VhCVE: A Collaborative Virtual Environment Including Facial Animation and Computer Vision

Henry Braun, Rafael Hocevar, Rossana B. Queiroz, Marcelo Cohen, Juliano Lucas Moreira, Julio C. Jacques Júnior, Adriana Braun, Soraia R. Musse, and Ramin Samadani*

Graduate Programme in Computer Science
PUCRS - Av. Ipiranga, 6681, Porto Alegre, RS, Brazil

*Hewlett Packard Laboratories
1501 Page Mill Rd., Palo Alto, CA, USA

Figure 1: *Snapshpts from VhCVE platform.*

Abstract

In this paper we present a platform called VhCVE, in which relevant issues related to Collaborative Virtual Environments applications are integrated. The main goal is to provide a framework where participants can interact with others by voice and chat. Also, manipulation tools such as a mouse using Computer Vision and Physics are included, as well as rendering techniques (e.g. light sources, shadows and weather effects). In addition, avatar animation in terms of face and body motion is provided. Results indicate that our platform can be used as a interactive virtual world to help communication among people.

Keywords:: CVE, Facial Animation, Avatars, Computer Vision

Author's Contact:

{henry.braun, rafael.hocevar}@cpph.pucrs.br
{rossana.queiroz, marcelo.cohen,
julio.silveira,soraia.musse,adriana.braun}@pucrs.br
ramin.samadani@hp.com

1 Introduction

Many years ago, at the beginning of Virtual Reality (VR) area, people expected to have a new interface where they could interact with others, in a parallel world. The application which probably summarizes this expectation is the CVE (Collaborative Virtual Environment). CVEs are three-dimensional computer-generated environments where users are represented by avatars and can navigate and interact in real-time, independently of their physical location [Frécon 2004].

In CVEs, VR and Computer Graphics (CG) technologies are used to immerse multiple individuals in a single shared space. Such environments support a range of activities, e.g. virtual conferencing [Frécon and Nöu 1998] and games [1]. Although new technologies are available, CVEs still present relevant challenges, such as human-computer interaction in the virtual world, real-time rendering and animation, how to control and interact with avatars, among others.

In our platform we use OpenGL for low level rendering process (such as facial animation) and the graphics engine Irrlicht [2] for shading and other functions in the graphics pipeline. The advantage of using a graphics engine for rendering is the optimization and the faster coding. Other aspects are also important in CVEs, such as:

- Avatars animation (body and face);
- voice tools;
- chat tools;
- functions to provide object interaction (Physics);
- human-computer interaction based on Computer Vision (CV);
- real-time frame rates, among others.

In this work, we propose a modular approach, i.e. a number of separate modules are composed together in order to provide the CVE functionalities. The advantage is the possibility of integrating several toolkits, by combining their output in the same application. Our platform helps to create an application for real-time visualization of virtual humans, allowing the best possible interaction among connected people. The CVE is called *Virtual Humans Collaborative Virtual Environment*-(VhCVE).

The paper is organized as follows: related works are described in the next Section, followed by an overview of the VhCVE architecture and its key components. Section 4 presents some results obtained and, finally, section 5 describes the ideas for new components and final remarks.

[1] http://secondlife.com/

[2] http://irrlicht.sourceforge.net/

2 Related Work

Nowadays, a rather well known CVE is Second Life. It offers the exploration of a 3D virtual world, allowing the user to create his/her own avatar, as well as walking and flying to different 3D environments. In this system, it is also possible to use voice and text communication for more realistic human interactions. Other well known CVE is Playstation Home [3]: integrated with every PS3 system, it brings a large community of users allowing them to chat, walk around, see movies, customize their avatars and play games. In the Home world it is possible to see several advertisements such as movie trailers and game releases.

Few surveys have been published in literature about CVEs. In 2004, Emmanuel Frécon [Frécon 2004] presents a very complete overview of CVEs since 1990s. Also, Frécon discusses proposed standards as well as system trends. In 2008, Wright [Wright and Madey 2008] describes APIs, frameworks and platforms used to build CVEs.

