RAAT – The Reverie Avatar Authoring Tool

A JavaScript library for designing online 3D character creator software

K. C. Apostolakis and P. Daras
Information Technologies Institute
Centre for Research and Technology Hellas
Thessaloniki, Greece
kapostol@iti.gr, daras@iti.gr

***Abstract*—avatar embodiment within the World Wide Web has gained a lot of popularity in recent years thanks to the introduction of networked virtual environments created for socialization and entertainment purposes. As each of these virtual worlds generates a unique set of user requirements concerning representation preferences based on the environment's context, it becomes clear that every attempt at creating such virtual worlds should encourage the development of the appropriate avatar authoring tools, being based on a thorough study of avatar desirable features. The Reverie Avatar Authoring Tool (RAAT) introduced in this paper helps developers address these ever-emerging avatar feature requirements, allowing them to easily set up and design online character creation applications, tailored to the virtual environment specifications. Summarizing the design process to a simple task of documenting the application interface within a single script, RAAT encapsulates the demanding tasks of character creation within simple function calls, while also offering a web-based real-time solution for photorealistic integration of user physical appearance onto the character mesh.**

Keywords—3D Avatar; Avatar customization; Virtual world; Browser software

I. INTRODUCTION

Since the introduction of the World Wide Web, a lot has changed, both within the web itself, as well as how the everyday average user connects and interacts with its content. As hardware cost and size decreases in a steady rate that allows users to access the web through innovative web browsing hardware devices, such as smart phones and tablets, so do web technologies need to mature and take advantage of hardware accelerations and multimedia support, while definitively departing from the more traditional software package installation requirements. Recent developments in web technologies, such as HTML5, WebGL, CSS3, Microsoft Silverlight, and many others, have led web developers to enrich websites with stunning interactive 3D graphics, ranging from simple embellishments to full-blown commercial applications featuring interactive virtual environments, examples of which include 3D browser games, virtual tours to remote real-life locations, and networked virtual worlds setting the stage for social interaction between remote users. In the latter sense, users cease to merely exist as static profiles, but rather embody a customizable representation of one's self, a virtual entity otherwise known as an *avatar*.

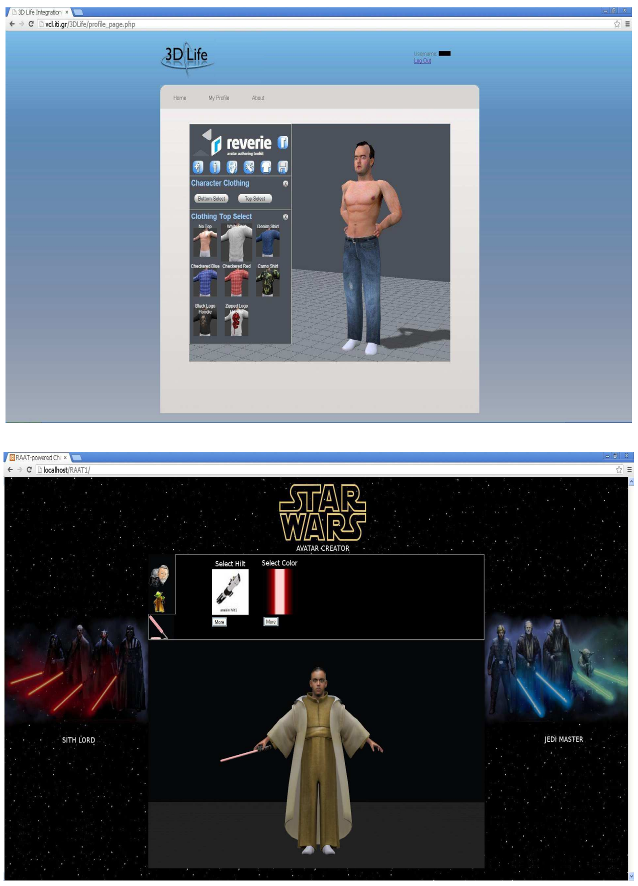

Figure 1. Examples of character creator web based applications created with the Reverie Avatar Authoring Tool. Both pages are displayed using Google Chrome web browser.

