3D on the WEB and Virtual Humans

Christian Babski, Daniel Thalmann

Computer Graphics Laboratory, Swiss Federal Institute of Technology
CH1015 Lausanne, Switzerland
{babski,boulic,thalmann}@lig.di.epfl.ch

Abstract: It is now possible to explore 3D worlds by using a classical WEB navigator as easily as it is to view 2D HTML pages. The capability to use 3D was previously restricted to powerful machines, it is no more the case. 3D provides a new interface to communicate. The Definition of realistic virtual humans in those worlds produces more intuitive interface for neophyte users. In this paper, we propose a set of tools to define virtual humans and bring them to life by using a standard 3D language for the WEB.

1 Introduction

Until 1995, information on the WEB was exclusively in two dimensions, using hypertext to organise WEB pages. By the emergence of a new standard language known as Virtual Reality Modelling Language (VRML), it becomes possible to define a three dimensional world that everybody can examine on the WEB, by using the same navigator as before. The availability of the third dimension opens new doors for the exploitation of the WEB. Numerous disciplines such as architecture or chemistry are widely using 3D to visualise models of house or molecules. By using VRML, such models can be now distributed on the WEB and viewed by anybody. By making disappeared classical file format problems links to data exchange, VRML introduces a new dimension to the information diffusion. Over the function of a standard exchange format, the capability of VRML to manage events and to establish link with other standard language such as *JAVA*, gives the opportunity to use collaborative work through a classical WEB. If people were talking about modelling a VRML worlds few years ago, it is now justify talking about developing a VRML application.

2 Background

The initial process for making available 3D on the WEB was initiated at Siggraph'94. The first version of the Virtual Reality Modelling Language (VRML) is born the year after. It was chosen to take advantage of the knowledge of *Silicon Graphics Inc.* in 3D by adopting its *Open Inventor* language syntax. VRML 1.0 is a subset of the *Open Inventor* file format enhanced with few functionalities linked to the WEB (like the ability to link a 3D object to a normal HTML page through the use of an URL). Basically, VRML 1.0 is just a file format that permits the description of 3D scenes. It was voluntarily decided not to include any kind of animation capabilities in order to focus on the definition of a reliable kernel that can be enhanced afterwards. There was a lack of applications being able to display the new file format and 3D hardware acceleration on PCs was not as developed as it is today. The more comfortable computer to visualise this new WEB format was *Silicon Graphics Inc.* workstations through an application called *WebSpace*. This application was running as a stand-alone application invoked by the WEB browser.

The enhancements of VRML 1.0 were finished for Siggraph'96 where VRML 2.0 [1] was introduced. The main interest of VRML 2.0 resides in the fact that it is not anymore a simple 3D file format. It includes a system of events that permits to establish a kind of communication between any VRML objects. Another important innovation is the ability to

include a piece of program inside a VRML file as well as the ability to establish a connection between an external program and the VRML application. The outcome of VRML 2.0 was complemented with a set of new plug-ins. Then, as it was already the case for 2D by using HTML or some other 2D animation formats, VRML can be displayed on any machine by using these plug-ins for WEB navigators (Blaxxun [2], CosmoPlayer [3], Cortona [4]) or still stand-alone applications (VRWave [5]). VRML files are downloaded and parsed by the VRML browser in order to be displayed.

The capability of VRML to be updated from an external application written in *JAVA* (presented in section 3) gives the possibility to build shared virtual environments very easily: in this kind of world, several participants can see each other and in a certain way they can interact. The VRML browser ensures all the display where the network part is ensured by using *JAVA* libraries. Such multi-users worlds [10] [11] [12] are usually based on a client-server architecture. As soon as we are talking of shared virtual environment, the notion of avatars (user graphical representation) is just underneath. Avatars can have very original shape [13] [14]. But as soon as we want to use VRML to perform specific tasks (like student's registration by using a virtual administration desk [15]), the way participants or even, autonomous characters are represented is important. These worlds will be used by end users that are not familiar with 3D and shared environments, which means that it is useful to be able to have realistic virtual humanoids. The VRML Humanoid Animation working group (HANIM) [16] defined a generic way to describe realistic virtual humanoids in VRML. This proposal is partly based on a body hierarchy defined at *the Computer Graphics Laboratory* [17], which is briefly presented in section 3, and on *Jack* body [18]. Once a body is available as a hierarchy in VRML format, it is possible to animate it, still by using VRML capabilities. We present our work on body control and animation in VRML worlds in section 4. Finally, we highlight the difficulty related to VRML browser implementations, which rarely fully include VRML specifications.

3 VRML Techniques

3.1 Events

In VRML, it is possible to send/receive event to/from objects. Each kind of objects (known as node: e.g. *Group*, *Material*, *Texture*) have a set of events it can receive and/or send. For instance, a *Transform* node can generate a *translation_changed* event (which specifies an update of the position of the object) and can receive a *set_translation* event which updates the position of the object. VRML world builders can then define which outgoing event field will be sent to which incoming event field in order to make nodes communicating by using a *route* command.