The report by [Wright and Madey 2009] led to another publication. The authors propose the survey with two main goals: first, to concisely review several prominent, active desktop-VR technologies, and, second, to recommend the technology or technologies most well suited to building a CVE.

According to the classification proposed in [Wright and Madey 2009], in this paper we describe a CVE platform which uses Irrlicht, OpenCV, OpenGL, and other components, as later described.

3 CVE Architecture

VhCVE performs real-time visualization of virtual humans in 3D environments using a set of libraries and toolkits, where most of them are open source. Figure 2 illustrates architecture components and subcomponents. The components are the main modules in the architecture: Application Manager, Core and State Manager, while subcomponents are related to specific purposes. All subcomponents have an initialization process and most of them contain an update function, e.g. Webcam and Network Managers which are related with image capturing and packet communication, respectively.

The visualization is performed using Irrlicht rendering engine. This engine includes important functionalities and interfaces for communication among components. Irrlicht is an open source engine, and uses either DirectX or OpenGL in order to create 3D scenarios and animations. Other components are described next. One possible output of CVE is the **Video Recorder**, responsible for calculating the elapsed time between each frame and grab a screenshot of the scene. The files can be compressed and filtered using any video processing software. Next Section presents components and subcomponents of VhCVE, while in Sections 3.2 and 3.3 Computer Vision and Facial Animation features are described.

3.1 Components and Subcomponents

There are three main components which are responsible for executing and managing the subcomponents. The **Core** component coordinates the subcomponents initialization and it is also responsible for the display functions. The **State Manager** defines the finite state machine used to control VhCVE, sending events/request to be treated by the **Application Manager**. This last component send tasks to be executed into the subcomponents, as required. In next sections we present further details about subcomponents organized in three main topics: Interfaces, Characters and Visualization.

3.1.1 Interfaces

In DirectX or OpenGL environments, the Graphics Processing Unit (GPU) hardware is represented by a software entity called *device*. The **Device Manager** allows to access the required device from any other subcomponent. One of integrated devices aims to perform physic calculations. Physics is an important part of games, for this

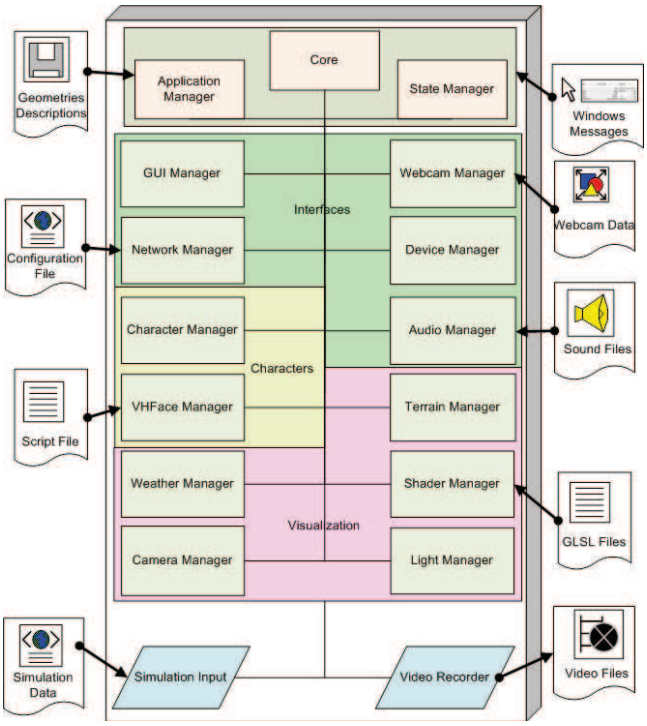

Figure 2: *CVE architecture overview.*

reason we use a library called IrrPhysx [4], which is a PhysX wrapper for the Irrlicht graphics engine.The main idea of this wrapper is to abstract the PhysX SDK methods from the user. Instead, we just have to use a simple interface into Irrlicht. Using PhysX, we are able to simulate the behavior of several kinds of objects like rigid and articulated bodies, cloths, fluids and terrains, which can be interesting in CVE applications. Figure 3 shows an example of rigid bodies with animation.

