ANIMATING AND RENDERING VIRTUAL HUMANS
Extending X3D for Real Time Rendering and Animation of Virtual Characters

Yvonne Jung
Fraunhofer IGD, Darmstadt, Germany

Keywords: Rendering, Virtual Characters, Animation, Simulation, Virtual Reality, X3D.

Abstract: In this paper, we focus on the different aspects of real time visualization and animation of realistic virtual characters. The common goal was to come up with solutions based on the concepts of the open ISO standard X3D and if necessary to propose generalized extensions to the standard. First we describe the high level control language PML and its implementation, which is also suitable for non-graphics experts. Then we focus on realistic rendering, how X3D must be extended to allow special effects and realization of novel rendering algorithms, including skin and emotion rendering. Next we explain the challenges of dynamics related to virtual characters covering play-back and blending of pre-defined animations, online simulation of locomotion and last but not least hair simulation.

1 INTRODUCTION

Virtual characters are well known from movies, ads and computer games. While most of these movies feature non-realistic characters, e.g. Toy Story, there are already some showing virtual humans with very high realism, e.g. Final Fantasy. In first person shooters like Half-Life 2 we often see realistic virtual characters. But current solutions have major drawbacks and research is still needed. Today, extreme realism can only be achieved in movies, where real-time is not an issue at all. Virtual characters in movies and games are the result of many month of work of artists. In computer games movements like mimic, gestures, and locomotion are limited and do not allow expressive, non repetitive behaviors. Controlling virtual characters needs a lot of CG knowledge. Our vision is a widespread deployment of interactive and reactive virtual characters, but today it needs expertise, much time and effort to use them in an application. As long as this does not change, we will see characters in movies and games only. Thus we are working on solutions to overcome the major barriers concerning rendering aspects. All the work described in this paper was carried out in the project Virtual Human, where we were responsible for the graphics output, meaning the rendering of scenes and realistic virtual characters as well as playing back animations.

An important topic was to provide an abstract layer to the graphics environment, because controlling of character behavior and emotional states was done by researchers in the area of Artifical Intelligence and Digital Storytelling. Understandably they did not want to bother with low-level graphics and demanded a high level control language where they could issue commands like ”character one waves hand and simultaneously blinks with his eyes”. Therefore we developed the PML language corporately, and in chapter 2 we will focus on its design from a graphics perspective. Methods for realistic rendering will be described in chapter 3. Since realistic characters move around, gesture and show emotions, dynamics are also essential, see chapter 4 for further details. Lessons learned, a summary and outlook for future work conclude this paper. A common goal was not to develop methods only suitable to use in our own specialized environment. We also thought about how virtual characters could be realized effectively when using a common standard for interactive 3D like the open ISO standard X3D. In case it was necessary to extend the standard, we did also provide appropriate X3D binding specifications.

2 CONTROLLING GRAPHICS

When visualizing virtual characters one also has to think about interoperability aspects. PML (Player Markup Language) is an XML-based Markup Language which takes up concepts from RRL (Rich Rep-

Jung Y. (2008).
ANIMATING AND RENDERING VIRTUAL HUMANS - Extending X3D for Real Time Rendering and Animation of Virtual Characters.
In *Proceedings of the Third International Conference on Computer Graphics Theory and Applications*, pages 387-394
DOI: 10.5220/0001093503870394

resentation Language) for the description of agent behavior in net environments (Piwek et al., 2002). It is an abstract specification language which is independent of the implementation of the player and the virtual environment and is used as descriptive interface markup language between a dialog engine and a graphics engine. PML is a language for controlling virtual environments with special regards to character animation and user interaction, and therefore defines a format for sending commands to a 3D virtual reality system. Additionally it defines a message format which can be sent to a player or received from it. At the beginning of a new scene all objects and characters are defined by a ”definitions” script. In the course of the story all runtime dependent actions like character animations are described by so-called ”actions” scripts, whereas the temporal order is given by a special schedule block. A basic principle in this architecture is the fact that the player is assumed to be ”ignorant” without any autonomous behavior. Keeping in mind, that all rendered geometry has to be created in a pre-process, it seems to be logical that all objects including the way they act have to be registered first, before any action can be triggered. Therefore a PML script can have the following distinct root elements:

definitions define all possible/ available animations of a virtual character or an object;

actions contain the schedule of animations, ranging from key-frame to IK, and similar actions like setting visibility or starting the TTS system;

message is used for bidirectional communication;

query can be used for retrieving scene information.