The *route* command can be defined in a static way, declared in the same way as classical objects. It can also be defined dynamically, by using a *Script* node. It is then possible to add a *route* command or to delete an existing one at runtime. This can be useful to attach an object to a moving character, just by routing the position of the body to the position of the object. When the actor release the object, the script can delete the *route* command and the object will stay in place. *Script* node gives the possibility to perform intelligent changes on VRML world. By including the capabilities of a programming language, updates of the world can be the result of a complex computation instead of a direct *route* of a basic information from one node to another. The *Script* node can contain several types of scripting languages :

- Interpreted language: *vrmlscript* [6], *ECMAscript* [7] which are using the JavaScript Authoring Interface (JSAI)

- Compiled language: *JAVA* [8] (which is the most powerful solution) by using an *External Authoring Interface* (EAI) [9] which define the way it is possible to communicate with the VRML world.

The list can be enlarged, depending on the VRML browser implementation.

The following figure (Fig. 1) gives an idea of what VRML permits in terms of events communication.

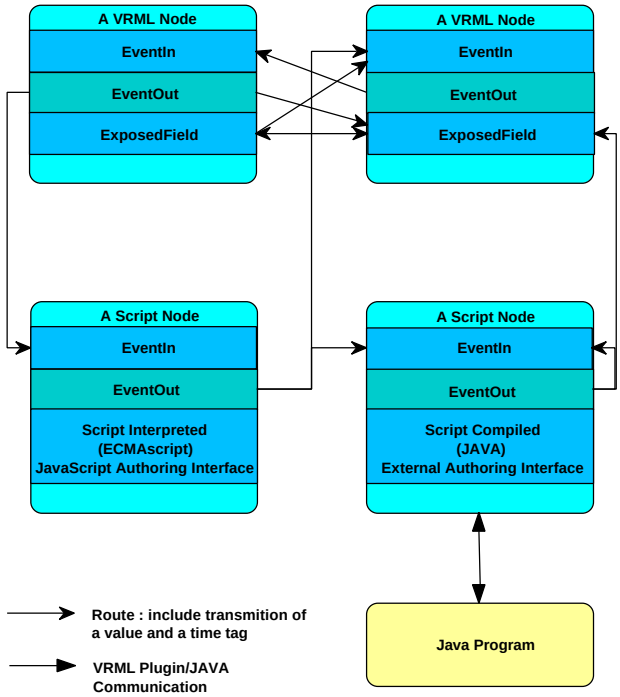

Fig. 1 - Set of legal event's routes.

Routes are only possible between events of the same type: it is not legal to send a *translation_changed eventOut* to a *set_rotation eventIn*. *ExposedField* is a combination of both *eventIn* and *eventOut*. It is capable to send as well as to receive an event. An important point is that a *Script* node, as well as classical nodes, is able to communicate with another *Script* node, or even it can intercept an event that it generates. There is also no limitation from the number of source or destination for a *route* information. A same event (in or out) can be the source or the destination of several *routes*. One event can launch a set of actions as well as be the destination of a set of *eventOut*. This implies the presence of a time tag for each event in order to deal with synchronisation problem inside *Script* nodes.

3.2 Sensors

A set of different type sensors is available in VRML. They mainly permit to intercept user interaction. According to their function, they can generate events that can be routed to a script or directly to other nodes. It is possible to define three categories :

- Manipulation sensors: when attached to an object, they generate different kind of transformation (*SphereSensor*, *CylinderSensor*, *PlaneSensor*) computed from user actions on the object. By routing those transformations to the object's position/orientation, it gives the impression of manipulating the object.

- Space sensors: they generates events according to the user position. It is possible to know if an object is visible for the user (*VisibilitySensor*), touched by the user (*TouchSensor*) or if the user is close to an object (*ProximitySensor*).

- Time sensor: introduces a notion of time passing in VRML scenes. It is useful for synchronisation of different events or animations. It acts like a clock with a set of controls linked to a set of *eventIn* and *eventOut*. The user can set the starting time, the stopping time or duration. When running, a *TimeSensor* generates time ticks.

3.3 Prototype

VRML prototype gives the possibility to create parameterised objects. It is composed of a header containing parameters and an associated piece of VRML code. Along this VRML code, some node characteristics point to prototype's parameters. . Once the prototype is defined, it becomes a legal node you can define in the scene with different set of parameters (Fig. 2). Each time the user wants to create a new instance of this prototype, the default values can be used as well as some new one by adding them in the instance call. A prototype can contain any VRML nodes (including *script* node) or other prototypes. It is then possible to define prototypes to apply a particular behaviour to any objects by embedding a script in the prototype for example.