Figure 3: *Rigid bodies with animation.*

VhCVE has a simple graphics user interface (GUI) allowing access to the framework functionalities in an easy and fast way. The **GUI Manager** subcomponent is based on Irrlicht GUI toolkit classes, and also is responsible for creating and updating GUI components such as labels and buttons.

To perform the Internet connection and voice over IP communication there is a component called **Network Manager**, which uses an input file containing the server IP and port for connection (it is used in the CVE initialization). RakNet is a cross-plataform C++ game network engine [5]. This network engine is easily integrated with Irrlicht. RakNet is responsible for creating the peers, receiving and sending data packets to the server. This server keeps receiving the players positions, rotations, text messages (illustrated in Figure 4)

[3] http://www.us.playstation.com/PS3

[4] http://chris.j.mash.googlepages.com/irrphysx

[5] http://www.jenkinssoftware.com/

and facial status. After that, it sends all the data to the players in order to provide visualization.

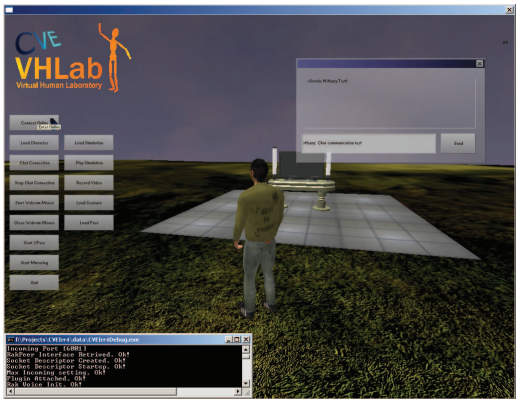

Figure 4: *Chat communication.*

The RakVoice toolkit is a feature of RakNet. This toolkit allows voice communication in real-time among connected users into the chat room. The RakVoice is attached into the RakNet device, after that it is necessary to specify the size of the audio that will be encoded at a time. RakVoice provides the means to encode and decode sound data. In order to listen to the sound, a sound engine is required to provide other audio features such as mute, play and stop. Every time a user speaks in the microphone the audio is buffered and sent to the server, which in turn is responsible for sending it to the other players. Our framework is not capable yet to treat avatars proximity in terms of spatial sound, so in this case the voice communication works for everybody online in the room.

To handle sound output, Irrlicht allows the use of a library called *Ambiera IrrKlang Audio Library*[6]. The **Audio Manager** performs the environment audio and provide support to different audio file formats such as MP3, OGG and WAV. This is the default library for sound output enabling stereo and 3D sound sources. It is important that CVEs support full 3D environmental sounds in a intelligent way. Every sound emitting entity must have a radius distance attached, determining the reachability and the volume of the emitted sounds in order to prevent mixing of distant sounds, hence producing unrealistic environments.

Any webcam can be used to capture images. Then the **Webcam Manager** handle these images as frames and process them using OpenCV allowing to create applications such as CVMouse and Virtual Mirror, explained in Sections 3.2.1 and 4.

3.1.2 Characters

The virtual characters are structured in four parts: body model, bones, animations and facial animation. The **Character Manager** manages this structure in memory and it is in charge of all actions related to characters along the simulation, except the facial animation. The body animation is executed through Irrlicht *AnimatedMeshSceneNodes*. The bone system allows attaching objects to the characters, such as a briefcase or a hat. The facial animation uses other subcomponent called **VHFace Manager** to handle the character's facial expressions. Further details are explained in Section 3.3.