The animation tags of a PML actions script refer to preloaded animations, which are referenced by their name. There are distinguished different kinds of animations like morph targets for facial animation (”singlePose”) or key-frame animation for gestures (”multiPoses”), because every animation type must be handled different and has a varying set of attributes. An example of a rather unusual animation which can be handled quite easily this way is the change of the face complexion. Usually only the changes in geometry by means of displacers or morph targets are addressed in computer graphics. This is a well known problem, and the classification usually is based on Ekman’s Facial Action Coding System (Ekman, 1982), which identifies certain Action Units for morphing. But with the help of modern graphics hardware the more subtle changes concerning face coloring can also be covered. For some emotions, changing of skin color is essential for correct perception (e.g. embarrassment and shock). It is caused by varying blood circulation and results in pallor or blushing. As the color variation can be limited to several face areas, a 3D texture map that denotes these areas can be used. Just like a stack of plates the 2D images, one for each complexion, are layered, starting with the palest face image and ending with the reddest one. Using this method not only prevents from setting invalid emotional stages but also leads to convincing results by employing texture fetching hardware for color interpolation by assigning each emotional image a number between zero and one (the intensity), which then is used for indexing into the 3D texture image stack (Jung and Knöpfle, 2006). The following actions script (which combines typical facial animations like blinking with blushing and crying) shows how such an animation can look like.

```
<actions id="cde002" start="true">
<character refId="Lebacher">
<speak id="tx">
  <text>I am so sorry about this.</text>
</speak>
<startIdleList id="idle1" refId="Blink" />
<complexion id="L1" refId="redHead" />
<complexion id="L2" refId="tears" />
</character>
<schedule>
<seq>
  <action refId="idle1" begin="0" dur="0"/>
  <par>
  <action refId="L1" begin="0" dur="4000"/>
  <action refId="L2" begin="0" dur="4000"/>
  <action refId="tx" begin="1000" dur="0"/>
  </par>
</seq>
</schedule>
</actions>
```

We used the InstantReality framework (Avalon, 2007) as the basis for our developments. Here the graphical properties and the behavior of the virtual world are defined by a scene graph following and extending the concepts of X3D (Web3DConsortium, 2007). For handling animations we added some additional nodes. Below the X3D interfaces are shown. The ”TimelineComposer” is the PML interface and can be seen as a bindable node. It handles all communication with the system and forwards PML commands to the parser. During parsing, the scheduling block is sequenced and single action and definition chunks are created and routed to the appropriate disposer components. When receiving a start message, the scheduler dispatches the action chunks to the ”AnimationController” node of the corresponding character or object. The ”TimelineComposer” triggers and holds references to all ”AnimationController” nodes, which in turn point to all ”AnimationContainer” node types; i.e. the ”InstantAnimationContainer” for referring to transitions which are state changes like toggling visibility or object reposition-

ing, and the ”TimedAnimationContainer” (including two specialized subclasses for locomotion generation) for storing all time based animations like key-frame animations and inverse kinematics. Because a complex story can lead to an arbitrary number of gestures or respectively animations, the main job of the ”AnimationController” is to blend and cross-fade all kinds of animations. This was due to the requirement, that for correct blending, cross-fading and generally updating all animations of an object at a single time step, the controlling unit needs knowledge of all animations, a task which could not be handled with the simple routing mechanisms of VRML/ X3D.

```
TimedAnimationContainer : AnimationContainer {
  SFString [] name ""
  MFNode   [] targets []
  MFString [] fieldnames []
  MFNode   [] interpolators []
  SFFloat  [] duration 0
}
AnimationController : AnimationBase {
  SFString [] name ""
  MFNode [in, out] animationContainer []
}
TimelineComposer : X3DNode {
  SFString [in,out] command ""
  SFString [out] message
  MFNode   [in,out] animationController []
}
```