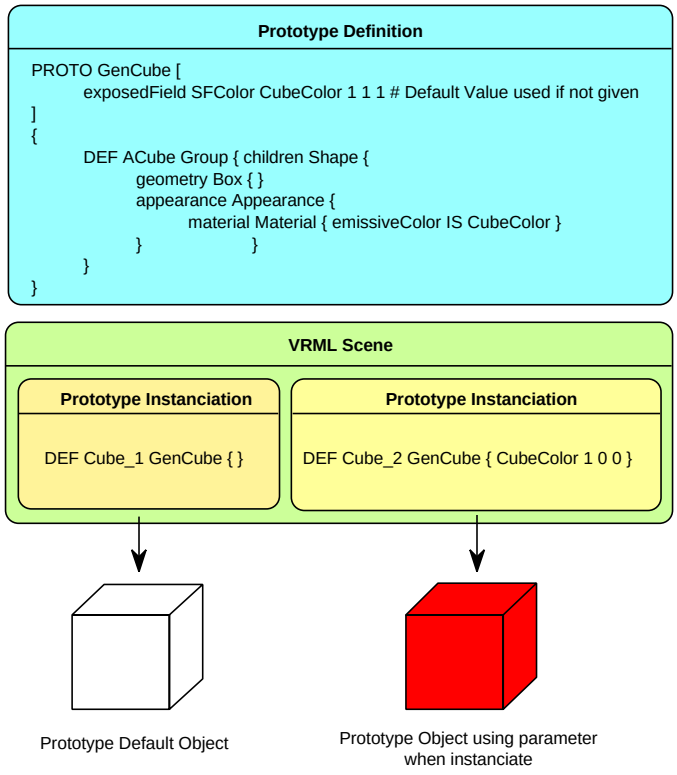

Fig. 2 - VRML Prototype system

4 The Articulated Body Model

The articulated body is defined as a hierarchy composed of 68 (62 + 6 for global position) degrees of freedom (plus 30 degrees of freedom for each hands when included) (Fig. 3). It is defined as a specialised version of a more generic scene graph management library which allows to define any kind of skeleton (virtual humans as well as virtual animals) [17]. This hierarchy was first exported to VRML in order to retrieve the same hierarchy with the same

movement capabilities inside VRML worlds. A body definition is based on a template that permits to obtain different kinds of body with different physical characteristics (size, weight...). As a so detailed hierarchy is not always needed for an application, any subset of joints is legal. Nevertheless, a minimum number of joints is specified in order to keep a humanoid shape (legs and arms are always present and can not be removed without loosing some motion capabilities for the final virtual humanoid). This implies that choice mainly affect the definition of the spine which can be reduced to the *vl2* joint. In HANIM 1.0 specification, no fixed level of details concerning the hierarchy was defined. In the last HANIM 1.1 document, a set of predefined levels of articulation for a body definition is described. It clearly defines the minimum legal set of joints that should be included to be HANIM 1.1 compliant.

The exported hierarchy complies with the standard definition of human bodies inside VRML according to specification of the HANIM working group [16]. Information concerning movement limitation, limb centre of mass and other information that can be useful for body movement generator (like inverse kinematics or physically based) is included inside the body hierarchy itself through the use of VRML prototypes.

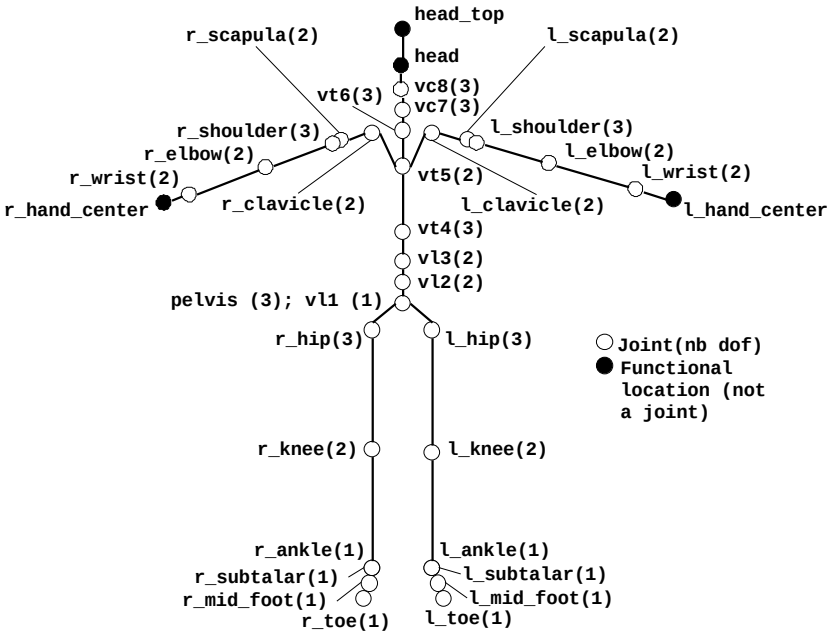

Fig. 3 - body hierarchy set of joints (number of degrees of freedom).

A set of surface are then designed and attached to their corresponding joint (Fig. 4).

There are two main types of prototypes defined to be used in a VRML body hierarchy:

- The *Joint* prototype: each sphere of the skeleton (Fig. 4) corresponds to a joint node. This node contains the mobility information (movement limitation, centre of rotation…) plus a specific name, which is useful for body analysis (determining present nodes) and animation feature. This name is generic and is fixed for each body part. When users want to animate such a standard body, they access to the desired limbs through their standard names and then update rotation (see below in the chapter on animation). By default, such a joint embeds three degrees of freedom, but by adjusting associated information concerning angle limits, it is possible to forbid any undesired degree of freedom. Children nodes of the joint can be other joint nodes or a segment node.

The *Segment* prototype: it is supposed to contain the geometry attached to the corresponding joint (its parents in the body hierarchy). The segment prototype receives body parts. This node can include levels of details for each body part.

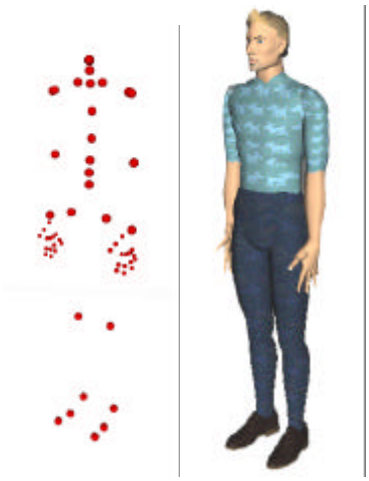

Fig. 4 - A human skeleton and a human body, with a graphical representation, inside VRML (each sphere corresponds to a joint with one, two or three degrees of freedom).