Avatars, representation of which range from simple 2D graphical images to fully rendered and textured 3D models, have been deployed to populate networked virtual environments for quite some time (such as the highly popular social 3D virtual environments Second Life[1] and IMVU[2]), and gradually, major players in the video gaming and network solutions industry have come to adopt avatars as a means for their users to personify themselves, examples including Nintendo's Mii's, Microsoft's XBOX Live Avatars and Yahoo's Avatars. Furthermore, users have shown to respond

This work was supported by the European Commission under contract FP7-287723 REVERIE (http://www.reveriefp7.eu).

[1] http://www.secondlife.com
[2] http://www.imvu.com

overwhelmingly well to the idea of creating, owning and caring for their very own 3D virtual identity [1] [2], and will even consider the actual purchase of virtual ornaments (such as clothing, props, hairstyles, and in some cases, virtual pets) in order to ensure their avatar is unique, easily identifiable and special [3].

As networked cyberspaces inhabited by virtual characters steadily grow to become a frontier in social computing, it becomes apparent that certain tools will need to be made available for both developers and users to encourage the continued interest in creating and inhabiting such shared virtual environments. Designing the tools for avatar creation however, requires prior thought on several factors that might influence how users will interact with the virtual environment and its inhabitants, as well as how they will react towards, and bond with their own incarnation within this virtual world. Different needs of representation will arise within a virtual environment geared towards an adult audience with a focus on socialization, in contrast to massively multiplayer on line games where character features and visual traits are sometimes intentionally exaggerated, unrealistic and blown out of proportion. Taking all of the above under consideration, it's safe to say that no one current solution exists that can cover all of the possible user requirements in the process of avatar creation. Each new networked virtual environment will generate its own content-based context, and target a different group of users with varying demands and expectations concerning the appeal of their virtual personalities. As a result, with each new virtual social environment, a new daunting task of designing and developing avatar authoring tools will emerge, adding to the overall complexity of the development plan.

In order to address this issue, and offer a solution that greatly simplifies the authoring tool design process, in this paper we introduce the Reverie Avatar Authoring Toolkit (RAAT), an open source JavaScript software library that allows developers to easily design online interactive character creation software, tailored to the exact needs of the hosting networked virtual world. The library includes pre-defined user interface controllers for loading and manipulating multi-part character geometry directly on the client-side, and comes with a set of server-side tools that allow users to facilitate their own visual appearance onto the character using Active Shape Models [14]. As the design process is reduced to a simple task of modifying a single template file script, RAAT-powered avatar creator software serve as a prime example of how the latest Web technologies can support the creation of an attractive, competitive software package that draws its strength from simplicity, adaptiveness, portability and zero user-end prerequisites concerning both physical memory space as well as installation times.

The remainder of this paper is organized as follows. In Section 2, an overview of the RAAT library is presented along with a brief summary of recent related work, while Section 3 documents the RAAT library software components for a comprehensive overview of the library-powered software structure. Section 4 finally concludes this paper along with some insight on future work.

II. Background And Related Work

In general, studies have suggested that the process of avatar creation could lead to users forming a stronger sense of self-presence and psychological closeness to the created character [4]. Much of the related literature on avatar creation methodologies focuses on research on the fields of automatic mesh deformation and modeling [7] [8] as well as user reconstruction [9] [10], in an attempt to create faithful representations of the user's physical appearance and integrate those onto the virtual character geometry. The idea behind such attempts is that users are more likely to relate to a virtual character that closely resembles their own appearance, enabling a higher sense of presence. Moreover, automatic methods for the retrieval of the user's shape and appearance greatly enhance usability of such systems. As the human face is one of the most easily recognizable and expressive features of the human visage, many researchers have studied ways to transfer the facial likeness of people onto the avatar's face using photographs [8] [11] [12] [13]. Many of the reported results have been adopted onto commercial and open-source applications, with various degrees of customization options and tools to accomplish the desired effect.