3 REALISTIC RENDERING

When looking at games or technical demonstrations, one might think that most problems are solved and generalized solutions are standardized and readily available. But currently for instance the exchange schema Collada or even X3D do not support advanced interaction and rendering methods. To overcome these limitations we present various enhancements to the present X3D standard, comprising nodes for advanced rendering techniques as well as extensions for animation of characters and implementation of storylines, allowing application developers to create and author realistic VR environments easily. Though, rendering virtual characters has a lot of challenges. First, the algorithms should be easy to use and integrate into different applications. Second, to have flexible control of the character requires a flexible animation system including body movements (gestures, walking) and speech (TTS, mimics). Then, visual realism means to have realistic models, natural gestures and a realistic simulation of materials. Finally everything has to be done in real-time, because the interface must react immediately to user input.

Human skin can be classified according to spatial scale. The micro scale is defined by cellular elements whose dominant effects are scattering and absorption. For approximating subsurface scattering effects, the distance light travels through skin is measured with the same technique as is used for shadow mapping (Green, 2004). Hence we need to introduce multi pass rendering in the context of X3D, which can be understood in two ways. First, multi pass means the ability to dynamically render to an off-screen buffer. In the Xj3D extension (Xj3D, 2004) a simplified possibility for creating off-screen images was proposed with the ”RenderedTexture” node. Second, multi pass denotes the ability to render geometry in an ordered sequence, usually with different drawing operations like blending enabled, which is currently not possible in X3D. We are using an extended ”RenderedTexture” to provide the ability for off-screen rendering including associated buffers like the depth buffer. It has an additional field called ”depthMap”, which allows the automatic generation of depth maps for e.g. additional user created shadows as needed for the light pass of the skin shader (Fig. 1). Because this is only useful in combination with appropriate transformation matrices, the ”projection” (model view projection matrix of camera space) and ”viewing” fields (model matrix of parent node) are added.

Figure 1: Light pass - Blur pass - Final render pass.

A person usually has more than 100,000 hairs, which cannot be simulated in real-time. In order to reduce geometric complexity and to avoid rendering problems we model hair wisps as small quad strips. For creating an impression of thin, semi-transparent hair, textures with hair like transparent patterns are mapped onto the hair patches. To provide an impression of hair volume, alpha blending is used, which requires correct back-to-front sorting of the hair wisps. Therefore we propose the ”SortedPrimitiveSet” node (chapter 4.2). However, for simulated quad strips no unique sorting order can be determined and therefore severe sorting artifacts may result. As proposed in (Scheuermann, 2004), most artifacts can be alleviated by a multi-pass approach. Although this is only suitable for presorted hair without animation, in combination with our method for rendering human hair (Jung et al., 2005) it leads to good results. After rendering all back-facing polygons with depth writes disabled and depth test set to ”less”, the front-facing polygons

are rendered. This is accomplished by means of our ”MultiPassAppearance” node and the additional use of special render mode nodes like the ”FaceMode” and the ”DepthMode” for fine grained render state control. Multi pass rendering in X3D is thus pretty easy by using the proposed node extensions.

This shows that for complex rendering tasks control over the rendering order as well as over low level rendering modes is needed. Therefore we extended the X3D Shape component with nodes for setting different render states and therewith the ”Appearance” node with the appropriate fields. First we introduce the SFInt32 ”sortKey” field for defining the rendering order, what is essential in combination with e.g. alpha blending or depth writing. For rendering operations that belong closely together as it is the case for the two pass hair shader, we also introduce the ”MultiPassAppearance”. The ”appearance” field simply contains an ordered sequence of ”Appearance” nodes. Additionally we propose an ”AppearanceGroup” that extends the ”Group” node with an ”appearance” field. This is useful if a whole group of nodes, like head and hair, share the same material properties as is the case for the light pass, where the fragment’s distance to the light source is written and fragments with alpha values smaller a certain threshold are discarded. Besides ”FaceMode” and ”DepthMode” some more nodes for allowing finer control over rendering states are also introduced: The ”BlendMode” allows access to blending and alpha test; ”StencilMode” and ”ColorMaskMode” should likewise be self-explanatory. If the corresponding fields in the ”Appearance” node are not set, standard settings are used.

