh. Post processing rendering effects are applied to ensure a smooth color transition between the face mesh and the body mesh diffuse textures.

shape models which in turn correspond to the 3D vertices of the avatar face model. This enables the acquisition of both 3D vertex as well as 2D texture coordinates data per tracked landmark, to generate a highly realistic virtual 3D representation of the user using a single image, standard HD camera capturing framework. By supplementing additional 3D mesh deformation and texturing techniques, we demonstrated how our framework further enhancement of the rendered output by generating identifiable virtual 3D user caricatures. We expect our results to be particularly useful for future tele-immersive systems, video games, low-cost actor performance capturing, virtual mirror applications and more.

Acknowledgement

The research leading to this work has received funding from the European Community's Horizon 2020 Framework Programme under grant agreement no. 644204 (ProsocialLearn project).

References

1. Apostolakis, K. C., and P. Daras. "RAAT-The reverie avatar authoring tool."*Digital Signal Processing (DSP), 2013 18th International Conference on*. IEEE, 2013.

2. Apostolakis, Konstantinos C., et al. "Blending real with virtual in 3DLife." Image Analysis for Multimedia Interactive Services (WIAMIS), 2013 14th International Workshop on. IEEE, 2013.

3. Cao, Chen, et al. "3D shape regression for real-time facial animation." *ACM Trans. Graph.* 32.4 (2013): 41.

4. Cho, Taehoon, et al. "Emotional Avatars: Appearance Augmentation and Animation based on Facial Expression Analysis." *Appl. Math* 9.2L (2015): 461-469.

5. Cootes, Timothy F., et al. "Active shape models-their training and application."*Computer vision and image understanding* 61.1 (1995): 38-59.

6. Cootes, Timothy F., and Cristopher J. Taylor. "Statistical models of appearance for computer vision." (2004).

7. Cootes, Timothy F. "Modeling and Search software". Available at http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/software/am_tools_doc/index.html

8. Ekman, Paul, and Wallace V. Friesen. *Manual for the facial action coding system*. Consulting Psychologists Press, 1978.

9. Ostermann, Jorn. "Face animation in mpeg-4." MPEG-4 Facial Animation: The Standard, Implementation and Applications (2002): 17-55.

10. Rhee, Taehyun, et al. "Real-time facial animation from live video tracking."*Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2011.

11. Sanna, Andrea, et al. "A kinect-based interface to animate virtual characters."*Journal on Multimodal User Interfaces* 7.4 (2013): 269-279.

12. Shapiro, Ari, et al. "Automatic acquisition and animation of virtual avatars." *VR*. 2014.

13. Spanlang, Bernhard, et al. "Real time whole body motion mapping for avatars and robots." *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*. ACM, 2013.

14. Wei, Yao. "Research on facial expression recognition and synthesis." *Master Thesis, Department of Computer Science and Technology, Nanjing* (2009).

15. Weise, Thibaut, et al. "Realtime performance-based facial animation." *ACM Transactions on Graphics (TOG)*. Vol. 30. No. 4. ACM, 2011.

16. Zahariadis, Theodore, et al. "Utilizing Social Interaction Information for Efficient 3D Immersive Overlay Communications." *Novel 3D Media Technologies*. Springer New York, 2015. 225-240.