The **Simulation Input** is a module responsible for integrating our CVE with external simulators. It receives the simulation data file and it is responsible for reading and parsing it. This file has XML format and defines a simulated scenario containing positions of virtual characters, among other data. With this feature it is possible to have NPC (Non Player Characters) interacting into the CVE world, e.g. crowd simulation [Musse and Thalmann 2001].

[6] http://www.ambiera.com/irrklang/

3.1.3 Visualization

The **Light Manager** allows determining how the environment illumination should be. It is possible to add and modify light sources customizing the scene and increasing the environment realism. The shadows created by the light sources are also controlled by this component.

In order to increase the graphic quality and realism of the visualization, we can use the **Weather Manager**. This component is capable of creating weather effects such as fog, rain, snow and clear weather. For better visualization results the **Shader Manager** allows the use of different rendering techniques e.g. bump mapping, cartoon shader (illustrated in Figure 5). These shaders are written in OpenGL Shading Language (GLSL) [Rost 2005].

Figure 5: *Cartoon shader example.*

For terrain generation we can use the **Terrain Manager** subcomponent, built upon Irrlicht *TerrainSceneNodes*. This subcomponent manage terrains created by a height map input and also integrate the terrain mesh with PhysX.

The **Camera Manager** subcomponent allows the user to explore the scene in a practical way. It provides three types of camera navigation, also making it possible to change navigation styles at any point of the application. The possible styles are: i) FPV (First person view), allowing to navigate like a first person point of view; ii) TPV (Third person view) aims the camera at the main avatar; and SCV (Static camera view), which fixes the camera, enabling the mouse to control the framework menus.

3.2 OpenCV

OpenCV [Bradski and Kaehler 2008] is largely used by scientific communities for facial tracking among other applications. In this work we propose to integrate Irrlicht and OpenCV in order to provide interaction tools based on computer vision algorithms. Indeed, this integration is very easy since OpenCV and Irrlicht has similar "frame loops", providing a solution without extra callbacks implementation. The image captured through OpenCV can be dealt in Irrlicht display loop, eliminating the need of a new thread or callback function.

3.2.1 CVMouse

The objective of *CVMouse* is to work as a pointer/scratch interface improving the human-computer interaction. Using a webcam and computer vision algorithms, *CVMouse* is able to simulate the states of the mouse. The idea is to track a learned-based color distribution and generate a mouse state, based on previous information.

The *CVMouse* states are, but not limited to: (i) capturing mouse position; (ii) clicking the mouse with the left button; (iii) clicking and holding the left button; (iv) releasing the left button. Basically only the first two stages could be used in machine-interaction applications, but the following ones are implemented with the objective of using them in drawing applications, such as a pencil in a painting system (or to drag and hold things).

In this work we use the YCbCr color space as color representation, but probably many other color spaces could be used, such as HSV or Lab. The idea is to use a color invariant feature to be more robust with illumination changes. Another used information is the area of the tracked object. This can be used to estimate the distance to the

object from the camera. With the information of the position of the object through time and their distance from the camera, we can simulate the states of a virtual mouse.

To learn the color distribution, we fix a colored object in a specific region, as shown in Figure 6, and for the channels Cb and Cr we compute their median value and standard deviations. We do not use the 'Y' information, with the objective to be lighting invariant. It is important to notice that the color of the adopted object must not be visible in any other location of the scene, because the tracking system will track the largest object with the detected color.

Figure 6: *Learning color distribution.*

Creating a color model Let S be a vector (with size = n) composed by the pixel values captured over the predefined region, in the Cb channel. Firstly we compute their median and standard deviation values (m and σ, respectivally).

Before create a color model, we remove some outliers, in a simple way, as described in Eq. 1:

$$S_{new(i)} = \begin{cases} S_j & \text{if } \|S_j - m| \leq 2 \cdot \sigma, \text{ (for } j = 1 \text{ to } n) \\ & \text{else} \end{cases}, \quad (1)$$

then we recompute the standard deviation (σ) over the new vector S_{new}. Thus, we can create a more robust color model. We apply the same method to the Cr color channel. After the outliers removal approach, our color model is composed of $C(m_1, \sigma_1, m_2, \sigma_2)$, where m_1 is the median of channel 1 and σ_1 is their standard deviation (respectively for the second channel).