Figure 2: Left: Weeping woman (see ch. 2). Right: Sweat.

The human face communicates with various types of signals like muscular activity, but it also exhibits other signals, which are often caused by strong emotions (Kalra and Magnenat-Thalmann, 1994). Usually only the changes in geometry are addressed, but realistic skin rendering also requires displaying changes of skin color dependent on physical conditioning and emotional state - people always communicate. Some emotions are also accompanied by other phenomena, sadness often comes along with tears and sweat can be a sign of fear. Mostly sweat can be regarded as being static, but tears have to run down the face. Here we use the same animation technique as before, but now with a stack of normal and gloss maps.

4 DYNAMICS

4.1 Playback of Predefined Animations

Dynamics occur in different ways. The most obvious are the movements of the character, including locomotion and gestures. But also human hair is not static and must be simulated to achieve high visual realism. Generally spoken two types of approaches can be distinguished, the play back of predefined animation data on the one hand, and the online computation of animation data on the other hand. The X3D H-ANIM component provides support for character animation based on predefined animations (Web3DConsortium, 2006). It is based on a Skins and Bones model and defines a common skeletal model including locations and names of specific bones. Different levels of quality are defined. The animation data itself is stored in X3D interpolators; one interpolator per joint, and the data flow is defined via X3D routes. For simple scenarios, like a single animation to be played, H-ANIM works well, but it is hard to use in cases where multiple animation sets are available, which are combined and concatenated dynamically during run-time. The overall structure of such an application gets unmanageable and confusing because of the vast amount of nodes, routes and missing information about membership to specific information. Tracing and debugging is almost impossible, especially when routes are created and deleted during run-time to blend animations together in scripts. As already explained, an X3D compatible binding is proposed, which introduces several nodes that primarily act as data containers.

For efficient combination and concatenation of animations we also need additional information about the animations, e.g. data look ahead and a list of active animations and animations that will be activated within the next time-frame, which the humanoid animation component does not provide. To solve these problems we have designed animation storage nodes, which provide a consistent view on an animation set, including membership information of nodes to a specific animation. Furthermore we developed a centralized control engine for animations, overcoming the problems mentioned above, which is explained in chapter 2. For doing convincing character animation in complex and responsive environments, H-ANIM needs to be extended for incorporating blending of different animations like waving and turning around

at one single time step, which cannot be accomplished with current X3D concepts. The same goes for cross-fading different succeeding animations in order to alleviate jerky leaps between e.g. an idle motion and a subsequent gesture. By using our previously explained animation controlling extension, mixing of animations can be easily done in X3D. This is conceptually the same for facial animation based on morph targets (Alexa et al., 2000) or the H-ANIM displacer nodes. Both above mentioned types of mixing animations are realized with geometric algebra (Hildenbrand, 2005) which especially for rotations is superior to simple linear interpolation schemes. But there are still some issues, mainly due to unsuitable animation data. If the spatial distances between the first and the last animation frame are too big, this either leads to jerks or to sliding effects, depending on the blending parameters, which in the latter case introduce damping effects if too many time steps are averaged.

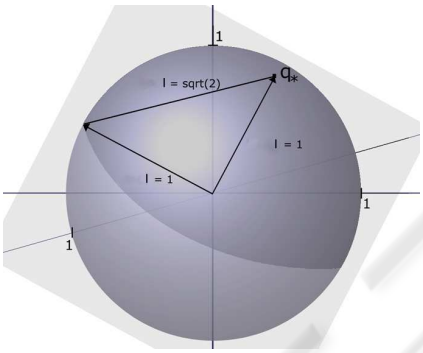

Figure 3: Unit hemisphere with reference quaternion q_*.