However, studies show that not all virtual environments are driven by the same needs for avatar representation, meaning, some features are more desirable than others in specific situations. The way the avatars look in terms of realism, multitude of customization options and more importantly, their relation to the context (i.e. how the avatar's look and feel matches that of the virtual environment it's intended to populate) all have an impact on how well users receive the experience of being able to customize their own virtual identity [5]. However translating user needs in terms of context is a complex, yet viable task, in order to ensure users will preserve an interest in investing both time and caring to their virtual representation, and as a result the virtual environment in general. Design choices, such as being able to incorporate realistic visual features onto the appearance of the character, or rather select features from a wide range of pre-made assets have to rely on how much effort users feel they should invest in creating the character, as well as their intention to share personal information, such as their physical appearance. For example, realistic representations of oneself are more preferable in environments, where virtual presence is closely related to social interaction and communication, as is the case with teleconferencing, while online game users on the other hand have been shown to appreciate a more generic, less realistic look that leaves more room for imagination [5] [6].

RAAT is a JavaScript library that allows developers to quickly design and set up every viable component of a complete avatar authoring engine. Such components include the graphical user interface (GUI) elements which serve as the main interaction dialog between the user and the authoring engine, as well as 3D scenes, objects and renderers serving the purpose of real-time visualization. As the entire math associated with rendering, projection and lighting calculations is handled by the underlying JavaScript 3D Library Three.js[1], all of RAAT's functionality and generation of WebGL primitives is encapsulated within comprehensible representations of parameterized object classes, such as 3D

[1] Three.js library available at http://mrdoob.github.com/three.js/

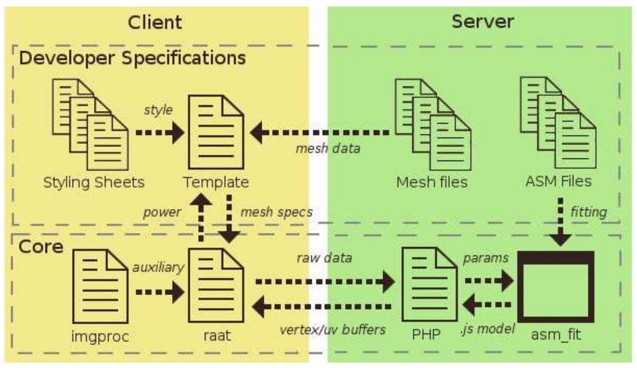

Figure 2. Structure of a typical RAAT-powered character creator software.

viewers, texture composers, GUI windows and multi-part character meshes. Being powered by Three.js, which features support of sophisticated rendering techniques, such as hardware-accelerated graphics and post-processing rendering effects, RAAT encourages developers to focus on more creative aspects, such as manipulating the GUI layout, provide or limit functionality, where deemed necessary, and ultimately modeling the 3D base mesh to be parameterized using the end-user application. As Three.js provides a wide range of mesh format file loaders in order to introduce externally created content into the virtual scene that serves as a canvas for RAAT to display and deform character meshes on, in a nutshell, RAAT can be viewed as a powerful external add-on to the Three.js library, allowing developers to easily and quickly set up fully functional impressive character creation software, tailored to the exact needs and visual style of the web application intended to use the created content.

In addition to being a powerful asset for designing avatar creation applications, RAAT benefits from the added value of being completely based on current web technology, meaning that the resulting software can be directly hosted on an ordinary website, requiring no prior download or installation procedure to be performed by the users before being able to create their characters. RAAT's object-oriented design allows it to be easily integrated into virtually any HTML5 webpage, while offering web developers the basic tools with which a character mesh can be loaded, customized and exported, encapsulating the entire graphical user interface into a single HTML division section (div), that can be easily placed anywhere within the page body, and styled to preference using standard CSS styling sheets.

