Gillies, Marco and Spanlang, Bernhard

Comparing and Evaluating Real Time Character Engines for Virtual Environments

Running head:

Comparing and evaluating real-time character engines for virtual environments

Marco Gillies[1], Bernhard Spanlang[2] [3]

[1] Department of Computing, Goldsmiths, University of London, London, UK, m.gillies@gold.ac.uk

[2] Departament de LSI, Universitat Politcnica de Catalunya, Spain [3] EVENT Lab, Universitat de Barcelona, Spain, bspanlang@ub.edu

Abstract

As animated characters increasingly become vital parts of virtual environments, then the engines that drive these characters increasingly become vital parts of virtual environment software. This paper gives an overview of the state of the art in character engines, and proposes a taxonomy of the features that are commonly found in them. This taxonomy can be used a tool for comparison and evaluation of different engines. To demonstrate this we use it to compare three engines. The first is Cal3D, the most commonly used open source engine. We also introduce two engines created by the authors, Piavca and HALCA. The paper ends with a brief discussion of some other popular engines.

Comparing and evaluating real-time character engines for virtual environments

Human-like characters are increasingly becoming an integral part of virtual environments. These could be members of large crowds that are added to an architectural Virtual Environment in order to give a sense that it is inhabited. On the other hand it could be a single character that is designed to have detailed social interactions with people entering the environment. In between these two extremes there are simulations involving varying numbers of characters with very diverse levels of graphical realism and complexity of behaviour. Mostly the characters will aim to provide a sense that an environment is inhabited and also a form of social or non-social interaction. Many characters will act as avatars: representations of real people that visit an environment. Avatars are generally directly controlled by their users and many of the issues relating to them are to do with providing powerful but easy to use controls, as well as the ability to customize their appearance. Characters can also be computer controlled and autonomous. They must interact with the environment, other characters, and real people without any human intervention. Control methods might be relatively simple for members of a large crowd but might have to involve very sophisticated artificial intelligence for characters that closely interact with humans. Given their importance, there is an increasing number of software engines for animating and rendering characters in virtual environments. This paper will give an overview discussion of contemporary character engines and an in depth analysis of two specific engines.

There are many requirements of a character engine. One of the most obvious is that the engine should provide an appropriate level of realism. This is a more complex issue than it at first seems. Firstly, a high level of photorealism is not appropriate to all applications. More subtly, realism of characters is far from being one dimensional. Realism of graphical appearance is important, but how the character moves and behaves is also a vital component of realism. Garau et al. (Garau et al., 2003) present results suggesting that ensuring that the levels of realism of

these different aspects match, is as important as the level of realism itself. Another requirement is that the character must be able to interact with its environment appropriately. For avatars this might mean that its user has a suitable interface for controlling its behaviour, while for agents this means having sufficient A.I. to respond to the environment. Both these types of control require that the animation of the character must be able to adapt to new situations. Finally, character engines for virtual environments must work in real time. This means that the engine must be efficient but also that it must run robustly without any human intervention other than the control of avatars.

The rest of this paper will discuss the design and implementation of character engines. The next section will give an overview of the state of the art and propose a taxonomy of features for character engines. This taxonomy can be used to analyse and compare engines. We will use it to provide a detailed analysis of three engines. The first is Cal3D a popular open source engine, that gives a good implementation of what is considered the standard functionality in a contemporary engine. The second is Piavca, an engine developed by one of the authors (Gillies) that proposes a new architecture model for some aspects of a character engine. The third is HALCA (developed by Spanlang) which extends Cal3D across a number of dimensions of our taxonomy. Finally, we will give a shorter overview of some other popular engines.

Virtual Character Engines

Virtual Character engines are generally complex pieces of software that perform a number of different functions. If we are to analyse and compare different engines we need a common structure which brings out the commonalities and differences between engines. In this paper we will analyse engines in terms of four basic functions/components that are to be found in most engines:

Appearance: the static appearance of the character, including modelling and rendering.

Rigs: the basic control structures that are used to move the character.

Animation Generators: the methods that are used to generate the movement data that animates the rigs.

Control: how the engine can be controlled by higher level processes, either by user interfaces or artificial intelligence simulations.

In many engines these are explicitly represented in the architecture as a hierarchy of layers with appearance as the lowest and control as the highest (as shown in Figure 1). For example Piavca (discussed in section) roughly decomposes into these layers. Higher levels control the activity of lower levels (possibly with some feedback). However, other engines are not explicitly structured in this way, and the different levels of our taxonomy are combined together in a single software module. For example, HALCA (section) combines rigging and appearance in its shader architecture and VHD++ (section) has an open modular architecture in which elements of functionality can be combined in arbitrary ways. Therefore the elements presented here are better thought of as classes of feature than necessarily layers of a generic engine.

figure 1 here

For each component an engine can be analysed in terms of whether the component is present at all, how powerful the component is (which advanced features it supports), how flexible it is (how easy it is to alter the functionality), and suitability for typical Virtual Environments applications.

In the rest of this section we will discuss the state of the art for each component, both in terms of features that are commonly implemented in current engines and current research that is likely to find its way into future engines. In the next section we will analyse a number of animation engines in terms of these features.

Appearance

The appearance of the characters is created with much the same computer graphics techniques as other aspects of virtual environments. Polygon meshes are the most common

underlying representation of characters. The appearance is generally enhanced with texture mapping and other more advanced mapping techniques such as displacement mapping. The increasing use of GPU based shaders allow for more complex rendering effects. Image based methods are becoming more common. While characters are generally hand modelled, scanning techniques now allow for automatic capture of surfaces and appearance properties. As this aspect is not particularly unique to character engines (with the exception of work on rendering based on the reflectance properties of human skin e.g. Donner, Weyrich, d'Eon, Ramamoorthi, and Rusinkiewicz (2008)), and has considerable overlap with general graphics, we will discuss it in less detail than the more character specific aspects.

Rigs

A rig is the commonly used term for the control structures that are used to move a character. At this level the actual movement over time is not considered, only the mechanisms that make it possible. A rig typically reduces the full complexity of a character mesh to a small number of degrees of freedom that accurately represent the movements of the human body. Types of rig are generally divided between those for body animation and those for facial animation.

Body animation is now almost entirely performed using skeletal rigs. Though some early engines used direct animation of the character mesh, as far as we are aware, skeletal rigging is current the only widely used method. A skeletal rig is an abstraction of the human skeleton consisting of a number of rigid bones linked by rotational joints, forming a hierarchy. Typically only the top of the hierarchy (the root) has a translational component. The body rig is animated by rotating the individual joints and translating the root. The fact that a skeletal rig can map, at least approximately, onto the human skeleton, and therefore the real movement structures of the body, explains the success of the method. The movement of the skeleton is normally transferred to the character mesh by a process called smooth skinning. This consists of giving a number of bones weighted influences on each vertex of the mesh. The movement of the vertex is the

weighted sum of the movements of the bones that influence it. This method is popular and can easily be implemented on a GPU making it efficient, but it can lead to unrealistic distortions. There have been recent attempts to improve on it, including better mathematical formulation of the technique such as Dual Quaternion Skinning (Kavan, Collins, Zara, & O'Sullivan, 2007) and data driven methods that make it possible to model more detailed deformations of the body, such as Pose Dependent Skinning (Allen, Curless, Popović, & Hertzmann, 2006).

Facial animation rigs are more varied than for body animation, due to the lack of natural representation like the skeleton. However, two basic methods predominate. The first is facial bones. This is the same basic technique as skeletal animation and skinning for body animation, though in this case the bones do not correspond to real structures in the body, they are merely convenient control handles for creating realistic facial movements. Whereas the joints in skeletal animation are mostly rotational, facial bones are mostly animated by translation. The other facial rigging technique, and by far the most popular is called Morph Targets or Blend Shapes (Parke, 1972). This is a holistic technique in which examples of complete facial expressions are created. These examples are blended together to produce new expressions. A final method that has been investigated in research but has yet to be used in real time engines is the physically realistic modelling of facial muscles (Terzopoulos & Waters, 1990). The movement is controlled with muscle activations. Though these methods have the potential for very accurate modelling of facial movements, the musculature of the face is extremely complex making them difficult in practice.

Animation Generators

Animation is change of the properties of a character over time. This is achieved by changing the parameters of the rig over time. This can be done by a wide variety of methods that can generate time varying rig parameters; we call these methods "animation generators". They often function as mapping from high level control input to low level rig parameters. While in some cases it can be enough to directly control low level rig parameters (for example, when using

morph targets that directly correspond to meaningful expressions), generally more complex animation generators are used.

Data driven animation. The most common source of animation in real time engines is to replay animation data that has been created off-line. This data typically takes the form of a sequence of discrete frames each containing a set of rig parameters. These frames are interpolated to obtain the final animation. The data is typically created either by hand animation or through motion capture (recording the movement of a person in a form that can be applied to an animated character). Data driven methods have the advantage of high levels of artistic control, realism and expressivity. Motion capture is more often used when the the focus is on realistic animation while hand animation is used for more expressive animation.