Background subtraction For each pixel (x, y) in the whole image captured from the camera, we compute the absolute difference over the median values (for the channels Cb and Cr). We define the foreground pixels as the ones whose difference are lower than a predefined threshold ($K = 6$, obtained by experiments), as can be seen in equation 2:

$$B_{Cb(x,y)} = \begin{cases} 1 & \text{if } \|Cb(x, y) - m_1\| \leq K \cdot \sigma_1 \\ 0 & \text{else} \end{cases}. \quad (2)$$

The final binary image is composed by an AND operator over the two channels (B_{Cb} and B_{Cr}).

Some morphological operators (closing and opening) are used to eliminate small artifacts and to close small holes in the resulting binary image.

States definition To simulate the states of the mouse, we monitor the position of the tracked object in time and their size. In our application we use the last N frames ($N = 30$, obtained by experiments) as mouse tail, to generate the states of the simulated mouse. In a general way, each state is described as follows:

- state (i) capturing mouse position: the centroid of the biggest object (after the background removal) is defined as mouse position in each frame;
- state (ii) clicking the mouse with the left button: if, in the last N frames, the position of the mouse do not change very much (it means that it stop to move) and its area increases at a predefined threshold (T_c) and after decrease approximately to the initial area, we assume that the mouse was clicked ($T_c = 3\times$ initial area, obtained by experiments);
- state (iii) clicking and holding the left button: if, in the last N frames, the position of the mouse do not change very much (it means that it stop to move) and its area exceeds a predefined threshold (T_c) and remains, we assume that the mouse was clicked and hold;
- state (iv) releasing the left button: the same as state (ii).

When the user simulates the click of the mouse with the *CVMouse*, the position of the cursor usually changes a lot. To prevent that the user clicks in erroneous positions, we fix the last detected position of the object after the system detect that the object stopped to move. In this way, the user can click in a very specific location. Figure 7 shows an example of object tracking.

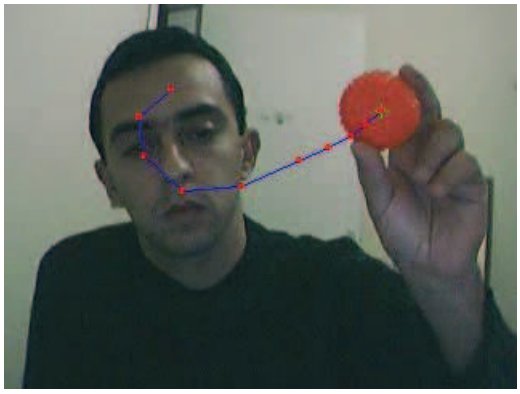

Figure 7: *CVMouse tracking an object.*

3.3 VHFace - Facial Animation

Another challenge in the project of a CVE is to provide real-time generated facial animation to the avatars. The facial animation module, called VHFace Manager, integrates the framework proposed by Queiroz et al. [Queiroz et al. 2009], based on XFace core libraries [Balci 2004]. The framework follows the MPEG-4 Facial Animation (FA) standard [Pandzic and Forchheimer 2003] for parameterization and animation of faces. The standard specifies a set of 84 feature points (FPs) located on the 3D face mesh. A subset of them acts as control points for the 68 Facial Animation Parameters (FAPs), also defined in the standard.

The two first FAPs describe high-level actions (6 facial expressions and 14 visemes) and the remaining deal with specific regions of the face, describing low-level actions, such as "raise left cornerlip" and "close top left eyelid". The FAPs are encoding as numerical values, which are measured by a set distances of key-features of the face, called FAPU (Facial Animation Parameter Units).