The definition of a virtual humanoid is completed by defining a body prototype. This prototype contains the body hierarchy itself, the list of included joints and included segments, viewpoints and sites (information for inverse kinematics). The specification defines the maximum number of joints and segments defining a body. This *Humanoid* prototype, used as a parameter in animation functions described below, permits to fit animation prototype with bodies. Any extension to the classical body definition, which only describes hierarchy and geometry, can be added. As these virtual humans are supposed to interact with the surrounding world, some functionalities such as grasping an object or walking should be considered. This is discussed in the next section. Associated to such a body definition, a *JAVA* class can be defined (Fig. 5). It is in charge of retrieving pointers to joints and segments and storing information inside *JAVA* structures. This class can be attached to any HANIM bodies. On the top of such a basic class, any *JAVA* program can take the control of the body hierarchy to add functionalities for animating the body. By extending this class, it avoids to each program to have to retrieve on its own the needed information on the body structure to perform its task.

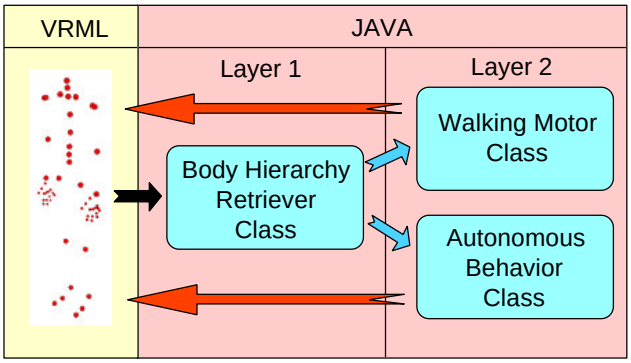

Fig. 5 - Body Hierarchy Retriever Class.

4.1 Direct Animation

VRML proposes various ways to animate a 3D object (in our case, a virtual human). The first very basic means is the notion of *sensor*. We have used it to integrate our bodies inside VRML scenes (Fig. 8). This *sensor* can be linked to one (or all) body part and acts as a virtual tracker. The user is able to move any limb just by picking it and moving the mouse (this implies sending update rotation signals from the sensor to all involved body parts). This

body animation tool illustrates the transformation dependencies on a classical hierarchy (Fig. 6).

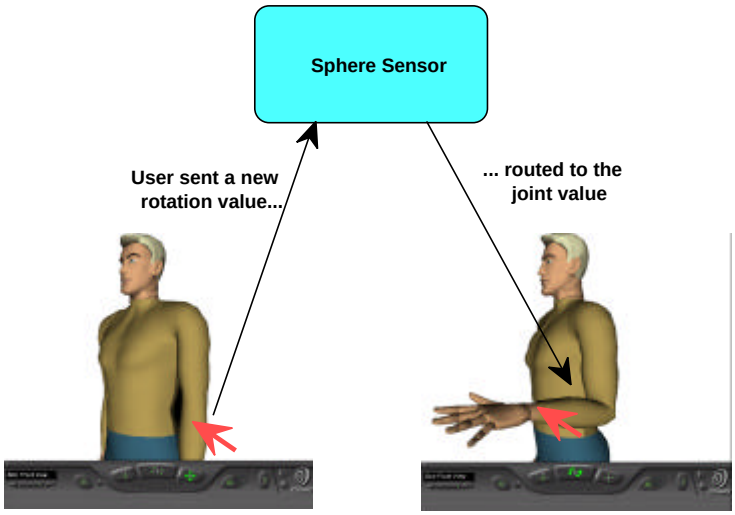

Fig. 6 - Real-time control of a body by using *SphereSensor* node.

By adding a control on angle value limits extracted from the *Joint* prototype, this kind of direct interaction can be useful to test if a human body shape fit well in a given environment (new design sit for example - Fig. 7).

Fig. 7 - An actor manually positioned to test the shape of a chair.

4.2 Interpolated Animation

Another way to generate animation in VRML is to use interpolation fields, which permit to linearly interpolate VRML node characteristics. For example, using an associated interpolation node can change the translation or the rotation field of a *Transform* node. Similarly, the transparency or the colour field of a *Material* node can evolve over time. These interpolation nodes use *TimeSensor* nodes that insert a notion of time inside VRML worlds. They include a set of keytimes associated to key values determining a keyframe-based animation. By using events, new values are routed to each body parts in order to update body posture, creating the final animation.