The main drawback of data-driven animation is inflexibility. As it consists of replaying pre-existing data, the details are difficult to change in the real time engine. This makes it difficult to adapt animation to new situations, for example, the animation of picking up an object will depend on the relative positions of the character and object. Solving this problem has been the major research topic in data-driven animation in recent years. The following are some of the most commonly studied:

Constraint based editing. This task consists in adding new constraints to a piece of motion data, or altering existing ones. Examples include altering a picking up motion given a new position of the object, or ensuring that the feet remain planted on the floor when walking over uneven terrain.

Style based editing. Altering the style of an animation or transferring the style of one animation to another. Style can mean many things, for example, the mannerisms of an individual or a particular emotional tone of an animation (e.g. a happy walk).

Transitioning and Sequencing. Playing a number of clips one after the other in a new order so as to get a new sequence of actions.

General Parameterization. Extracting useful parameters from a large data set of similar motions. For example, parameterizing a kicking motion by the target of the kick.

A variety of methods have been used to tackle these problems. Some act on single motions, for example time and space warping (Witkin & Popović, 1995). It is now more common to use large databases of motion data, and to use methods that combine different animation clips to achieve new effects. The most commonly used method is to perform a weighted interpolation on a number of methods, for example Rose, Cohen, and Bodenheimer (1998)'s work on style based editing or Kovar and Gleicher (2004)'s work on general parameterization. Other approaches include stitching together different animations on different parts of the body (Heck, Kovar, & Gleicher, 2006). Recent research has applied machine learning to a number of these problems, particularly style based editing (Brand & Hertzmann, 2000). Transitioning and sequencing is generally not problematic if the end pose of the first motion is similar to the start pose of the second (taking velocities into account). A smooth transition can be achieved by transforming the root of the second animation so that it starts at the end position of the first and then blending between the two clips over a short overlapping window. However, transitioning between very different poses is very difficult and no satisfactory general solution has been found. The normal approach is to enforce constraints that only similar poses can be transitioned. This is typically done by storing them in a graph structure in which only clips with similar start and end poses are linked. Traditionally these structures have been created manually, but Motion Graphs have provided a way of automatically creating these graph structures (Arikan & Forsyth, 2002; Kovar, Gleicher, & Pighin, 2002; J. Lee, Chai, Reitsma, Hodgins, & Pollard, 2002). Thus, there are a number of commonly used data driven techniques that can be reapplied to a number of different problems. This implies that a character engine can be structured around a few basic methods while allowing variations in their use.

Procedural animation. As we have seen, data driven animation can be inflexible, making it difficult to achieve precise effects. An alternative approach is to generate animation on the fly, algorithmically, in order to precisely match a set of requirements. This approach is called procedural animation. Much early work on animation took this approach for example the work of Zeltzer and Johnson (1991) or Perlin (1995), but this approach lost popularity due to its inability to match the realism and expressiveness of data driven animation. However, recent years have seen a partial revival for example Kopp and Wachsmuth (2004)'s work on generating gesture based on a statistical model of movement or Neff and Fiume (2006)'s work on aesthetic stance. This new generation has yet to influence mainstream real time engines, but we will discuss some methods that have remained popular over the years.

The first is inverse kinematics. The inverse kinematics problem is finding a set of joint rotations that bring an end effector (e.g. hand or foot) to a specific position in space, while keeping the limbs involved in a natural looking state. It has a wide range of applicability, from the interaction with objects to walking over varying terrain. For general skeletal structure the problem is underdetermined and there are no closed form solutions. This has lead to optimization and other iterative solution methods (Korein & Badler, 1982). While common in animation tools, the computational cost of these methods have generally made them unpopular in real time engines. However, there have been many proposed analytic solutions that are specific to human limbs (J. Lee & Shin, 1999; Tolani, Goswami, & Badler, 2000; Vinayagamoorthy, Garau, Steed, & Slater, 2004). These can give realistic results at a low computational cost. A related area is the simulation of gaze behavior, which involves pointing the head and eyes to a location. It is very challenging to capture data of gaze behavior, but simple to animate procedurally (S. P. Lee, Badler, & Badler, 2002; Vinayagamoorthy et al., 2004; Steptoe et al., 2008; Chopra-Khullar & Badler, 1999; Gillies & Dodgson, 2002).

One potentially powerful approach is to physically simulate the dynamics of human movement. For example the work of Hodgins, Wooten, Brogan, and O'Brien (1995) or Faloutsos,

Panne, and Terzopoulos (2001). Current use of physical simulation in real time engines is mostly restricted to simulating highly dynamic movement with little conscious control, e.g. the "rag-doll" movements popular in game engines.

Combining Methods. It should be clear from the above discussion that there is a great diversity of methods that can be used for character animation. Moreover there is no single best method for all applications. If the character is performing a pre-recorded speech then replaying motion capture is likely to be the best option, while walking might require interpolating and transition between motion clips and detailed object interaction would require inverse kinematics. As all of these tasks might occur in a single scenario a general engine must support all of them. In fact, different methods of animation must often be combined concurrently. For example, even when simply replaying a pre-existing piece of motion capture data, a character is likely to need gaze animation which must be generated procedurally. This means that it should be possible for an engine to combine very diverse styles of animation on a single character.

A number of hybrid methods that combine data driven and procedural animation have been suggested. One example is the work of J. Lee and Shin (1999) that applies inverse kinematics constraints to motion capture data. They use spline curves to smoothly blend the frames at which inverse kinematics is applied with the rest of the motion, an approach (Gleicher, 2001) calls "Per-frame IK + Smoothing". There have also been many techniques that use physical simulation as a way of manipulating motion data. Typically these methods are used to either alter dynamic parameters of motion or apply balance constraints when editing motion. A good example is the work of Zordan and Hodgins (2002) that provides a physical simulation of movement. During normal movement the physical simulation is completely driven by motion capture data. However, when a highly dynamic event like an impact occurs a response is automatically generated by the simulation.

Control

The requirements on an animation engine are highly dependent on the style of control, whether the control is by a person via a user interface or by an AI system. In fact many of the key research problems in animation come down to making real time animation effectively controllable while maintaining realism of animation.

As described in the introduction, characters can be divided into avatars, which are controlled by humans and agents, that are autonomous (though the distinction is not absolute as we shall see). Avatars can be controlled with simple sets of commands such as "move forward" or "turn left". However, recently with more sophisticated forms of input device, whether they are sensitive analog joysticks or full body motion tracking, the types of control can be much more diverse. Control schemes can, however, broadly be divided into two types, discrete control which consists of triggering actions or events from a finite set (e.g. "jump", "wave") and continuous control, responding in real time to a continuously changing, possibly multi-dimensional, control signal. Autonomous characters require some A.I. techniques to control them which can range from ad hoc methods to sophisticated A.I. techniques such as as natural language control (Badler et al., 2000), logic based reasoning (Funge, Tu, & Terzopoulos, 1999), planning (Aylett, Dias, & Paiva, 2006), agent based A.I. (Brom et al., 2006) and the control of crowds using path planning (Pelechano, Allbeck, & Badler, 2007). One particularly active area of research in A.I. for character control has been has been non-verbal communication and emotional expression (Vinayagamoorthy et al., 2006). To add to this complexity more sophisticated avatars can have their own autonomous behavior (Vilhjálmsson & Cassell, 1998; Gillies & Ballin, 2004), which combines the issues of both human controlled and autonomous characters.

Other considerations

Implementation Issues. The evaluation of a character engine will also include a discussion of its implementation. This will discuss the general architecture of the engine, its interfaces to other

aspects of a virtual environment, and how it is designed to be used.

The previous discussion has mostly been in terms of implementing engines for high level character animation concepts. However, most of the methods are highly generic and re-usable across different software engines. A worthwhile aim is therefore, not only to develop individual engines but also standards by which high level animation concepts can be shared between engines. In particular it is important to develop generic interchange file formats. The file format is related to the issue of content pipelines, the sequence of tools and file formats used to create the character data and import it into the real time engine. These have generally been complex, with a number of different types of data (e.g. character meshes, skeletons and motion capture) being created with separate tools and requiring a number of additional tools and operations to combine them. The situation has been improving somewhat in recent years with commercial tools such as 3DStudio Max providing an integrated environment for many of these task.

Uses. The final part of our evaluation of a character engine will be a brief discussion of some uses of the engine, demonstrating the types of application it is suited to.

Case Study: Cal3D

Cal3D is an open source character animation engine that is very commonly used in open source and virtual environment projects. It originated as part of World Forge, an open source multi-user virtual worlds project, however, it has become an independent project. It provides a full range of animation functionality and is designed to be easily integrated with other engines.

Appearance

Cal3D itself does not have a renderer so its appearance depends heavily on the graphics engine it is used with. The characters themselves are modeled as polygon meshes and texture maps are supported. While there is no built in renderer Cal3D is distributed with an example program that includes a renderer and this example renderer is commonly used as the basis of the

renderer in projects that use Cal3D. This renderer uses standard OpenGL functionality (though there is also a DirectX version) and is able to produce good quality rendering, though it lacks advanced features.