III. Library Components And Software Structure

The RAAT software library consists of a number of components, whose principal role is to encapsulate most of the internal functionality inside a shell consisting of simple function calls, as has been described in the previous Section. Software powered by the RAAT library consists of a number of files, some of which need to be specified by the developer designing the end-user application. The core library itself is comprised of the client-side JavaScript code contained within the minified core scripts containing all of the abstract classes for the GUI elements, as well as a number of auxiliary image processing algorithm implementations for texture manipulation at pixel level. The minified scripts are intended to be included within the HTML script along with a Template JavaScript file that has been appropriately modified by the application developer, and documents the overall structure and functionality of all GUI elements while setting up identifiers for styling with complementary external CSS scripts.

On the server-side, RAAT can communicate via PHP with a standalone ASM fitting application. The functionality is provided for avatar authoring software where user representation via photorealistic data is determined to be a welcome feature. If desirable, the application will automatically process and generate mesh geometry and texture data obtained from photographs, in a similar fashion to [12]. A graphical representation of a typical RAAT-powered character creator software architectural structure can be seen in Figure 2.

A. *Core Functionality*

RAAT core functionality at the client-side is organized within two minified JavaScript files, dubbed *imgproc* (Image Processing) and *raat*. The first provides a number of auxiliary functions to the latter, intended for texture manipulation at pixel level, most notably whenever photographic data is used to incorporate user appearance to the character mesh. This process requires textures to be created and seamlessly blended; while a recoloring operation might be applied to ensure overall character skin tone matches the photographed skin color.

As described in the previous Section, RAAT is comprised of a number of methods that encapsulate the majority of the library's functions into a set of comprehensive component classes and function calls. The base set of these classes includes:

- 3D Viewer – The 3D Viewer class contains everything related to the real-time 3D rendering environment for visualization of the character. A call to the 3D Viewer's constructor will setup the scene, lighting, camera and WebGL renderer and create controls for navigating the environment with the mouse.

- Character Canvas – Since a virtual character may be a compilation of different meshes (for example, body, hair, clothing, etc.), the Character Canvas class contains information about the mesh structures that comprise the final character, and offers a simple interface with which to cause mesh geometry to deform using pre-defined morph targets (such as increasing a character's body mass to generate a more chubby character). Developers are responsible to provide their own mesh files and corresponding morph target data, according to the virtual environment context.

- Window – The Window class creates an HTML division element that can be styled to preference using CSS and can contain any number of stylizable interface elements for users to interact with the creator, such as interactive buttons, draggable sliders and file uploaders.

```
setup: function (parentID, x, y, width, height) {
   // create 3D viewer
   this.viewer = new RAAT.Viewer3D(“areaWebGL”, width, height);

   // main buttons window
   var elements = [{
      interfaceType: “interactiveButton”, id: “b1”, html: “New Character”,
      javascript: function () {settings1.div.style.display = “block”;
      }}, {
      interfaceType: “interactiveButton”, id: “b2”, html: “Clothing”,
      javascript: function () {settings2.div.style.display = “block”;
      }}, {
      interfaceType: “interactiveButton”, id: “b3”, html: “Hairstyle”,
      javascript: function () {settings3.div.style.display = “block”;
      }}];
      this.toolkit = new RAAT.Window(this.viewer, “panel”, null, 84, 222,
true, elements);

   // -1: new character dialog window
   var elements1 = [{
      interfaceType: “exitButton”, id: “ex”, html: “X”,
      javascript: function () {settings1.div.style.display = “none”;
      }}, {
      interfaceType: “interactiveButton”, id: “b4”, html: “Male”,
      javascript: function () {Template.viewer.setupMesh(“male.js”, 0);
      }}, {
      interfaceType: “interactiveButton”, id: “b4”, html: “Female”,
      javascript: function () {Template.viewer.setupMesh(“female.js”, 0);
      }}];
      var settings1 = new RAAT.Window(this.toolkit.div, “dialog”, “New
Character”, 148, 148, false, elements1);

   // -2: Clothing dialog window
   var elements2 = [{
      interfaceType: “exitButton”, id: “ex”, html: “X”,
      javascript: function () {settings2.div.style.display = “none”;
      }}, {
      interfaceType: “interactiveButton”, id: “b6”, html: “No Clothes”,
      javascript: function () {Template.viewer.deallocateMesh(1);
      }}, {
      interfaceType: “interactiveButton”, id: “b7”, html: “Suit”,
      javascript: function () {Template.viewer.setupMesh(“suit.js”, 1);
      }}];
      var settings2 = new RAAT.Window(this.toolkit.div, “dialog”, “Clothing”,
148, 148, false, elements2);

   // -3: Hairstyle dialog window
   var elements3 = [{
      interfaceType: “exitButton”, id: “ex”, html: “X”,
      javascript: function () {settings3.div.style.display = “none”;
      }}, {
      interfaceType: “interactiveButton”, id: “b8”, html: “NoHair”,
      javascript: function () {Template.viewer.deallocateMesh(2);
      }}, {
      interfaceType: “interactiveButton”, id: “b9”, html: “Hairstyle 1”,
      javascript: function () {Template.viewer.setupMesh(“hairstyle_1.js”, 2);
      }}];
      var settings3 = new RAAT.Window(this.toolkit.div, “dialog”, “Hairstyle”,
148, 148, false, elements3);
},
```