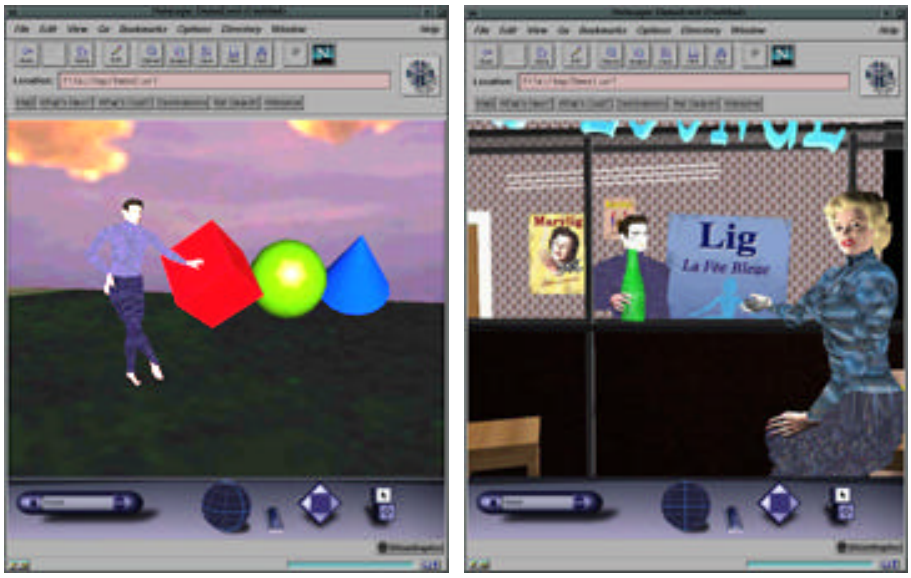

Fig. 8 - Static bodies in a VRML scene (manually positioned using *Sensor*.)

Fig. 9 - An animated body presenting a set of administrative documents in a room inside the virtual EPFL [15].

Fig. 10 - A live motion capture applied to a character and a skeleton [21].

By using this system, all existing animation, created using our body motion software [19] [20], can be exported to VRML and applied on a HANIM virtual body. This includes keyframe-based animation (Fig. 9) as well as motion capture sequence (Fig. 10).

This animation scheme defines static animation, meaning that, once the animation is defined, it does not evolve. Due to the standardisation work, these animations can be applied to any HANIM compliant body without any change, by embedding them inside a specific VRML prototype we present now. By passing pointers on involved humanoids through the prototype parameters system, needed routes to perform the animation are generated dynamically. We also increase the control of the animation at the time level. It is possible to launch animation at any time, to synchronise animation with another one and to apply the same animation to several bodies at the same time without defining other references. This gives the opportunity to define a set of useful animation [22] that can be used by anybody in order to test or animate a HANIM compliant body in a given VRML world.

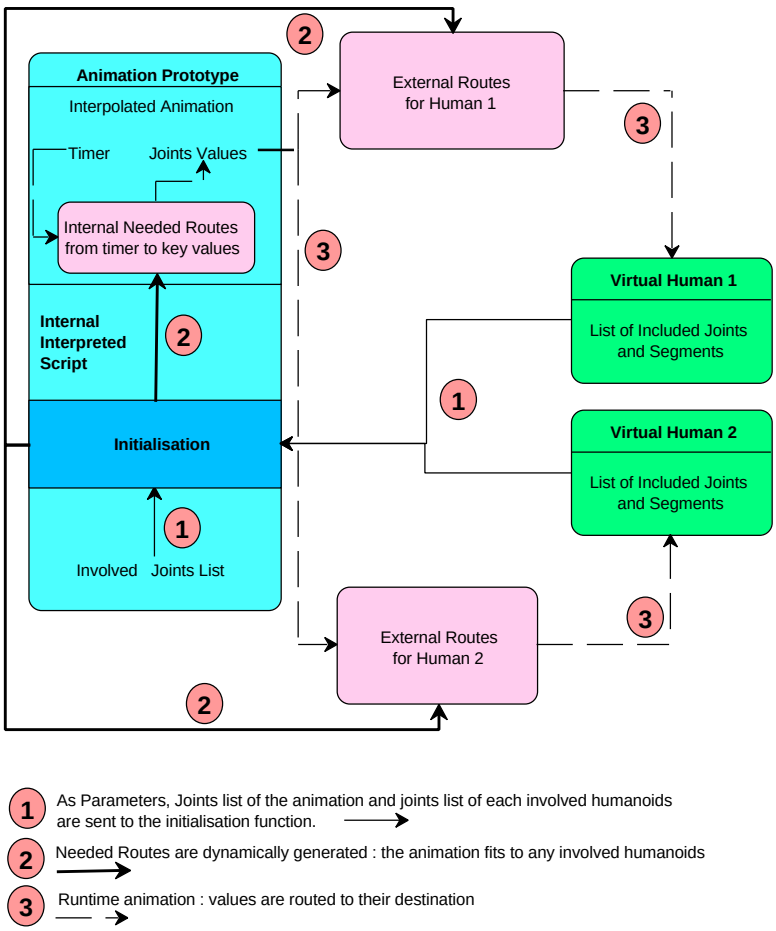

Fig. 11 - The animation prototype: dynamic routes creation system.

Note that the list of humanoids to animate is sent to the animation prototype. As described in the previous paragraph, a particular animation sequence can be adapted to fit the number of joints available in the virtual human model. The routing map for events which are generated by the animation over time, is created dynamically, based on the joint list defined in the *Humanoid* prototype. The first task of our animation prototype is to create needed *routes* for each actor according to their list of present joints. Then, as soon as the animation is launched, events are followings those *routes* (Fig. 11). The generation in a dynamic way of needed *routes* results in an economy of bandwidth by avoiding transmitting these data over the network at the loading time.

This kind of animation needs a lot of data in order to animate a fully defined character. Typically, for a full HANIM skeleton, such animation includes a set of 47 interpolation arrays (45+2 (global position) joints for the body plus 15 joints for each hand if included).