Rigs

Cal3D supports skeletal rigs and also morph targets. An engine for skeletal animation, skinning and morph targets is included, as is file format support for skeletons and bone weights. The skeletal and morph target data is held in an object called Model which is the representation of a character. Both a software implementation of skinning and support for GPU based hardware implementation is provided. Morph Targets, though present, are somewhat less well supported than skeletal animation. They are not supported in the file format or the hardware skinning model.

Animation Generators

Cal3D primarily works with stored animation data rather than procedural animation. The main mechanism for combining animation is interpolation; different animations can be blended together with user specified blend weights. This is controlled by a module called the "mixer". There are two modes in which animations maybe triggered, as an "action" or as a "cycle". Actions are played once and then end when completed. Smooth sequencing of actions is achieved by gradually increasing its blend weight from zero when it begins and gradually returning it to zero before it ends. Cycles are repeated animations that loop continuously until they are stopped. Cal3D therefore supports interpolation and (to a degree) sequencing, two of the most fundamental data driven animation techniques. However, the order in which they can be applied and the way they are combined is fixed, which may make it impossible to implement some advanced animation methods.

Control

Controlling the character in Cal3D consists in triggering actions and cycles. Any higher level control methods are left to the client program.

Implementation

Cal3D is a C++ software library that is designed to be integrated with other engines rather than be used in a stand alone manner. As shown in Figure 2, Cal3D is designed to sit on top of a renderer that is provided by the graphics engine used and to be controlled by the main Virtual Environment engine. This architecture makes it easy to combine with existing systems. Cal3D provides a file format for importing mesh, skeleton and animation data. Exporters for this format are provided for popular 3D tools such as 3DStudio Max and Blender. This provides Cal3D with an integrated content pipeline via these tools.

figure 2 here

Uses

The fact that Cal3D provides complete animation functionality while being easy to integrate with other engines makes it a popular choice for VE projects. The distribution comes with a simple demo program that shows off the functionality (see Figure 3), but this is not generally useful as a stand alone program, it is more intended as a guide to how to use the library. IMVU uses and extends the Cal3D engine for a 3D chat room. The most common use of Cal3D is to integrate it into an existing Virtual Environment or game engine. For example, it has been successfully integrated into OpenSG (Reiners, 2004) and now acts as its default character animation engine. In this case a Cal3D character has been wrapped as an OpenSG node and the character can be controlled (actions and cycles can be triggered) via OpenSG's field mechanism. The fact that the two libraries can be so closely combined shows the flexibility of Cal3D's basic architecture. Cal3D has also been used as the basis of other animation libraries that expand its

functionality, for example Piavca and HALCA, both of which are discussed later in this paper.

figure 3 here

Discussion

The ease of integrating Cal3D with other VE engines, together with its robust implementation of standard character animation functionality make it a popular choice. Its methods for combining animations provide most of the basic requirements for a typical character. However, the fixed functionality for animation generation can be somewhat limiting.

Case Study: Piavca

Piavca (the Platform Independent Architecture for Virtual Character and Avatars) is a character engine developed by one of the authors (Gillies) for use in virtual reality research. It was originally developed at University College London but is now in use at a number of other institutions. The current implementation is built on top of Cal3D (described in the previous section), it uses Cal3D's rigging functionality but provides a new, more flexible model for animation generators. Piavca is available for use under an open source license.

Appearance

The renderer used in Piavca is based on the renderer provided with the Cal3D demos, and so shares the same features. A few minor additions have been made, such as support for transparent textures.

Rigs

Piavca uses Cal3D's rigging system and so the functionality is the same. Better file format support and a hardware implementation have been added for Morph Targets.

Animation Generators

The main innovation of Piavca over and above Cal3D is a new, more modular model for animation generators. This allows more flexibility in the generators that are available and how they are combined.

Two key object oriented techniques are vital to the implementation of Piavca. If we are to combine very diverse animation methods then we must be able to handle them in the same way. This means that each method must be an implementation of a single abstract representation of animation. Designing an abstract interface that is suitable for all animation methods is therefore key to the design. This design also allows us to create an inheritance hierarchy of animation methods. This is very well suited to the situation described above where there is a great diversity of methods but many are variants of a few basic techniques that may act as intermediary base classes. The central concept of the model is a single abstract representation of all possible animation generation processes. We call this representation a *Motion*. A Motion can be thought of as a data stream, representing the changing values of joint rotations, morph targets or marker positions over time. Each Motion can control multiple items, whether these are joints, morph targets or other animation primitives. The basic operation on a Motion is to get the value of a particular item at a given time. All Motions are accessed using the same, global, continuous time representation, rather than, say, a discrete keyframe representation specific to one Motion type.

The object oriented design has a further advantage. Not only can two animation methods with the same abstract interface be combined effectively, but the combination itself can implement the same interface and therefore can be combined in new ways. The same can be done for operations on a single animation, for example scaling or time-warping. This makes it possible to easily build up complex animation systems by composing simple operations. Operations are evaluated on a frame-by-frame basis, rather than being calculated when the operator is first applied. This allows for a reasonably constant frame rate.

Motion generators that have been implemented include a data driven keyframe animation

Motion, a procedural gaze Motion and a Motion based on real time tracking data. A number of operators have been created. For example, unary operators, acting on a single Motion include:

- time warping
- looping
- a sub time range of a Motion
- transformation of representation (e.g. the Exponential Map (Grassia, 1998))
- masking off certain joints/morph targets so that they are not animated

Operators that combine more than one motion include:

- interpolating multiple Motions
- transitioning between a set of Motions (called a ChoiceMotion)
- Prioritized combining (joints are animated by high priority motion, if they are present in them, otherwise by the lower priority ones)

There are numerous subclasses of these Motion classes, for example, there are numerous types of ChoiceMotion which differ in the way in which they choose the next Motion to transition to.

These operators can be combined to produce complex behavior types. For example, a MotionGraph (Arikan & Forsyth, 2002; Kovar et al., 2002; J. Lee et al., 2002) has been implemented in Piavca (Gillies, Pan, Slater, & Shawe-Taylor, 2008) by using the sub time range operator to break up the motions and using multiple ChoiceMotions, one for each node. Two algorithms need to be provided, one for selecting, at creation time, which time ranges to use and another for choosing, at run time, which child of the choice motion to play. This framework can be used for implementing sophisticated behavior. An example described below is an implementation of Proxemics behavior (see below). Another useful property of this animation model is that interpolation in different spaces can be simply implemented by performing a transformation operator on the inputs to the interpolation Motion.

Control

Piavca supports both discrete and continuous control types. In our model discrete control is event driven. Any Motion can receive events which take the form of textual labels, and respond to them. In hierarchies of Motions events are propagated from parents to children. Events can, for example, be used to trigger particular actions (e.g. "Wave", "Jump") or to switch between different types of behavior (e.g. walking to running).

In continuous control the character will respond to a constantly varying signal. This signal may be multi-dimensional and consist of different items that vary over time (for example, the position and orientation of a head tracker). This representation of a control signal is very similar to our original representation of an animation stream. In fact, we can simply use the Motion class to represent our control signals as well as our animation. This simplifies the general architecture. For example, when dealing with interacting characters it is possible to use the Motion that animates one character as the control signal of the other.

Implementation

Piavca is implemented as a C++ library using Cal3D for the basic rigging functionality. The C++ library consists of the implementation of the Motion classes as described above as well as a class *Avatar* which represents a character and interfaces with Cal3D. A Python wrapper is also provided that provides a scripting interface as well as greater functionality. Piavca is designed to be integrated with Virtual Environment engines and it has been used with XVR (Tecchia, 2006) and OpenSG (Reiners, 2004). A number of stand alone applications have also been created using the Python interface. The overall architecture is shown in Figure 4.

figure 4 here

One of the stand alone applications is a design tool for character behavior, shown in Figure 5. It provides a graphical environment for creating and editing combinations of animation generator (Motions). An XML-based file format is also provided to save and load combinations of

Motions. The combination of the tool and file format make possible to create complex behaviors without programming, greatly speed up the process of creating a character and making it available to non-programmers. The content pipeline for Piavca is therefore the following. Motion data is created via motion capture and integrated with mesh data in 3DStudio Max. The result is then exported to the Cal3D file format. This data is then imported into the Piavca design tool where additional animation generators are added. The result of this is a Cal3D file format and an additional Piavca XML format which can then be imported into the Piavca real time engine.

figure 5 here

Uses

A Virtual Reality Experimental Framework. Piavca has been used to create software for virtual reality experiments that involve interaction with a virtual character (Pan, Gillies, & Slater, 2008). The experiments are run in a partially Wizard of Oz manner, an experimenter controls the speech of the character, but much of the non-verbal interaction is automatic. Figure 6 show the behaviors that are used.

figure 6 here

The character's speech consists of an number of "utterances", these are audio data of the speech, combined with facial and body animation. The experimenter can directly trigger these utterances, which override certain other behaviors. Once triggered the utterances are sequenced with smooth transitions.