A FAP-based animation provides, for each animation frame, the variation of the FAP values. Thus, for each frame, we have a stream of these values. In order to optimize the sending of these streams through the network, a bit mask is also sent indicating which of the 68 FAPs are active (i.e. had values changed) in the frame.

Having the FAP stream, it is necessary to deform the face skin vertexes to produce the animation. Each FAP acts over one FP and its neighborhood (influence zone) vertexes, producing a deformation in the mesh. This means that each FAP value is scaled by its FAPU to provide the displacement of the FP vertex and the vertexes of its influence zone can be deformed through the application of different functions, such as cosine [Balci 2004] and radial basis [yong

Noh et al. 2000; Wu et al. 2006] functions. The information about the FAPU, FPs and their influence zones of a 3D mesh (the Face Definition Parameters – FDP) are described in files called FDP.

In this context, the VHFaceManager has two main modules (see Figure 8):

- The CV2FAP/FDL2FAP module, which receives as input Computer Vision data or high-level face actions and maps it to FAPs, according to Queiroz et al. [Queiroz et al. 2009] methodology. The high-level face actions include facial expressions, lip synchronization and eye behaviors, using the FDL script language (Facial Description Language), also described in [Queiroz et al. 2009].
- The FAP2MeshDeformation module, which receives the FAP streams provided by the CV2FAP/FDL2FAP module or received by the network and performs the deformation in the 3D mesh, according to the FDP model data.

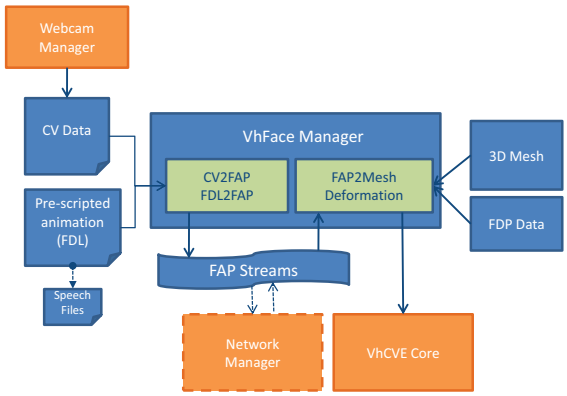

Figure 8: *Diagram of the VhFaceManager Architecture*

Note that FAP values are independent of the 3D model. These values are scaled by the FAPU, providing the displacement of the affected FPs. The influence zone of each FP is affected by a deformation function, also according to the FAP direction. So, a FAP-based animation can be performed by different face models.

4 Results

This section presents some preliminary results aiming to explore the features robustness of the proposed framework. Figure 9 shows an application in a city environment which allows the user to explore it with an avatar.

Figure 9: *VhCVE system display.*

Using the CVMouse model presented in Subsection 3.2.1, we built an application that allows the user to draw in the screen by moving a colored real object, as showed in Figure 10.

We also implemented an application called "Virtual Mirror", in which the characters "mimic" the facial movements of the user. It

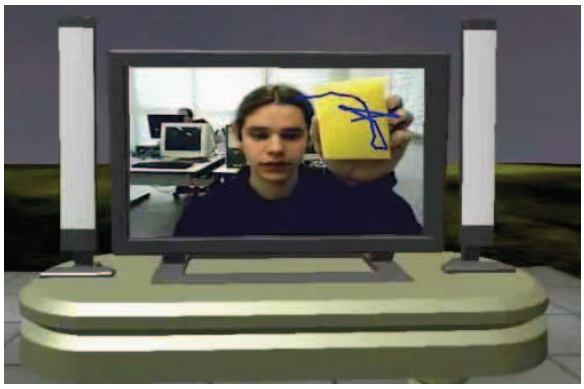

Figure 10: *Illustration of CVMouse-based application.*

explores the functionalities of the VHFace and Webcam Manager modules, using facial feature detector algorithms (for face, eye and mouth regions) from OpenCV [Viola and Jones 2001; Castrillón et al. 2007]. The mapping of the detected features to simple events (opened mouth, closed mouth, smile and direction of the horizontal gaze) and then to FAP animations is implemented according to the methodology of Queiroz et al. [Queiroz et al. 2009]. Once information from face components is acquired, it is used to animate the avatar's face, as shown in Figures 1 and 11.