These arrays store key-times and rotation values (axis angle coded on 4 float values). Even adjusting the number of frames per second would not allow obtaining an acceptable file size as soon as we want to code complex animation (of a realistic duration) incoming from a captured motion capture for example. Another point is that, according to the load of the machine, the browser adapts automatically the frame rate when playing animation. It can result in a movement sampled at 2 Hz if the machine is over loaded.

4.3 On The Fly Animation

The most efficient way to animate virtual humans in VRML is to use animation motors. This permits to code directly an algorithm to animate virtual bodies. Like previously presented, it can be done by using an interpreted language (like *vrmlscript*) or by using an external JAVA program that connects to the VRML plug-in. In both cases, it is possible to modify a position or an orientation of a VRML object, as well as adding or removing objects from the VRML scene. We will present two examples (one for each method): the first one is exploiting the *vrmlscript* language for performing real-time facial animation, where the second one is based on a JAVA program in order to perform real-time motion capture using VRML.

4.3.1 Facial Animation

This facial animation system is based on a VRML prototype proposed in the HANIM 1.1 specification. The basic prototype definition is linked to a given 3D surface. The prototype, called the *displacer* prototype, is composed of two arrays:

- The first array contains a subset of vertices composing the associated surface. It is possible to include the entire surface. Vertices are referenced with their indexes in the definition of the surface.

- The second array is linked to the first one. To each vertex referenced in the first array is associated a 3D displacement in the second one. This displacement defines the movement that can be applied to the associated vertex to obtain a “deformed” position.

This prototype mainly permits to perform local deformation like those implied when realizing facial animation. Even if it is theoretically possible to deform an entire surface with such a prototype, the control of the global deformation is not easy in this case and the result is usually not realistic when performing virtual body animation.

When trying to use such a prototype to generate facial animation, a limitation appears. In order to obtain complex facial expression, it is needed to use an entire set of these *displacer* prototypes. The problem is that a same vertex can be involved in several *displacer* prototypes, and in this case, nothing is specified concerning the way *displacer* prototypes should be combined. We have proposed an extension to the default specification in order to face this problem. Our solution resides in the definition of an additional VRML prototype called the *displacer group* prototype. This prototype will group all *displacer* prototypes that are modifying the same surface. To the *displacer* prototype, we add a parameter that defines the amount of *displacer* to be applied. It permits to use totally as well as partially the embed deformation. Then, instead of modifying directly the surface, *displacer* prototypes are sending their modifications of the surface to the *displacer group* prototype. It will take in account all modifications at once, composing them and finally deformed the surface with the resulting deformation (Fig. 12).

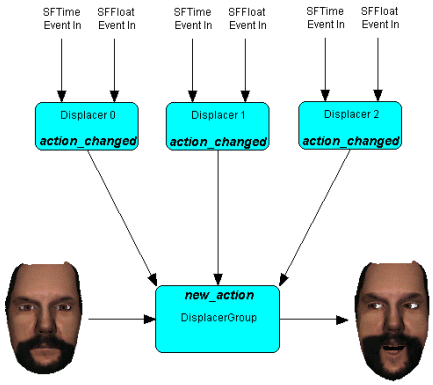

Fig. 12 - The *displacer* group prototype action.

By defining a set of basic *displacer* prototypes, it is possible to obtain a complete set of different facial animations (Fig. 13). This system permits to generate an infinity of different facial expressions with the same amount of data, whatever the length of the animation. This demonstrates the ability of VRML 2.0 to face CPU consuming task like surface deformation. By adopting a JAVA solution, it is even possible to apply real-time deformation to an entire virtual human body [25].

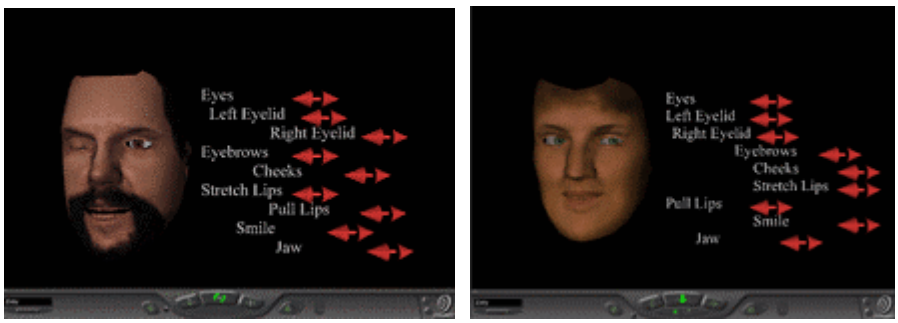

Fig. 13 - Two VRML faces with the same set of *displacer* prototype [23].

4.3.2 Motion Capture

Motion capture is another example that can demonstrate the ability of an application based on a WEB context and using VRML to perform a complex task in real-time. Motion capture permits to animate a virtual character with movements performed at the same time by a real person. This implies the use of a motion capture system that is able to retrieve the posture of the real person. In our case, we are using a magnetic system called *MotionStar* from *Ascension Technology* [24]. Magnetic sensors are attached to the different body parts of a real person. The system is creating a local magnetic field within which the real person has to move. It is then possible to obtain from the system the location and the orientation of each magnetic sensor at any time. Based on this system, we have developed a motion capture application using JAVA and VRML.