The rest of the character's behavior is automatic. The character responds to the behavior of the experimental subject based on two sensors, a head tracker and a microphone. The head tracker is used to direct the character's gaze towards the participant but also to implement proxemic behavior. This means that the character will maintain an appropriate social distance to the participant as well as turning to face him or her. The second use of the head tracker is to detect when the participant shifts posture. This is done so that the character can synchronize its

posture shifts with those of the participant, a behavior that is believed to increase rapport (Kendon, 1970). The microphone is used to detect when the participant in speaking so that the character can display appropriate feedback behavior, in this case nodding. Figure 7 shows some still frames from an interaction with our virtual character.

figure 7 here

A real time non-verbal interaction environment. Piavca has also been used to create a large screen environment for studying multi-party gestural interaction (Healey, Frauenberger, Gillies, & Battersby, 2009). A participant interacts with two virtual characters on a large screen display (see Figure 9). The participant is tracked using a VICON motion capture system and the data is used to influence the behavior of the characters. The motion capture data is used to detect specific actions by the participant to which the characters respond. When the participant approaches the screen the character also moves towards the plane of the screen, so that they appear to be approaching for a conversation with the participant. A number of gestures are also randomly triggered. All of these actions are triggered using a Choice Motion. The characters also turn their gaze towards the participant. This is done by representing the real time motion capture data as a Motion. This Motion is then used as an input control signal for a gaze behavior. The behaviors that the character can perform are shown in Figure 8, any of which can be altered depending on the experimental context. This system significantly extends the range of possible experimental studies of non-verbal behaviors. It can be used to create real-time manipulations of a variety of aspects of non-verbal interaction ranging from large scale manipulations of body position to small-scale manipulations of the position, timing and form of gestures.

figure 8 here

figure 9 here

Discussion

Piavca provides a far more flexible architecture for motion generators than the fixed functionality of engines such as Cal3D. This, in combination with the general control scheme, makes it possible to create a wide variety of complex behavior patterns with little or no programming. This provides a major advantage over most current real time engines. However, there are some difficulties with this approach that need to be mentioned. Firstly, greater flexibility comes at a cost of greater complexity. When the set of behavior generators and ways of combining them is fixed, the possible combinations can be tested exhaustively. However, with a large and extensible number of generators and combinations there are a huge number of possible interactions, all of which could be problematic. We have found that a number of combinations of operators do give undesirable results. This can create difficulties when trying to create new behaviors. A number of idioms have been developed to work around difficulties, and these idioms are gradually being enshrined in standard motion creation scripts to make it easier for novices. One particular combination issue that has caused considerable problems in the past has been the relative timing of Motions. In many cases, most obviously when transitioning, Motions with different start times had to be combined. This resulted in each Motion having its own local time (i.e. a local 0 time relative to the global time of the environment). This resulted in a considerable overhead and complexity of having to keep track of local times and converting between local and global time when combining Motions. It was eventually decided that all Motions should use global time but keep track of their start time. This simplified matters but it is still a complex issue.

Problems also occur due to the assumption that all animation generators can share a common representation, while in fact some generators lack features that others possess. The most important example has been access to past and future times in a Motion. With stored animation data it is a simple matter to access future or past animation data, however, this is not possible with procedural animation that is generated on the fly or animation based on real time tracking data. Looking backwards or forwards is too useful a feature to completely ban, but it can only be

applied to certain Motions. No elegant solution has been found as yet. In practice, it seems that deciding which operators can be applied to which Motions is a fairly common-sensical decision and does not seem to have caused problems for users up to now.

Finally, perhaps the simplest problem to solve in the Piavca framework is the lack of animation generators. For example, there is currently no support for Inverse Kinematics. The modular nature of the architecture makes it possible to add new generators relatively easily so they can been added as and when they are needed.

Case Study: HALCA

HALCA (Hardware Accelerated Library for Character Animation) is developed in the EU project PRESENCCIA at the Universtat de Catalunya and the Universitat de Barcelona (Spanlang, 2009). It's main goal is to allow the user to animate and visualize several (up to a few hundred) realistic looking characters on single display PCs, in HMD (head mounted displays) and CAVE like systems. HALCA is being used for experimental studies in Neuroscience and Presence by partners within the EU project PRESENCCIA but has also started to be used in several other national and European projects. HALCA is closely related to Cal3D, extends Cal3D's animation and visualization functionalities and allows the user to script it from VR applications such as XVR. Figure 10 (HALCA architecture) depicts the overall architecture of HALCA which is described in more detail in the following sections.

figure 10 here

Appearance

The visualization engine of HALCA can be run in different modes. In its simplest form HALCA uses the basic OpenGL features of the Cal3D demo implementation. This is a highly portable OpenGL implementation and to our knowledge runs on any graphics card that supports OpenGL. If HALCA is switched to shader based mode then it can either load and activate GLSL shader programs from a text file or it assumes that a shader program of the OpenGL context was

activated by the hosting application. If more complex render effects such as shadows or environment maps that require multiple rendering passes are used, then HALCA can handle dynamically changing shaders. For example if shadow buffers are used the first render pass will be a view from a light source and the shader will generate an OpenGL FBO (Frame Buffer Object) of the depth information from there. In the second render pass the shader is changed to one that renders from the current camera view that can use the previously generated depth information to mix shadows with the shaded and textured model. In HALCA multiple image map files can be loaded for each material of a character in order, for example, to simulate per pixel diffuse, opacity, gloss or bump properties of a character's skin. Such maps can also be used to interactively deform the characters mesh by using them as displacement maps in the vertex shader by GPUs that support VTF (Vertex Texture Fetch) More complex global illumination effects such as those created in the Virtual Light Field Project (Mortensen et al., 2008) have also been used in combination with HALCA. This is described in more detail in the Uses section below.

Rigs

HALCA uses the Cal3D rigging system. For more visually pleasing and efficient skin deformations (that avoid the candy wrapping effect of linear blend skinning) on hardware skinned models, HALCA extends Cal3D's CPU Dual Quaternion implementation to GLSL shaders. In shader mode HALCA can deliver the skeletal joint transformations to the GPU either as transformation matrices or as dual quaternions. HALCA provides two basic GLSL shader programs that can handle linear blend skinning or dual quaternion skinning (Kavan et al., 2007) to deform the mesh of a character according to the pose of the skeleton. These basic shaders can be easily extended to create specialized effects. During the initialization, when characters are loaded, the mesh information along with morph target information is loaded into OpenGL Vertex Buffer Objects (VBO)s on the GPU. HALCA has functionality to minimize the amount of data loaded to the GPU by reusing vertex and map image data as much as possible. For example, if

the same character is added to the scene twice the VBOs and texture maps are transferred to the GPU only once. Morph targets were added to the file format and their data can be passed on to VBOs so that they can be processed by the shader. Morph targets can be combined with both Linear Blend Skinning and Dual Quaternion skinning methods.

Animation Generators

HALCA extends Cal3D's abstract mixer class to add to the basic animation functionalities of Cal3D.

HALCA adds animation functions that can:

- play and blend animations of different avatars at different speeds.
- play and blend temporal parts of an animation.
- go through an animation not by time but by an external value (see Control section). We call such animations morph animations not to be confused with morph targets.
- play or go through an animation only on specified body parts.
- directly access and manipulate joint rotations and translations on top of blend and morph animations.
- efficiently access and manipulate the whole skeletal state. (One function call is required to accessed the skeleton or to set it to a particular pose by passing the skeletal state vector of a character to or from HALCA).

Owing to its simple access to every joint state of the character several Inverse Kinematics algorithms have been added to HALCA in the XVR scripting language.

HALCA gives access to properties of an animation such as the duration, frame rate, etc. to the hosting application. Such information can be useful to compute for example the actual walking speed of a character when animated.

Breaking up animations into motion graphs is something that is not done in HALCA because such a process may require manual intervention and can be done in an application

dedicated to this purpose off line.

As with all animation mixers not every motion blended with another creates natural looking movements. Therefore for mixing tasks where the outcome is not clear it is preferable to use the animation facilities of the preferred animation modeling application such as 3D Studio Max, Motionbuilder, Blender or Maya.

Control

HALCA provides three types of control mechanisms.

As in Cal3D events can be sent to HALCA to blend or play one or more motions at a time. In addition, as mentioned in the previous section, an animation can be controlled by an external value as opposed to playing through time. This functionality has turned out to be very useful for controlling the motions of an avatar, for example, by data of a human physiology measuring device. For example an animation of the movements of the spine can be linked to respiration data to animate a virtual character to mimic the breathing of a user.

If 3D tracking data is used then direction control can be used, for example, for head rotations, tracking data can be directly mapped by manipulating the neck joint rotation of an avatar. If the number of degrees of freedom tracked is high, for example from a Motion Capture system, then the functions to change the whole body state can be used for efficient control. Integration with a data glove where the data glove sensor output was mapped directly to finger joint rotations is another example of direct control. For arm or upper body movements controlled by trackers the Inverse Kinematics implementations that use direct joint rotation manipulation have been useful.

To control facial expressions based on morph targets HALCA provides functionality to dynamically modify the weights of each target. However, if lip synchronization is required HALCA assumes the animation to be made available from a modeling application such as 3DS Max or from real time input of a specialized program.

HALCA computes bounding volumes of the skeletal segments of an avatar in order to make an integration with existing physics engines such as PhysX, Havok, ODE, etc. possible. Positional and rotational changes of the bounding volumes can be exchanged between the physics engine and the animation system to make a combination of data driven animation and physical simulation possible.