Rotations internally are represented as unit quaternions. Because the same rotation can be described by the quaternions q and $-q$, care must be taken when blending quaternions. In order to get a unique description of a rotation, we first define a 4d unit hemisphere, on whose surface S the unit quaternions are located. Following (Park et al., 2002), the initial choice of the hemisphere is based on an arbitrarily chosen reference quaternion (e.g. the first one, depicted as q_* in Fig. 3). Our simplified method is based on the observation that the reference quaternion q_*, any other quaternion q and the origin of the 4d unit sphere are always coplanar. With the additional constraint, that all unit quaternions are located on the same hemisphere S, the maximum angular separation between q and q_* is 90°. The maximum distance directly results from Pythagoras: $\sqrt{1^2+1^2} = \sqrt{2}$. By calculating the Euclidian distance d between q_* and any quaternion q, we can check, if q lies on the reference hemisphere by comparing d with $\sqrt{2}$. If $d > \sqrt{2}$, then q doesn't lie on S. By simply negating q the given rotation is represented by a quaternion being located on S. In the last step all quaternions are multiplied with their weights, summed together and finally normalized for obtaining the interpolation result.

4.2 Realistic Simulation of Hair

The rendering of long, light colored hair is much more complex than that of short, dark hair. Thus due to the translucent characteristics of hair fibers the additional consideration of transmission, dispersion and self shadowing is required. In case of direct lighting, there are two different specular highlights (Scheuermann, 2004). The first one results from direct reflection at the surface of a hair fiber. The second highlight results from internal reflection. The incident light passes through the interior of the fiber and is reflected at the opposite side of the cylindrical shape. Because of refraction the light's direction changes, so the secondary peak appears shifted towards the hair tip. The highlight is weaker and gets colored by the pigments of the hair. In order to calculate the different peaks described above, two tangents T' and T" are needed, which are shifted in opposite directions. This can be achieved by adding a scaled normal onto the original tangent T, given by the hair's direction, which is updated during the simulation anyway.

Figure 4: Left: Man with a beard. Right: Hair simulation.

Deformable materials like hair or cloth are often simulated by using mass spring systems. They can be calculated with the help of differential equations by equating Newton's second law of motion ($F = ma = m\ddot{s}$) and Hook's law ($F = ks$), which relates the force F exerted by a spring with spring constant k and rest length l to its deflection $s = l' - l$. But explicit numerical methods for solving these equations do not necessarily converge if forces are too strong and the size of the time step lies above a certain threshold. Our hair simulation is derived from the cantilever beam method (Anjyo et al., 1992), which originally was intended for hair modeling. Compared to mass spring approaches, it provides a numerically simpler and visually more convincing way to simulate hair. The most important difference of kinematic models compared to mass spring systems is that the initial dis-

tance l can be fully conserved. Because neighboring elements don't interact by means of spring forces, oscillations can't occur. Thus a kinematic simulation system keeps stable even with much bigger time steps. Our modified cantilever beam algorithm internally works on a kinematic multi-body chain, as illustrated in Fig. 5, left. The nodes of the multi-body chain are defined by the vertices of the original geometry. Two types are distinguished, anchors and free moving vertices. Anchors are connected to the scalp, whereas all other vertices in the chain are free moving, due to external forces like gravity, and by applying the length conservation constraint.

Besides a convincing simulation a natural behavior in case of collisions is also required. Collisions with the body are a hard constraint and must be treated explicitly. Thus for approximation we use objects like spheres, ellipsoids and planes, for which intersection tests can be handled efficiently. Hair-hair collision can't be handled easily in real-time. Thus the interpenetration of hair wisps is avoided by using a slightly different bending factor for every chain, based on the position of its respective anchor, and by arranging hair strips on top of the scalp in different layers within a different distance to the head. For keeping this up during dynamics, each vertex, depending on its position, is assigned a virtual collision sphere with a different radius, in order to parameterize the distance to the head individually (Fig. 5, right).