Figure 11: *Illustration of Virtual Mirror application, integrating OpenCV and VHFace.*

As a result of preliminary performance tests, the framework bottleneck is visible when rendering more than 110 characters. The virtual human animation update is bound to the computer processing unit (CPU), requiring a large amount of CPU time. On the other hand, with static virtual humans (without body animations), the amount of rendered characters has increased to 360 while keeping a frame rate of 24 frames per second. Each virtual human contains 3686 triangles. The results were obtained executing VhCVE on a Intel E8500 Core 2 Duo, 4GB RAM DDR2, equipped with two GeForce 8800GTS OC 320MB in SLI mode.

5 Final Remarks

This paper presented a framework for collaborative virtual environments integrating several toolkits and computer vision algorithms. Moreover, it is possible to use the framework physics and network features to create different styles of multiplayer games, such as first person shooters or online role playing games.

The obtained results were acquired with the current VhCVE version. For the next version, we plan to include new techniques in order to improve the graphics quality and to increase the number of characters rendered in real-time. For instance, new shaders could be used to provide better lighting and also to offer new effects such as realistic water and fire. Also different techniques for shadow creation and weather effects could be implemented. Finally, the use of LOD (level-of-detail) and culling techniques, combined with impostors and rendering acceleration techniques [Ciechomski 2006], could increase the amount of characters rendered in real-time.

References

BALCI, K. 2004. Xface: Mpeg-4 based open source toolkit for 3d facial animation. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, 399–402.

BRADSKI, D. G. R., AND KAEHLER, A. 2008. *Learning opencv, 1st edition*. O'Reilly Media, Inc.

CASTRILLÓN, M., DÉNIZ, O., GUERRA, C., AND HERNÁNDEZ, M. 2007. Encara2: Real-time detection of multiple faces at different resolutions in video streams. *J. Vis. Comun. Image Represent. 18*, 2, 130–140.

CIECHOMSKI, P. S. D. H. 2006. *Rendering massive real-time crowds*. PhD thesis, EPFL.

FRÉCON, E., AND NÖU, A. A. 1998. Building distributed virtual environments to support collaborative work. In *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 105–113.

FRÉCON, E. 2004. *A Survey of CVE Technologies and Systems*. SICS Technical Report T2004:03. Swedish Institute of Computer Science.

MUSSE, S., AND THALMANN, D. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics 7*, 2, 152–164.

PANDZIC, I. S., AND FORCHHEIMER, R., Eds. 2003. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. John Wiley & Sons, Inc., New York, NY, USA.

QUEIROZ, R. B., COHEN, M., AND MUSSE, S. R. 2009. An extensible framework for interactive facial animation with facial expressions, lip synchronization and eye behavior. *Comput. Entertain. (to appear)*.

ROST, R. J. 2005. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional.

VIOLA, P., AND JONES, M. 2001. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on 1*, I–511–I–518 vol.1.

WRIGHT, T. E., AND MADEY, G. 2008. A survey of collaborative virtual environment technologies. Tech. rep., University of Notre Dame - USA.

WRIGHT, T. E., AND MADEY, G. 2009. A survey of technologies for building collaborative virtual environments. *The International Journal of Virtual Reality 8*, 1, 53–66.

WU, Z., ZHANG, S., CAI, L., AND MENG, H. M. 2006. Real-time synthesis of chinese visual speech and facial expressions using mpeg-4 fap features in a three-dimensional avatar. In *INTERSPEECH-2006*.

YONG NOH, J., FIDALEO, D., AND NEUMANN, U. 2000. Animated deformations with radial basis functions. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 166–174.