A motion capture session can be decomposed in two steps:

- The calibration phase: it is needed to make a correspondence between the initial posture of the real person and the initial posture of the virtual body (like described in the HANIM 1.1 specification). Initial postures of both virtual and real body are very close (theoretically the same). What the calibration phase is recording, it is the initial orientation of each sensor in the initial posture. This initial orientation will be taken in account in the second phase (Fig. 14).

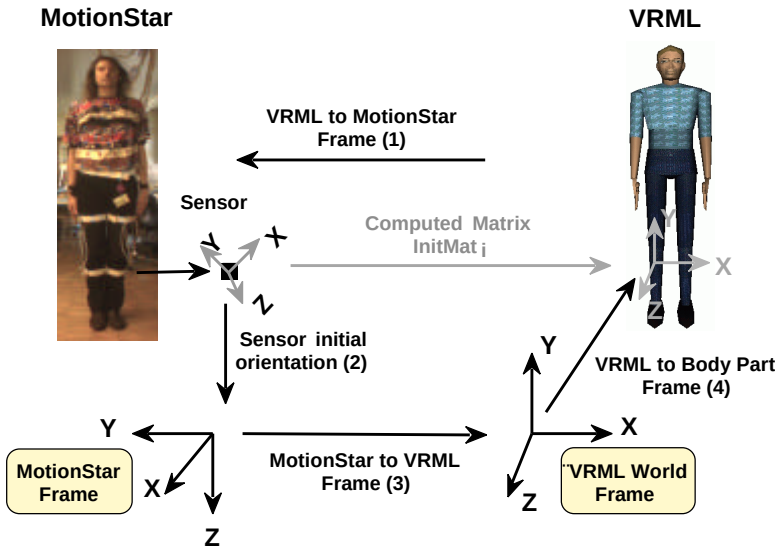

Fig. 14 - The calibration phase, generating $InitMat_i$ matrices to be used in the second phase.

- The animation phase: in this stage, data are retrieved from the system, transformed by using initial data computed during the calibration phase and converted to be applied to the HANIM compliant body. The final conversion mainly consists in generating relative movement, where we obtain global movement. As the object to animate (the virtual body) is a hierarchy, the movement of each body part is computed relatively to its parent.

Fig. 15 - Motion capture session.

To fully animate a virtual body, fifteen sensors are needed (body parts). In order to save computing time, it can be chosen to animate only upper body parts for example. Another optimization can be done at the motion capture system level. It is possible to obtain the position as well as the orientation for each involved sensor. Because we are animating a hierarchy, the spatial location of a body limb is computed by going through all parents of a given joint. Given the fact that sensors are not precise enough to be able to exploit the position in addition of the orientation (in order to get a more precise final posture), we can ask to the system not to send position, but only orientation. This will decrease the load of the communication between the motion capture system and the JAVA application. In our case, the communication was performed through a 10Mbits/s network link it asks for 25ms to get orientation for fifteen sensors. In addition of this time, we have to take in account the needed time to compute final data that can be sent to the VRML world for updating the virtual body posture. This takes an average of 38ms. This permits to reach an acceptable frame rate of 15 frames a second. But this does not take in account the needed time to display the 3D scene. This is totally linked to the hardware, which is evolving very fast these last years concerning

3D acceleration. By taking in account the evolution of PC, it is reasonable to think that it will be possible to obtain 25 frames a second without any problem soon. A more complete description of our system can be found in [26].

5 Conclusion

5.1 Results

We have demonstrated that by exploiting capabilities of VRML 2.0, it is possible to obtain a complete set of features for 3D management on the WEB. This includes basic animation system based on interpolated animation as well as complex animation motor involving real-time constraints.

Major problems encountered when using VRML is mainly linked to VRML plug-ins implementation. It is very hard to obtain a given system/application using the VRML technology that is working on all available VRML plug-ins. Usually, a WEB page proposing this kind of 3D features includes some recommendation on the preferred VRML plug-in to use in order to see the 3D world correctly. Problems can vary from incorrect light or color (different color for the same object from one plug-in to another for example) to a complete incompatibility if the script language used in a VRML file is not supported by the plug-in. These differences from one plug-in to another are mainly linked to the complexity of VRML. VRML is a very complete language, and some particular points in specification like lights or the events system are very complex to implement. Concerning lights for example, plug-ins usually have to use more or less precise estimation to be able to compute shading in real-time. Depending on these choices in the implementation, the result will differ from one application to another. From the point of view of artists, this is very painful to manage. Another problem is also linked to instability of the association plug-ins/WEB browsers that crashes often. We can notice that major problems are linked to plug-ins or WEB browsers themselves and not directly to the VRML 2.0 specification. An amelioration of actual VRML plug-ins in terms of stability and compatibility can increase the success of the 3D on the WEB. In the actual context, it is very difficult to sustain VRML for commercial application, as it is not possible to affirm that the final result will work in any context (like it is the case for the entire set of 2D media on the WEB – movies, animation…).

5.2 WEB3D Evolution

Based on experiences and problems exposed in the previous section, a movement called X3D [27] appeared in order to specify the next generation language for defining 3D on the WEB. The choice was made to propose a scalable language. Each user are supposed to fit the X3D language to their own needs. The X3D working group has defined a minimum set of features that has to be included in a viewer to be called X3D compliant. Then, through the use of modules, the basic set of features can be enhanced to include some additional capabilities. These modules are defined using the XML Document Type Definition (DTD) system [30] [31]. The support of the entire specification of VRML 2.0 can be one of them for example or the HANIM support can also be another possibility for a module. This simplification gives the ability to obtain X3D viewers written in java and exploited within a normal applet. This is the case for two available proposals that defines what can be a X3D kernel: both Shout3D system [28] and Blaxxun3D system [29] are proposing implementation examples within simple java applets. The kernel does not include any navigation information. It is in charge of the final application (applet) to implement a given way to navigate within the X3D world. It avoids having to install heavy applications that include features that are not useful for the end-user. At the same time, it avoids the instability linked to VRML WEB plug-ins. JAVA does not have this problem as it is embedded within WEB browsers.