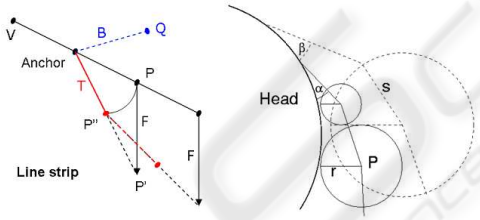

Figure 5: Simulation step; T and B resemble a quad strip.

```
SortedPrimitiveSet : X3DComposedGeometryNode {
  SFString [in,out] mode      "QuadSet"
  SFString [in,out] drawOrder "BackToFront"
  SFNode   [in,out] refPoint NULL
  SFNode   [in,out] coord    NULL
  SFNode   [in,out] color    NULL
  SFNode   [in,out] normal   NULL
  SFNode   [in,out] texCoord NULL
  SFNode   [in,out] tangent  NULL
  MFInt32 [in,out]  index    []
  SFFloat [in,out]  lowerBound  0.25
  SFVec4f [in,out]  upThreshold 0 1 0 0.85
}
```

All special simulation and rendering components are implemented as scene graph nodes in Avalon. The rendering component consists of the hair appearance and the "SortedPrimitiveSet" node. It holds all geometric properties like positions, indices and tangents and is responsible for the CPU based part of the sorting algorithm. The latter can be parameterized by the "drawOrder" field, the "upTheshold" field for defining the threshold for a second sorting step, and the "lowerBound" field for determining the percentage of quad strips which can be omitted after sorting during rendering due to occlusion. Because of its generic design our proposed "SortedPrimitiveSet" node likewise is useful for similar usages like rendering grass. It is updated via the X3D routing mechanism by the simulation component. This way the shaders only belong to the appearance nodes and are therefore interchangeable and easily to parameterize.

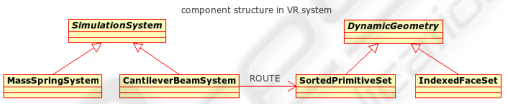

Figure 6: Component structure in VR system.

As shown in Fig. 6 we also propose some simulation system nodes. The "MassSpringSystem" is suitable for applications like cloth simulations. It inherits from our abstract "SimulationSystem" base node and is strictly separated from geometry and appearance components for maximum flexibility. The already described "CantileverBeamSystem" for simulating hair also inherits from the "SimulationSystem". With the "collisionDomainType" and "collisionDomainParam" fields collision objects can be specified, and because full triangle tests would be too expensive, the "massRadius" field can be used for setting the distance between connected vertices.

```
SimulationSystem : X3DNode {
  SFBool  [in,out]  enabled TRUE
  SFTime  [in,out]  time    0
  SFInt32 [in,out]  minStepsPerFrame -1
  SFInt32 [in,out]  maxStepsPerFrame -1
  SFTime  [in,out]  maxStepTime      -1
  SFBool  [in,out]  localCoordSystem TRUE
  MFInt32 [in,out]  index   []
  MFInt32 [in,out]  anchorIndex []
  MFVec3f [in,out]  coord   []
  MFVec3f [in,out]  normal  []
  MFString [in,out] collisionDomainType []
  MFFloat [in,out]  collisionDomainParam []
  SFVec3f [in,out]  gravity       0 -9.81 0
  SFVec3f [in,out]  externalForce 0 0 0
  SFFloat [in,out]  massRadius    1.0
  SFFloat [in,out]  staticFriction  0.5
  SFFloat [in,out]  slidingFriction 0.5
  SFFloat [in,out]  airFriction     1.0
}
CantileverBeamSystem : SimulationSystem {
  MFVec3f [in,out]  tangent []
  MFColor [in,out]  color   []
```

```
  MFVec2f [in,out]  texCoord []
  MFVec3f [in,out]  refPoint []
  SFFloat [in,out]  minBend 0.05
  SFFloat [in,out]  maxBend 0.1
}
```