HALCA provides functionality, via the shader system, to also control other properties of the character, for example to change the color of the skin, to dynamically create wrinkles, to make areas of the body invisible dynamically or to expand areas of the body.

Implementation

HALCA is implemented in C++ with heavy use of STL and shaders written in GLSL (Rost, 2006). HALCA is provided as a windows DLL that can be loaded into any hosting application that supports OpenGL 2.0 or higher. Since it is independent of the windows operating system it can also be compiled for other operating systems. It has been mainly used in XVR but also stand alone applications based on GLUT have been developed.

The HALCA content pipeline integrates Motion Capture data with the Cal3D pipeline. Motion Capture is processed and integrated with the character data using the MotionBuilder tool, and the resulting data is then exported to the Cal3D format using 3DStudio Max. Further behavior can be scripted in the XVR scripting language, resulting in run time behavior.

Uses

Fire Experiment. The fire experiment which was designed for the PRESENCCIA project was the first use of HALCA (Spanlang, Fröhlich, Fernandez, Antley, & Slater, 2007). For this experiment motion capture data was acquired in the Vicon system at UCL and animations were edited in Character Studio (part of 3D Studio Max). Pre designed Animation Clips were merged by scripting in XVR and HALCA. The goal of this experiment was to identify whether interactively controlled characters can influence how much a human participant gets scared by a

virtual fire.

Virtual Light Field. The flexibility of HALCA that makes it possible to render the scene with different shaders was used for the light field approach of global illumination (Mortensen et al., 2008) in order to interactively visualize virtual characters while taking global illumination effects such as shadows and mirror reflections into account. For this setup the CAVE like system at UCL was used and the hosting VR engine was XVR. Avatars were controlled interactively by the Inverse Kinematics functions built for HALCA which were connected to real time tracking input in order to move the body, head and the arms to mimic the movements of the user. The avatar was not rendered directly in the 3D environment, since in a CAVE system the user can see his/her own body, but was visible through mirror reflections and shadows in the virtual environment.

Body Ownership Experiments. At the time of writing several body ownership experiments are being carried out, the goal of which is to identify how much a human participant can be made to believe that the whole or parts of an avatar's body is his body (M.Slater & Sanchez-Vives., 2008). For these experiments an integration with the optical motion capture system Optitrack was developed to control the whole skeleton of a character in real time with HALCA (Spanlang & Slater, 2009). In addition several Inverse Kinematics functions were created to control the upper body of a character with just 2 or 3 trackers or with respiration physiology measurements. The integration of HALCA with a data glove is another example that was used for a body ownership experiment (Slater, Spanlang, Frisoli, & Sanchez-Vives, 2008)

Crowds. Owing to HALCA's efficient and flexible rendering capabilities it has been combined with the HiDAC crowd simulation module described by Pelechano et al (Pelechano, Allbeck, & Badler, 2008; Pelechano, Spanlang, & Beacco, 2009). In this integration the main functions used are motion blending and direct root, upper body and leg joint manipulation to control the walking behavior of the crowd members as well as their orientation, gaze and location

while removing foot sliding artifacts.

Our current tests have shown that it is possible to visualize more than 200 characters that are constructed with about 5000 polygons by using HALCA in GPU mode on a laptop with state of the art GPU.

Discussion

HALCA displays a different approach to extending the functionality of Cal3D. Whereas Piavca's extensions were narrowly focused, largely on the animation generation level, HALCA provides a wide range of extensions across all the levels. The shader architecture allows for great flexibility at the appearance level and for a range of advanced visual effects which are not possible with the other engines described in this paper. It also allows extensions at the rigging level (dual quaternion skinning) and for greater efficiency (allowing for large scale crowd simulation). The extensions at the animation generator and particularly at the control level have a strong focus on using real time data from trackers and other sources (e.g. physiological data). This makes the engine particularly useful for immersive virtual reality applications.

HALCA's functionality was successively extended for the new requirements that were given in the VR experiments in which it was used. Therefore the main architectural drive was to add as little to the existing functionality in order to fulfill the new requirements. For example in HALCA there is no attempt to abstract real time tracking data to an internal motion representation but instead direct joint manipulation is used. This strategy may not be as pretty from an architectural point of view but it has shown to be useful to get animation tasks done efficiently in VR applications. As opposed to Piavca (for example), HALCA builds on the existing animation framework of Cal3D. It is arguable whether HALCA's *ad hoc* strategy is of advantage in the long run or if it is better to adapt real time motion merging problems to a more general animation framework.

In HALCA it would be useful to be able to exploit the great flexibility that is gained by

allowing the user to dynamically modify the shader program that renders a character. This also brings flexibilities that could benefit if they were controllable by the animation framework. For example changing the skin color of an avatar dynamically by using a shader could be done more intuitively if it were possible to use an animation controller rather than having to directly modify a shader variable by scripting. Such tighter integration between the flexibility of shaders and animation control is work in progress.

Other Engines

This section will give a short analysis of some other character animation engines.

Built in animation systems.

Many VE and game engines have their own character engines built in. These generally, have animation functionality that is similar to, or more restricted than Cal3D. Most support skeletal animation, though Morph Targets are rarer, and have some support for combining animations. The appearance of the character is determined by the graphics engine and so is generally more varied than the animation functionality. The control of the character will also depend a lot on the rest of the engine. An example is the Horde3D graphics engine which includes skeletal animation, morph targets, and blending of animation data or the Ogre engine which includes skeletal animation and playing of animation data. Many open source engines VE engines use Cal3D as their character engine (including OpenSG as described above).

The most widely used engine in commercial game production is Unreal. As might be expected from a high end commercial engine successive versions of the engine track the state of the art in real time graphics and so are able to maintain very high quality appearance. An animation engine is included which supports both skeletal and morph target rigs. It also includes a number of animation generators. These include data driven animation, physics based animation and Inverse Kinematics as well as methods of combining animations such as blends and filters (only playing selected joints of an animation). These generators are combined using a data

structure called an *AnimTree*, each generator is a node and these nodes can be combined in a tree to form complex generators. Control of characters is either through user interaction for avatars or via a built in A.I. system. Character A.I. is defined using a scripting language which includes a number of A.I. methods including Finite State Machines and path planning.

Jack

Jack was one of the first fully featured character engines (Badler, Philips, & Webber, 1993), and is still used (now a commercial engine), particularly for simulation of the ergonomic properties of architectural and industrial designs. It contains a number of procedural animation generators and control methods. While the main focus is not on character appearance it is possible to extensively change the dimensions of the character in order to perform ergonomic tests with different sized individuals.

VHD++

The preceding discussion has show the variety of possible technologies that can be used at all levels of our taxonomy and how different technologies can be relevant to different applications. Modular engines can be very useful in this context as they allow tailoring of functionality to a particular application while making it possible to reuse common functionality. Many of the engines described up to now provide a degree of modularity, for example Piavca's Motions can be viewed as modular animation generators while HALCA's shader system provides modularity at the level of appearance and rigs. VHD++ (Ponder, Papagiannakis, Molet, Magnenat-Thalmann, & Thalmann, 2003) takes this tendency far further than any of the other engines discussed. In fact, it is less an engine than a modular framework for creating tailored engines. It consists of a generic kernel which manages modules and their communication and then a large number of modules that implement a wide range of virtual reality and character functionality. As these the kernel makes it simple to combine modules and have them communicate, the process creating new applications with diverse functionality is much quicker.

Modules have been implemented that supply functionality at all levels of our taxonomy. These include:

appearance: a multi-textured rendering system that allows for stereo viewing.

rigs: skeletal and morph target animation as well as cloth and hair simulation.

animation generators: keyframe animation, procedural animation (including Inverse Kinematics, gaze and a walk generator) and real time data tracking.

control: scripting, graphical interfaces and motion tracking for real time control and A.I. methods for path planning and controlling crowds etc.

VHD++ has been used extensively on a number of research projects and applications. Examples include JUST, a training simulation for health emergency personnel which allows trainees to interact in immersive virtual reality with movement tracking. CAHRISMA involves virtual reality reconstruction of cultural heritage which involves a variety of diverse crowd behaviours. VHD++ is available as open source software.

Commercial character engines

Havok animation is a character animation library that is shipped with the Havok game physics engine. It supports a sophisticated set of tools at the rig and animation generator level. At the rig level it supports skeletal rigs and morph targets. It supports data driven animation including transitioning, interpolation, addition and mirroring of animations. It also supports a number of different Inverse Kinematics methods, from a general CCD method to a specific solver for footsteps. Finally, being part of the Havok physics engine it has extensive support for physics based animation and integration of data driven and physics based animation. At the animation generator level it therefore exceeds most of the engines we have seen. While Piavca has more flexible support for combining and transforming animations, Havok animation provides a wider range of types of animation generator, such as inverse kinematics and physics based animation.

Havok animation is designed to be integrated with a game engine and so all appearance and control aspects will be handled by the main engine.