Figure 3. Code example of the RAAT Template file. This particular example sets up a 3D Viewer and a buttons panel, each button responsible for calling a Window class with extended functionality. Here, the first Window offers two buttons to select gender of the character, while the other two are similar in structure, offering options for Clothing and Hairstyle meshes to be added to the Character Canvas.

The above components are strategically placed within the HTML document using a template script that summarizes the overall software design into a set of Window class function calls. An example of a simple avatar creator application design via the template file is presented in Figure 3. It is the developer duty to ensure the consistency of all interface elements defined in the template and their respective functionality. This type of structure allows developers to quickly and effortlessly create, test and make changes to the character creator software, by simply altering the code written in the template file.

B. *Server-side processing*

In cases where user capturing and appearance integration onto the character mesh is desirable, RAAT core JavaScript code contains two more abstract classes that can issue requests to the host server via PHP, in order to send and obtain data

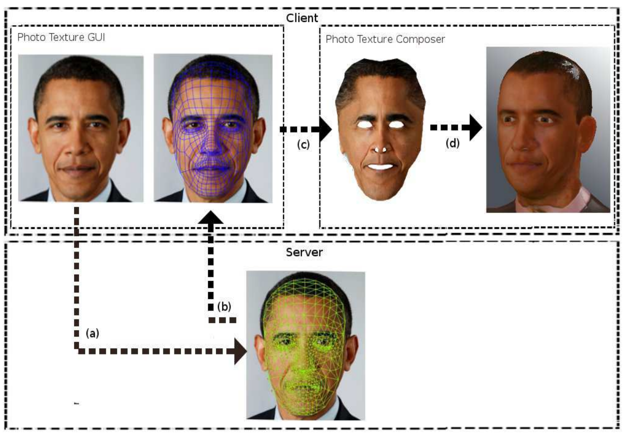

Figure 4. Overview of the ASM fitting and geometry/texture generation process. A frontal face image is provided by the user to which a best fit is determined (a). The 2D model vertices are used to generate a visible geometry structure and at the same time serve as UV coordinates for the texture-baking mesh (b). Vertices of the model are then manipulated and a texture is baked (c) before being applied onto the final character mesh (d).

from the ASM fitting application *asm_fit*. This application was written in C++ using OpenCV 2.3.1[1] and asmlibrary 6.0[2], and uses Viola-Jones classifier cascade [15] XML files and pre-built ASM data files, trained to automatically fit mesh vertex data to detected instances of desired object within a photograph or image. Web developers may choose to enable this functionality, by initializing instances of the following two classes within the template file:

- Photo Texture Graphical User Interface – This GUI class is structurally similar to the 3D Viewer, in that it creates a 2D rendering environment for viewing fitting geometry results on top of the user-specified image. The class contains functions that enable web camera streaming and snapshot capturing, and is responsible for sending the raw image data, issuing the call to the *asm_fit* application via PHP using the correct set of input parameters (concerning which Haar cascade file and which ASM data file to use). The server-side application responds by sending back the fitting results in JavaScript format to generate mesh geometry (vertex buffer) in real-time. This mesh is subsequently sent to the Photo Texture component for generating the final texture, as is described on the bullet below.