4.3 Simulation of Locomotion

Capturing and processing motion data is a tedious and time consuming task. Thus a better solution would be to automatically generate them. Furthermore there are animations whose appearance is not known upfront because they depend on external parameters. Examples are pointing gestures, where the direction is calculated during run-time, e.g. pointing towards a moving object, and character locomotion, where the target is defined during runtime (”go to the red cube”). In the following we will focus on locomotion, i.e. walking. Basically there exist two types of approaches for automatic generation of walking animations (Multon et al., 1999). The first one tries to simulate the physiology of the human body using kinematics or dynamic constraints. The flexibility is very high, because theoretically any kind of human motion can be calculated, but for decent results the complexity of such simulations is very high, too. The second type adapts captured animation data according to external parameters, e.g. ”interpolation” between walking and running to attain jogging. Here the complexity is lower, because originalities of human walking are already defined in the animation sets. The drawback of these approaches is the need for motion data upfront.

The most promising approach we found was the one described by (Park et al., 2002; Park et al., 2004), which synthesis animation data from previously captured animation data according to different parameters, e.g. mood of character and style of walking. In a first step one has to preprocess the motion data and create animation sequences. Each sequence comprises of one walking cycle with fixed speed, angle and mood. To walk on a given path or towards a specified target the sequences are automatically concatenated during runtime. The values itself are interpolated according to the input values defined by the application. We integrated the algorithm into Avalon and generated the necessary motion data with a specially written exporter. The calculation of the animation data (56 joints, 27 example motions, 3 parameter dimensions) took approx. 30 ms on a Pentium 4 with 2.4 GHz, thus it was real-time capable. The visual results were very convincing; especially the concatenated walking cycles looked very lifelike.

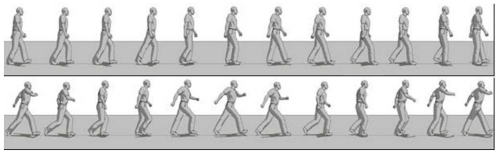

Figure 7: Different walking styles.

4.4 Content Creation

Content creation embraces the creation of the characters geometry including materials as well as animations, the scene, the behavior and the overall ”story”. For the creation of the digital story several methods can be used. Advanced ones provide non-linear storytelling and adaptive dialogues. But there are many applications where this complexity is not necessary. For such cases we developed simpler tools allowing us to put together story-lines in an easy and intuitive way. As described earlier, a story can be described with PML. It allows defining when and what a character or object in the scene is doing and how/ when the user can interact with the virtual environment. To allow interactions by the user and to change the flow of the story, we define short acts and transform them into PML scripts. A short act could be a dialogue between two characters on a given topic. Such PML scripts will be stored in special ”SceneAct” nodes which are connected by routes. The routes define the flow of the story. As soon as such a node gets a trigger on its input field, its PML script will be executed. The ”url” field holds the corresponding PML script, whereas the ”run” and ”finished” SFBool fields are used for the story graph composition. By adding other nodes in the route graph, we easily can add some non-linearity and possibilities for user interaction to the story.

5 RESULTS AND CONCLUSIONS

In this paper we described results we achieved in the Virtual Human project concerning rendering and animation of virtual characters. First extrinsic factors like properties of human skin and hair were incorporated. For modeling behavioral aspects intrinsic factors were also considered, including control languages, emotions, and animations in general. Though the rendering of scene and virtual characters was not only aimed for photorealism but also the behavior had to be modeled as realistic as possible. During the project we realized that realistic real-time rendering techniques well-known from modern computer games, such as image-based rendering as well as techniques requiring fine grained render state access were mostly ignored in the field of MR/ VR. The same goes

for the real-time simulation of deformable objects like cloth and hair. Although a lot of work already is done towards realistic rendering and simulation, research usually is conducted in a standalone application without embedding the algorithms into a wider field of applications as is needed for e.g. X3D, which is not only an open standard for interactive 3D graphics but is also easy to learn for non-programmers. Thus we proposed extensions of the standard, which were evaluated in two different scenarios.