From a practical point of view, X3D appears to be a VRML-like grammar embed in XML tags. The kernel of X3D includes only a subset of VRML 2.0 features. This is why it will be necessary to have an extension module to be totally compliant with VRML 2.0. Some actual implementations of the proposed X3D kernel are including a very restricted number of VRML nodes. For example, none of the VRML primitives (cube, sphere, cone…) are included in the Shout3D proposal where they are available within the Blaxxun3D proposal. The use of XML DTD permits to easily enhance the X3D language with some specific features. It will be possible for everybody to define a new set of 3D nodes just by defining the associated XML DTD that is describing their characteristics. The only way to do such a thing in VRML 2.0 is to use prototypes, which are still defined using VRML syntax with its problems (types incompatibility for example).

6 Acknowledgements

We would like to thank Mireille Clavien for body designs, Patrick Keller for his work on scene design and Tom Molet his presence.

7 References

[1] - VRML 2.0 specifications, http://www.vrml.org/Specifications/
[2] - Blaxxun VRML browser, http://www.blaxxun.com/
[3] - CosmoPlayer VRML browser, http://cosmosoftware.com/products/player/brief.html
[4] - Corona VRML browser, http://www.parallelgraphics.com/
[5] - VRWave VRML browser, http://www.iicm.edu/vrwave
[6] - *vrmlscript* reference page, http://cosmosoftware.com/developer/vrmlscript.html
[7] - ECMAScript reference page, http://www.vrml.org/Specifications/VRML97/part1/javascript.html
[8] - *JAVA* reference page, http://www.sun.com/java/
[9] - *EAI* reference page, http://www.vrml.org/Specifications/VRML97/part1/java.html
[10] - VNET Multi-users VRML system, http://ariadne.iz.net/~jeffs/vnet/
[11] - Blaxxun Multi-users VRML system, http://www.blaxxun.com/
[12] - Community Place Multi-users VRML system, http://sonypic.com/
[13] - Avatars examples, http://www.avatara.com/avatars/index.html
[14] - Avatars examples, http://www.digitalspace.com/avatars/index.html
[15] - Virtual EPFL school, http://ligwww.epfl.ch/~babski/vrml_epfl.html
[16] - HANIM working group, http://ece.uwaterloo.ca:80/~h-anim/
[17] - R. Boulic, T. Capin, Z. Huang, L. Moccozet, T. Molet, P.Kalra, B. Lintermann, N. Magnenat-Thalmann, I. Pandzic, K. Saar, A. Schmitt, J. Shen, D. Thalmann, “The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters”, Proc. Eurographics ‘95, Maastricht, August 1995, pp. 337-348.
[18] - P. Lee, C. Phillips, E. Otani, N. I. Badler, “The Jack interactive human model”, Concurrent Engineering of Mechanical Systems, Vol. 1, First Annual Symposium on Mechanical Design in a Concurrent Engineering Environment, Iowa City, USA, 1989, pp 179-198.
[19] - T. Molet, R. Boulic, D.Thalmann, “A Real-Time Anatomical Converter for Human Motion Capture”, Proc. 7th Eurographics Workshop on Animation and Simulation, Springer-Verlag, Wien, September 1996, pp. 79-94.
[20] - R. Boulic, Z. Huang, N. Magnenat Thalmann, D. Thalmann, “Goal Oriented Design and Correction of Articulated Figure Motion with the TRACK system”, Computers and Graphics, Pergamon Press, Vol. 18, No 4, 1994, pp. 443-452.
[21] - Keyframe and motion capture examples in VRML, http://ligwww.epfl.ch/~babski/StandardBody/world_final.wrl

[22] - Ready to use VRML animation, http://ligwww.epfl.ch/~babski/StandardBody/
[23] - Facial animation example, http://ligwww.epfl.ch/~babski/StandardBody/FacialAnimation/index2.html
[24] - Ascension Technology, MotionStar System for Motion Capture, http://www.ascension-tech.com/
[25] - C. Babski, D. Thalmann, “A Seamless Shape For HANIM Compliant Bodies”, Proceedings of VRML’99 Symposium, ACM Press, Paderborn, Germany, pp 21-28.
[26] - C. Babski, D. Thalmann, “Real-Time Animation and Motion Capture in Web Human Director”, Proceedings of Web3D & VRML 2000 Symposium. ACM Press, Monterey, California, pp 139-145.
[27] - X3D Working Group, http://www.web3d.org/x3d.html
[28] - Shout3D, http://www.shout3d.com/
[29] - Blaxxun3D, http://www.blaxxun.com/c/s?cat=7&sub=3&url=/products/blaxxun3d/index1.html
[30] - E. R. Harold, “The XML Bible”, IDG Books, 1999.
[31] - I. S. Graham, L. Quin, “XML Specification Guide”, Wiley Edition, 1999.