- Photo Texture Composer – This class is responsible for rendering (“baking”) the end result textures within a second, hidden 2D rendering environment, using the mesh obtained from the Photo Texture GUI component. Baking of the texture is achieved by projecting the mesh’ original UV coordinates to the 2D scene as vertex data and in a vice-versa manner project its vertex buffer as UV coordinates. This procedure ensures that the original format of the texture is not altered, to keep texture files consistent and usable outside the scopes of RAAT (for example, adding in-game texture layers to user-created characters, such as blood splatters in action game networked environments).

[1] OpenCV library available at http://opencv.willowgarage.com/wiki/
[2] asmlibrary 6.0 available at http://code.google.com/p/asmlibrary/

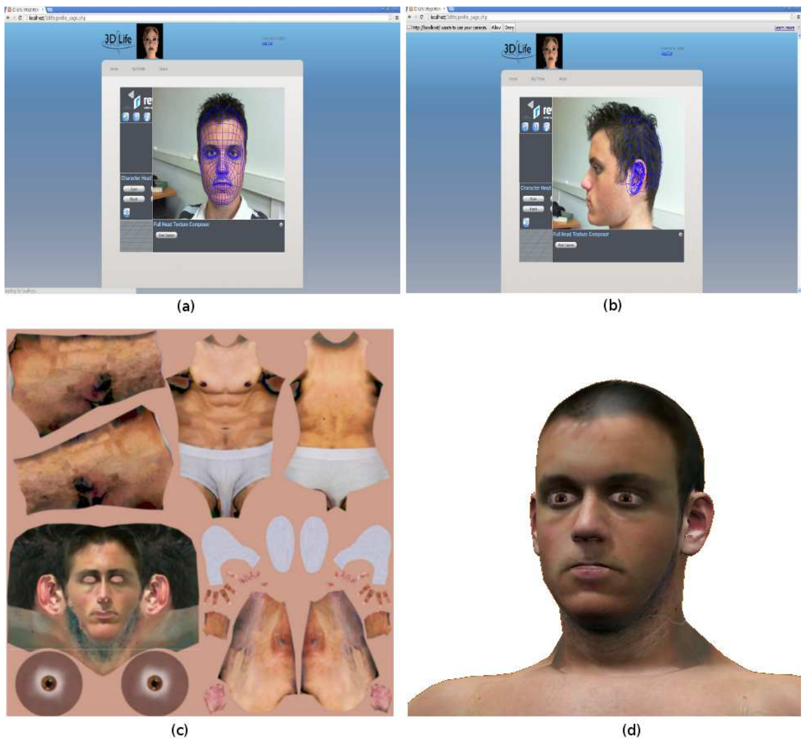

Figure 5. Photographic Texture Composer Example

For example, a frontal face detector can be used to identify the presence of a user's face when webcam video streaming is enabled, and a best fit is determined to generate the mesh geometry that appropriately represents the user's facial shape, and simultaneously retrieve UV coordinates to apply the photorealistic texture. The pipeline of the described process can be seen in Figure 4. In a similar way, any combination of Haar cascade files and pre-trained ASM data files can be used to generate mesh and texture data for different body parts, such as the head profiles, the eyes, the ears, hands, upper body, etc. An example is presented in Figure 5, where users capture a frontal (a) and profile (b) image of their heads to generate a full character head texture and recolor the body skin tone accordingly (c). The end result is then applied onto the character mesh (d). For a complete and comprehensive overview of training Haar cascade classifiers, as well as generating appropriate ASM data files, readers are referred to the respective OpenCV and asmlibrary documentation files.