One major goal was visual realism, where we face the same problems as research on humanoid robots: the ”uncanny valley”, a hypothesis introduced by M. Mori in 1970. He states that as a robot is made more human-like, the emotional response from a human being to the robot will become increasingly positive and empathic, until a point is reached beyond which it becomes strongly repulsive. But as appearance and motion continue to become less distinguishable from a human being’s, the emotional response becomes positive again and approaches human-human empathy levels. This also holds for virtual characters, and for convincing results we have to come very close to human-like appearance and behavior. We have developed powerful algorithms to improve rendering, even taken care of dynamics. But by simply applying them to a character model leaves us deeply stuck inside the uncanny valley without attaining convincing results, because parameter optimization is still tedious and has to be done by experts. An example is the forthcoming game from Crytek, which was postponed several times although lots of people are working on it. So, automatic generation of realistic virtual humans is not possible without human intervention. One solution could be to setup libraries a user can choose from and to have authoring tools, which guide through the creation process, starting at a very coarse level, and refining the choices step by step.

Concerning recorded motion capturing data the biggest problem was the data quality. Without heavy manual work one will face ”floating” characters or strange artifacts when blending between two very different poses. Since any kind of blending does ”interpolation” in some way there will be always cases were blending will fail and deliver unsatisfactory results. Without model knowledge or very accurate animation data we will not be able to blend animations convincingly. So, prerecorded animation must be planned accurately. It should be defined which is the starting and which is the ending pose as well as which joints are involved. Blending between very different poses therefore should be avoided. To increase flexibility research should focus on automatic real-time capable methods for the creation of animation data.

ACKNOWLEDGEMENTS

This work was part of the project Virtual Human funded by the German ministry for edu. and research.

REFERENCES

Alexa, M., Behr, J., and Müller, W. (2000). The morph node. *Web3D - VRML 2000 Proc.*, pages 29–34.

Anjyo, K.-I., Usami, Y., and Kurihara, T. (1992). A simple method for extracting the natural beauty of hair. In *SIGGRAPH ’92*, pages 111–120. ACM Press.

Avalon (2007). Avalon. http://www.instantreality.org/.

Ekman, P. (1982). *Emotion in the human face*. Cambridge University Press.

Green, S. (2004). *Real-Time Approximations to Subsurface Scattering*, pages 263–278. Add. Wes.

Hildenbrand, D. (2005). Geometric computing in computer graphics using conformal geometric algebra. In *CG 2005*, volume 29, pages 802–810.

Jung, Y. and Knöpfle, C. (2006). Dynamic aspects of real-time face-rendering. In *VRST 2006*, pages 193–196, New York. ACM: VRST Cyprus 2006.

Jung, Y., Rettig, A., Klar, O., and Lehr, T. (2005). Realistic real-time hair simulation and rendering. In *VVG 05*, pages 229–236, Aire-la-Ville. Eurographics Assoc.

Kalra, P. and Magnenat-Thalmann, N. (1994). Modeling of vascular expressions. In *Computer Animation ’94*, pages 50–58, Geneva.

Multon, F., France, L., Cani-Gascuel, M.-P., and Debunne, G. (1999). Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation*, 10(1):39–54.

Park, S., Shin, H., Kim, T., and Shin, S. (2002). Online locomotion generation based on motion blending. In *ACM Symposium on Computer Animation*.

Park, S., Shin, H., Kim, T., and Shin, S. (2004). Online motion blending for real-time locomotion generation. In *Comp. Anim. and Virt. Worlds*. John Wiley a. sons.

Piwek, P., Krenn, B., Schröder, M., Grice, M., Baumann, S., and Pirker, H. (2002). Rrl: A rich repr. lang. for the desc. of agent behaviour in neca. In *Proc. of WS ”Embodied convers. agents, let’s spec. and eval. them”*.

Scheuermann, T. (2004). Practical real-time hair rendering and shading. Siggraph 04 Sketches.

Web3DConsortium (2006). *H-Anim*. http://www.web3d.org/x3d/specifications/ISO-IEC-19774-HumanoidAnimation/.

Web3DConsortium (2007). *Extensible 3D (X3D)*. http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification_Revision1_to_Part1/.

Xj3D (2004). Xj3d dynamic texture rendering ext. http://www.xj3d.org/extensions/render_texture.html.