Natural Motion provides a number of tools and engines for virtual characters in computer games and visual effects. Their engines only provide animation generation functionality, with control, appearance and skinning being provided by client engines. Their Morpheme engine includes powerful techniques for data driven animation including state machine based transitioning, multi-way interpolation and Inverse Kinematics. While morpheme has fewer inverse kinematics methods than Havok physics they generally have a wider range of data driven methods than any of the other engines that are considered here. Natural Motions Euphoria and Endorphin systems add a wide range of procedural animations including extensive physical simulation. They also have numerous methods for combining physically simulated and data driven animation.

Both Havok Animation and the Natural Motion systems are expensive commercial systems and they are mostly used for high end computer games.

SAIBA

A final project that should be mentioned is SAIBA (Kopp et al., 2005). This is not an engine, rather it is an attempt to standardize interfaces between A.I. and animation systems, with a particular focus on communicative non-verbal behavior. Two interface languages have been defined. One, called the Behavior Markup Language (BML), roughly corresponds to the interface between the control and animation generator levels presented here, while the Function Markup Language (FML) interfaces higher and lower level AI processes.

Engines are beginning to emerge that are implementing the SAIBA framework. One example is Greta (Bevacqua, Mancini, Niewiadomski, & Pelachaud, 2007), a character engine with special focus on expressive non-verbal behavior. Unlike most of the other engines we have discussed Greta has a stronger emphasis on facial animation than body animation as reflected by a sophisticated facial rig. The animation generator and control levels include a number of

specialized features for communicative and conversational non-verbal communication. These include methods for planning and executing complex facial expressions, which are based on psychological studies of expressive behavior. Other engines that have fed into the development of SAIBA include the Rea character (Cassell et al., 1999), Spark (Vilhjálmsson, 2005) and Max (Kopp, Jung, Lessmann, & Wachsmuth, 2003), all of which have a particular focus on the integration of conversation and gesture.

figure 11 here

Discussion and Evaluation

This paper has presented a comparison of a number of character engines for virtual environments. A summary is shown in Figure 11. A taxonomy of the different features of an engine was presented and used in the comparison. This taxonomy has helped to elucidate some of the similarities and difference between engines. It is clear that while some engines differ only in the presence or absence of individual features, some tackle substantially different aspects of the problem. It also helps explain how certain engines fit together. For example, both Piavca and HALCA extend Cal3D but they do so in very different ways. Piavca directly replaces the animation generator functionality of Cal3D with a completely new module which adds greater flexibility, but leaves the other levels largely unchanged. On the other hand HALCA focuses primarily in adding functionality at the levels in which Cal3D is lacking: appearance and control.

We hope that this analysis and taxonomy can help the future development of the engines discussed, and of future engines, by identifying areas in which the current state of the art is lacking.

Acknowledgements

We would like to thank the funders of this work, including the EU Future and Emerging Technologies Integrated Projects PRESENCIA and PRESENCCIA, the EPSRC project

Empathic Avatars and BT plc. We would also like to thank the members of the UCL Department of Computer Science Virtual Environment and Computer Graphics group for their help and support, and also the following people who have helped with the development of PIAVCA and HALCA: Mel Slater, Anthony Steed, David Swapp, Insu Yu,Pankaj Khanna, Adelaida Papadianaki, Angus Antley, Xueni Pan, William Steptoe, Christoph Groenegress, Joan Llobera, Nuria Pelechano, Ausias Pommes, Alejandro Beacco Porres, Daniel Marcos Perez, Elias Giannopoulos, Robert Leeb, Stuart Battersby, Christophe Frauenberger and Patrick Healey

References

Allen, B., Curless, B., Popović, Z., & Hertzmann, A. (2006). Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis. In *SCA '06: Proceedings of the 2006 acm siggraph/eurographics symposium on computer animation* (pp. 147–156). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Arikan, O., & Forsyth, D. A. (2002, July). Interactive motion generation from examples. *ACM Transactions on Graphics*, *21*(3), 483–490.

Aylett, R., Dias, J., & Paiva, A. (2006). An affectively-driven planner for synthetic characters. In *ICAPS 2006.* AAAI Press.

Badler, N., Bindiganavale, R., Allbeck, J., Schuler, W., Zhao, L., & Palmer, M. (2000). Parameterized action representation for virtual human agents. In J. Cassell, J. Sullivan, S. Prevost, & E. Churchill (Eds.), *Embodied conversational agents* (p. 256-284). MIT Press.

Badler, N., Philips, C., & Webber, B. (Eds.). (1993). *Simulating humans: Computer graphics, animation and control*. Oxford University Press.

Bevacqua, E., Mancini, M., Niewiadomski, R., & Pelachaud, C. (2007). An expressive ECA showing complex emotions. In *Aisb workshop on language, speech and gesture for expressive characters.*

Brand, M., & Hertzmann, A. (2000, July). Style machines. In *Proceedings of ACM SIGGRAPH 2000* (pp. 183–192).

Brom, C., Gemrot, J., Bída, M., Burkert, O., Partington, S., & Bryson, J. (2006, November). POSH tools for game agent development by students and non-programmers. In *9th international conference on computer games: AI, animation, mobile, educational & serious games.*

Cassell, J., Bickmore, T., Campbell, L., Chang, K., Vilhjálmsson, H., & Yan, H. (1999). Embodiment in conversational interfaces: Rea. In *ACM SIGCHI* (p. 520-527). ACM Press.

Chopra-Khullar, S., & Badler, N. (1999). Where to look? automating visual attending behaviors

of virtual human characters. In *Autonomous agents conference.*

Donner, C., Weyrich, T., d'Eon, E., Ramamoorthi, R., & Rusinkiewicz, S. (2008). A layered, heterogeneous reflectance model for acquiring and rendering human skin. *ACM Trans. Graph.*, *27*(5), 140.

Faloutsos, P., Panne, M. van de, & Terzopoulos, D. (2001). Composable controllers for physics-based character animation. In *Siggraph '01: Proceedings of the 28th annual conference on computer graphics and interactive techniques* (pp. 251–260). New York, NY, USA: ACM.

Funge, J., Tu, X., & Terzopoulos, D. (1999, August). Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proceedings of SIGGRAPH 99*, 29–38.

Garau, M., Slater, M., Vinayagamoorhty, V., Brogni, A., Steed, A., & Sasse, M. (2003). The impact of avatar realism and eye gaze control on perceived quality of communication in a shared immersive virtual environment. In *Proceedings of the acm sig-chi conference on human factors in computing systems.*

Gillies, M., & Ballin, D. (2004, July). Integrating autonomous behavior and user control for believable agents. In *Third international joint conference on autonomous agents and multi-agent systems.* Columbia University, New York City.

Gillies, M., & Dodgson, N. (2002). Eye movements and attention for behavioural animation. *Journal of Visualization and Computer Animation*, *13*, 287-300.

Gillies, M., Pan, X., Slater, M., & Shawe-Taylor, J. (2008). Responsive listening behavior. *Computer Animation and Virtual Worlds*, *19*, 1-11.

Gleicher, M. (2001). Comparing constraint-based motion editing methods. *Graphical Models*(63), 107-134.

Grassia, F. S. (1998). Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, *3*(3), 29–48.

Healey, P. G. T., Frauenberger, C., Gillies, M., & Battersby, S. (2009). Experimenting with

non-verbal interaction. In S. Kopp & I. Wachsmuth (Eds.), *8th international gesture workshop.*

Heck, R., Kovar, L., & Gleicher, M. (2006, September). Splicing upper-body actions with locomotion. *Computer Graphics Forum*, *25*(3), 459–466.

Hodgins, J. K., Wooten, W. L., Brogan, D. C., & O'Brien, J. F. (1995). Animating human athletics. In *Siggraph '95: Proceedings of the 22nd annual conference on computer graphics and interactive techniques* (pp. 71–78). New York, NY, USA: ACM.

Kavan, L., Collins, S., Zara, J., & O'Sullivan, C. (2007, April/May). Skinning with dual quaternions. In *2007 ACM SIGGRAPH symposium on interactive 3d graphics and games* (pp. 39–46). ACM Press.

Kendon, A. (1970). Movement coordination in social interaction. *Acta Psychologica*, *32*, 1-25.

Kopp, S., Jung, B., Lessmann, N., & Wachsmuth, I. (2003). Max–a multimodal assistant in virtual reality construction. *KI-Knstliche Intelligenz*, *3*(4), 11–17.

Kopp, S., Krenn, B., Marsella, S., Marshall, A., Pelachaud, C., Pirker, H., et al. (2005, September). Towards a common framework for multimodal generation in ECAs: The behavior markup language. In *Proceedings of the 6th international working conference on intelligent virtual agents* (p. 205-217). Springer.

Kopp, S., & Wachsmuth, I. (2004). Synthesizing multimodal utterances for conversational agents. *The Journal Computer Animation and Virtual Worlds*, *15*(1), 39–52.

Korein, J. U., & Badler, N. I. (1982). Techniques for generating the goal-directed motion of articulated structures. *IEEE Computer Graphics and Applications*, 71-81.

Kovar, L., & Gleicher, M. (2004, August). Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics*, *23*(3), 559–568.

Kovar, L., Gleicher, M., & Pighin, F. (2002, July). Motion graphs. *ACM Transactions on Graphics*, *21*(3), 473–482.

Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K., & Pollard, N. S. (2002, July). Interactive

control of avatars animated with human motion data. *ACM Transactions on Graphics*, *21*(3), 491–500.

Lee, J., & Shin, S. Y. (1999, August). A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of siggraph 99* (pp. 39–48).

Lee, S. P., Badler, J. B., & Badler, N. I. (2002, July). Eyes alive. *ACM Transactions on Graphics*, *21*(3), 637–644.

Mortensen, J., Yu, I., Khanna, P., Tecchia, F., Spanlang, B., Marino, G., et al. (2008). Real-time global illumination for VR applications. *IEEE Computer Graphics and Applications*, *28*, 56-64.

M.Slater, H. E., D. Perez-Marcos, & Sanchez-Vives., M. (2008). Towards a digital body: The virtual arm illusion. *Front. Hum. Neurosci.*, *2*(6), 110-117.

Neff, M., & Fiume, E. (2006). Methods for exploring expressive stance. *Graph. Models*, *68*(2), 133–157.

Pan, X., Gillies, M., & Slater, M. (2008). Male bodily responses during an interaction with a virtual woman. In H. Prendinger, J. C. Lester, & M. Ishizuka (Eds.), *Intelligent virtual agents, 8th international conference, iva 2008, tokyo, japan, september 1-3, 2008. proceedings* (Vol. 5208, p. 89-96). Springer.

Parke, F. I. (1972). Computer generated animation of faces. In *Acm'72: Proceedings of the acm annual conference* (pp. 451–457). New York, NY, USA: ACM.

Pelechano, N., Allbeck, J., & Badler, N. (2007). Controlling individual agents in high-density crowd simulation. In *ACM SIGGRAPH / eurographics symposium on computer animation.*

Pelechano, N., Allbeck, J., & Badler, N. (2008). *Virtual crowds methods, simulation, and control.* Morgan and Claypool Publishers.

Pelechano, N., Spanlang, B., & Beacco, A. (2009). A framework for rendering, simulation and animation of crowds. In *Spanish conference on computer graphics.*

Perlin, K. (1995, March). Real time responsive animation with personality. *IEEE transactions on*

visualization and Computer Graphics, *1*(1), 5-15.

Ponder, M., Papagiannakis, G., Molet, T., Magnenat-Thalmann, N., & Thalmann, D. (2003, July). VHD++ development framework: Towards extendible, component based vr/ar simulation engine featuring advanced virtual character technologies. In *Proc. of computer graphics international* (p. 96-104). IEEE Publisher.

Reiners, D. (2004). Special issue on the opensg symposium and opensg plus. *Computers & Graphics*, *28*(1), 59-61.

Rose, C., Cohen, M. F., & Bodenheimer, B. (1998, September - October). Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications*, *18*(5), 32–40.

Rost, R. (2006). *OpenGL® shading language (2nd edition)*. Addison-Wesley Professional.

Slater, M., Spanlang, B., Frisoli, A., & Sanchez-Vives, M. V. (2008, Sept). *Virtual hand illusion induced by visual-propioceptive and motor correlations.*

Spanlang, B. (2009). *Halca a library for presence research* (Tech. Rep.). event lab, Universitat de Barcelona, www.event-lab.org.

Spanlang, B., Fröhlich, T., Fernandez, V. D., Antley, A., & Slater, M. (2007). The making of a presence experiment: Responses to virtual fire. In *Annual international workshop on presence* (p. 303-307).

Spanlang, B., & Slater, M. (2009). Full body avatar interaction. In *Congreso internacional de interaccin persona ordenador AIPO.*

Steptoe, W., Wolff, R., Murgia, A., Guimaraes, E., Rae, J., Sharkey, P., et al. (2008). Eye-tracking for avatar eye-gaze and interactional analysis in immersive collaborative virtual environments. In *CSCW '08: Proceedings of the ACM 2008 conference on computer supported cooperative work* (pp. 197–200). New York, NY, USA: ACM.

Tecchia, F. (2006). Building a complete virtual reality application. In M. Slater, Y. Kitamura, A. Tal, A. Amditis, & Y. Chrysanthou (Eds.), *Vrst* (p. 383). ACM.

Terzopoulos, D., & Waters, K. (1990, August). Physically-based facial modelling, analysis, and animation. *Journal of Visualization and Computer Animation*, *1*(2), 73–80.

Tolani, D., Goswami, A., & Badler, N. I. (2000). Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, *62*(5), 353-388.

Vilhjálmsson, H. H. (2005). Augmenting online conversation through automated discourse tagging. In *6th annual minitrack on persistent conversation at the 38th hawaii international conference on system sciences.*

Vilhjálmsson, H. H., & Cassell, J. (1998). Bodychat: Autonomous communicative behaviors in avatars. In *second acm international conference on autonomous agents.*

Vinayagamoorthy, V., Garau, M., Steed, A., & Slater, M. (2004, March). An eye gaze model for dyadic interaction in an immersive virtual environment: Practice and experience. *Computer Graphics Forum*, *23*(1), 1–12.

Vinayagamoorthy, V., Gillies, M., Steed, A., Tanguy, E., Pan, X., Loscos, C., et al. (2006). Building expression into virtual characters. In *Eurographics conference state of the art reports.*

Witkin, A., & Popović, Z. (1995). Motion warping. In *ACM SIGGRAPH* (p. 105-108).

Zeltzer, D., & Johnson, M. B. (1991). Motor planning: An archetecture for specifying and controlling the behavior of virtual actors. *The Journal of Visualization and Computer Animation*, *2*, 74-80.

Zordan, V. B., & Hodgins, J. K. (2002). Motion capture-driven simulations that hit and react. In *Sca '02: Proceedings of the 2002 ACM SIGGRAPH/eurographics symposium on computer animation* (pp. 89–96). New York, NY, USA: ACM.

Figure Captions

Figure 1. Different layers in a character engine.

Figure 2. The basic architecture of Cal3D.

Figure 3. The characters included in the Cal3D demo program.

Figure 4. The basic architecture of Piavca.

Figure 5. Piavca's graphical design tool.

Figure 6. The behaviors used in the virtual reality experimental framework.

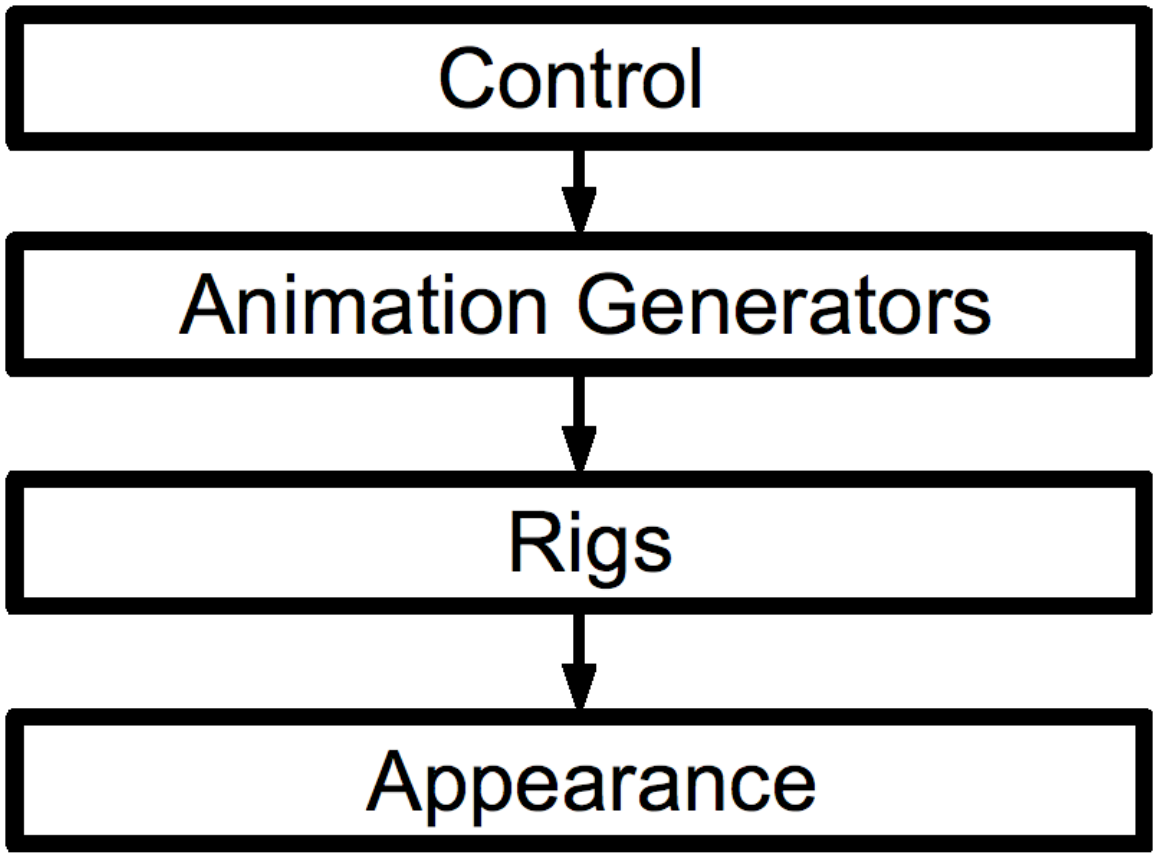
Figure 7. A real and virtual human interacting in an immersive virtual environment.