IV. Conclusion

In this paper, the Reverie Avatar Authoring Tool, a library intended to help web developers design online character creation software was described. RAAT is a powerful tool for quickly generating 3D character authoring software, and offers developers the tools necessary to set up crucial software components, such as real-time character renderers, graphical user interface tools and photorealistic texture composers. Thanks to the library's simple structure, developers are only eligible to create and add character mesh files according to the virtual environment's context, and are simply required to set up their application's structure using a simple template script that contains the entire software functionality. The resulting character creator application can be easily uploaded and hosted via HTTP servers, styled to preference using external CSS files, and easily modified at will, by making changes to the aforementioned template script. Lifting the burden of handling the intrinsic math behind the procedures implemented within RAAT, creativity and imagination are all that developers need in order to create stunningly looking online character creation applications in no time.

With the library's ease of use and majority of pre-built options, we expect RAAT to be welcomed as a prime solution to the challenge of creating online avatar authoring applications as an added asset of setting up a new networked virtual environment. Already, RAAT is intended to power the character creation tools for the FP7 3DLife and REVERIE projects. Future work will focus on adding features such as real time puppeteering of the created avatars via virtual mirroring, as well as full 3D reconstruction of users by utilizing support of the Microsoft Kinect sensor.

References

[1] D. I. Cordova, M. R. Lepper, "Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization and choice". Journal of Educational Psychology, 715-730, 1996.

[2] S. Lim, "The effect of avatar choice and visual POV on game play experiences". Unpublished Disertation, Stanford University, California, 2006.

[3] D. Chung, "Something for nothing: understanding purchasing behaviors in social virtual environments". CyberPsychology & Behaviour 6, 538-554, 2005.

[4] D. Chung, B. deBuys, C. Nam, "Influence of avatar creation on attitude, empathy, presence and para-social interaction". Human-Computer Interaction, Interaction Design and Usability, 711-720, 2007.

[5] M. Boberg, P. Piippo, E. Ollila, "Designing avatars". In proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts, pp. 232-239, ACM, 2008.

[6] N. Ducheneaut, M. H. Wen, N. Yee, G. Wadley, "Body and mind: a study of avatar personalization in three virtual worlds". In proceedings of the 27th international conference of Human factors in computing systems, pp. 1151-1160, ACM, 2009.

[7] M. Bastioni, M Flerackers, "MakeHuman: Open source tool for making 3d characters". 2007.

[8] J. Lee, Y. S. Choi, B. K. Koo, C. J. Hwang, "An intuitive system for 3D avatar with high-quality". In Consumer Electronics (ICCE), 2010 Digiset of Technical Papers International Conference on, IEEE, pp. 517-518, 2010.

[9] A. Hogue, S. Gill, M. Jenkin, "Automated avatar creation for 3D games". In proceedings of the 2007 conference on Future Play, pp. 174-180, ACM, 2007.

[10] D. Knoblauch, P. M. Font, F. Kuester, "VirtualizeMe: Real-time avatar creation for tele-immersion environments". In Virtual Reality Conference (VR), 2010 IEEE, pp. 279-280, IEEE, 2010.

[11] T. Sucontphunt, Z. Deng, U. Neumann, "Crafting personalized facial avatars using ediatable portrait and photograph example". In Virtual Reality Conference VR 2009, IEEE, pp. 259-260, 2009.

[12] Z. Mingming, L. Shoukuai, W. Jiajun, S. Huaqing, P. Zhigeng, "The 3D caricature facemodeling based onaesthetic formulae". In proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry, ACM, pp. 191-198, 2010.

[13] M. Zollhöfer, M Martinek, G. Greiner, M. Stamminger, J. Süßmuth, "Automatic reconstruction of personalized avatars from 3D face scans". Computer Animation and Virtual Worlds, 22(2-3), pp. 195-202, 2011.

[14] T. F. Cootes, C. J. Taylor, D. H. Cooper, J. Graham, "Active shape models-their training and application". Computer vision and image understanding, 61(1), pp. 38-59, 1995.

[15] P. Viola, M. Jones, "Rapid object detection using a boosted cascade of simple features". In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, pp. I-511, IEEE, 2001.