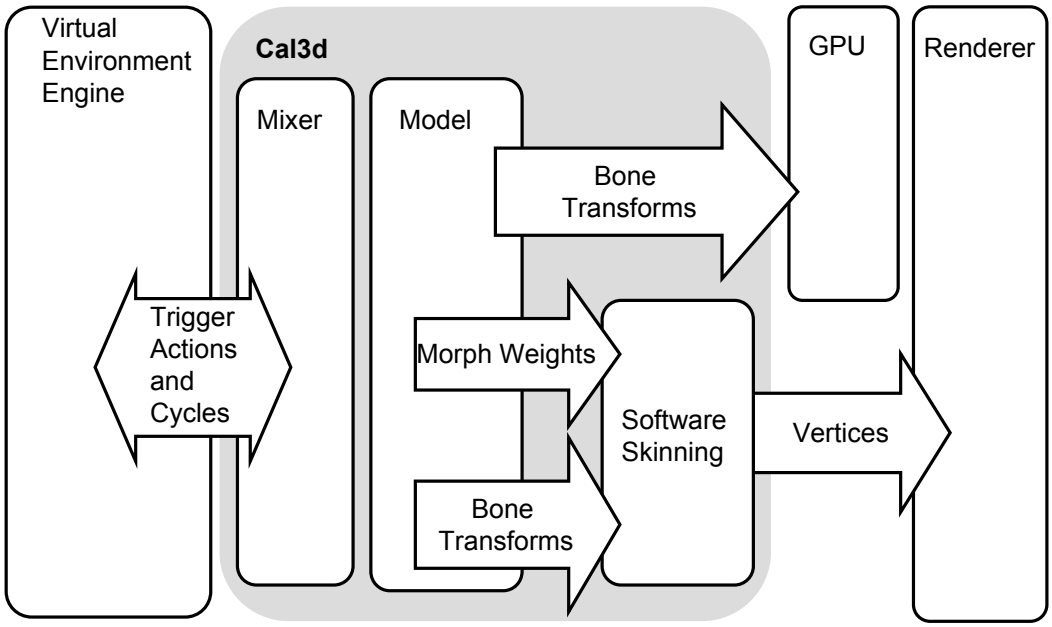
Figure 8. The behaviors used in the non-verbal interaction environment.

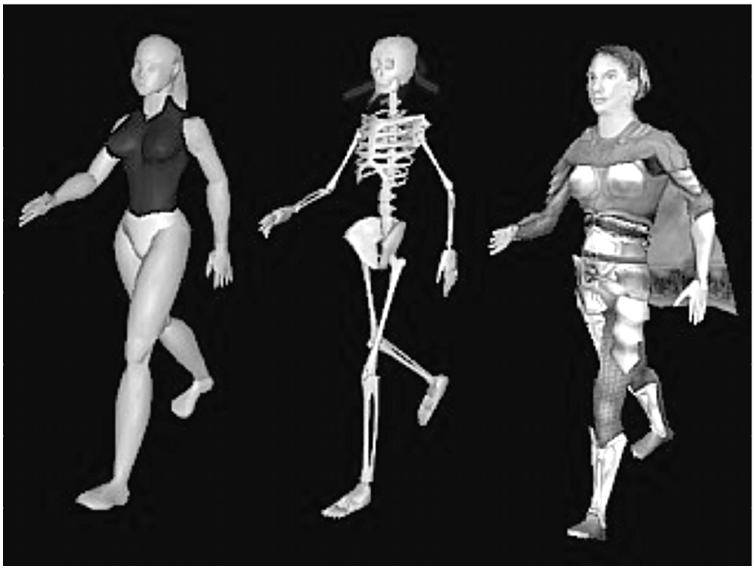
Figure 9. A Human Participant Interacting with Two Avatars.

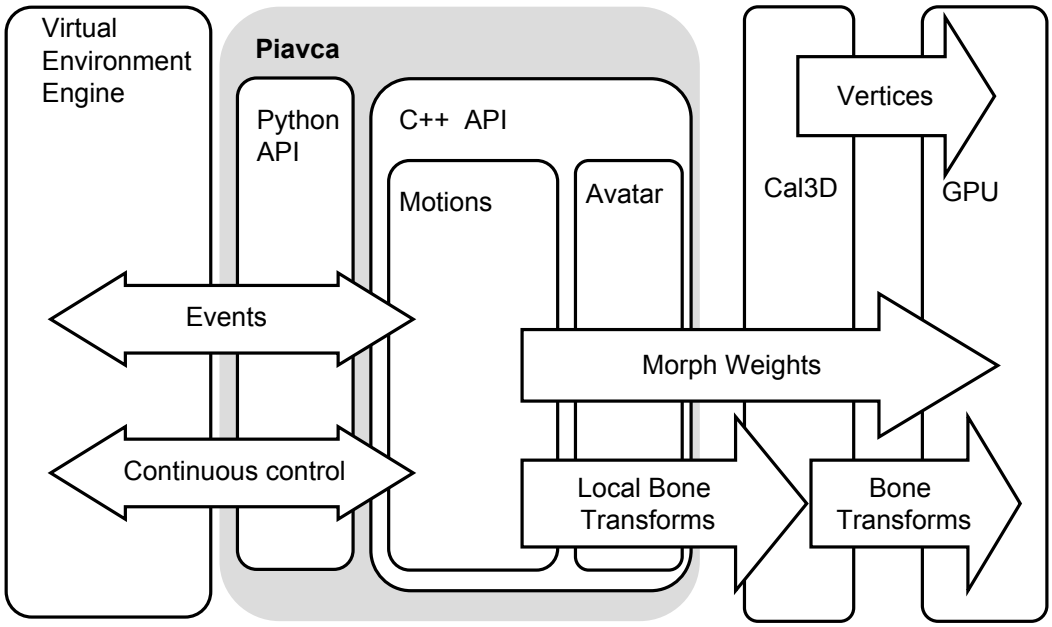
Figure 10. The HALCA architecture.

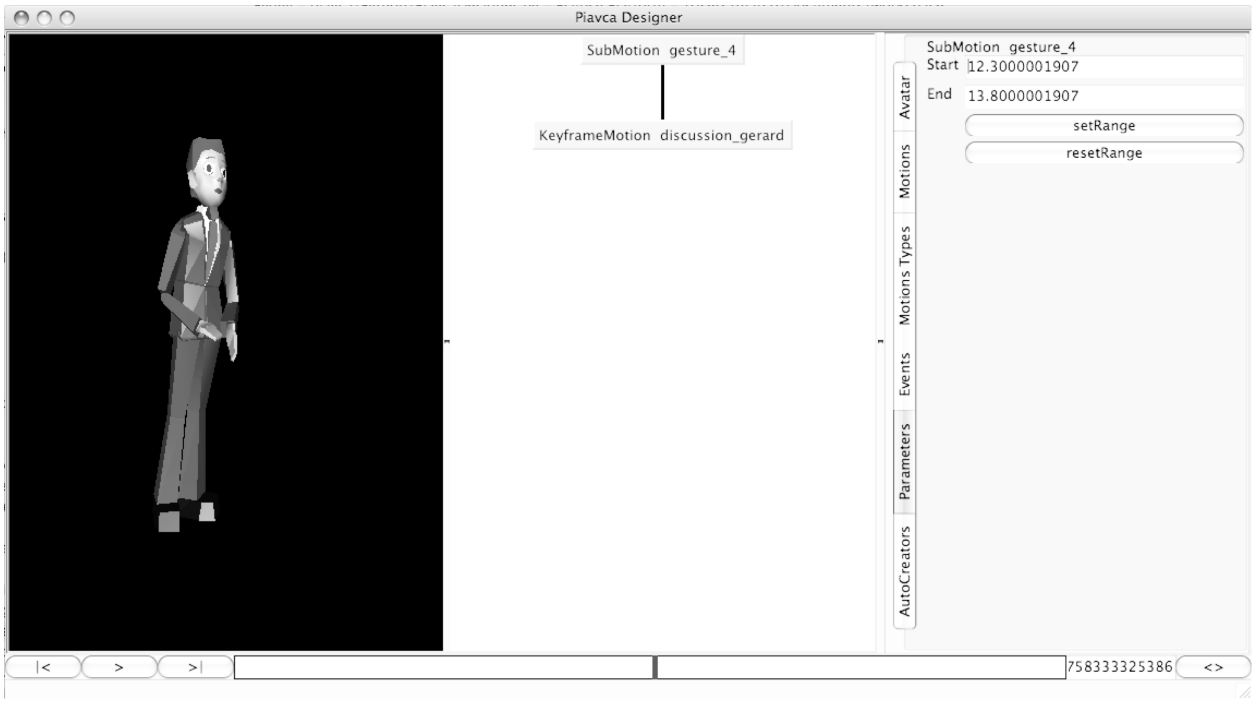
Figure 11. A summary of the comparisons made in this paper.

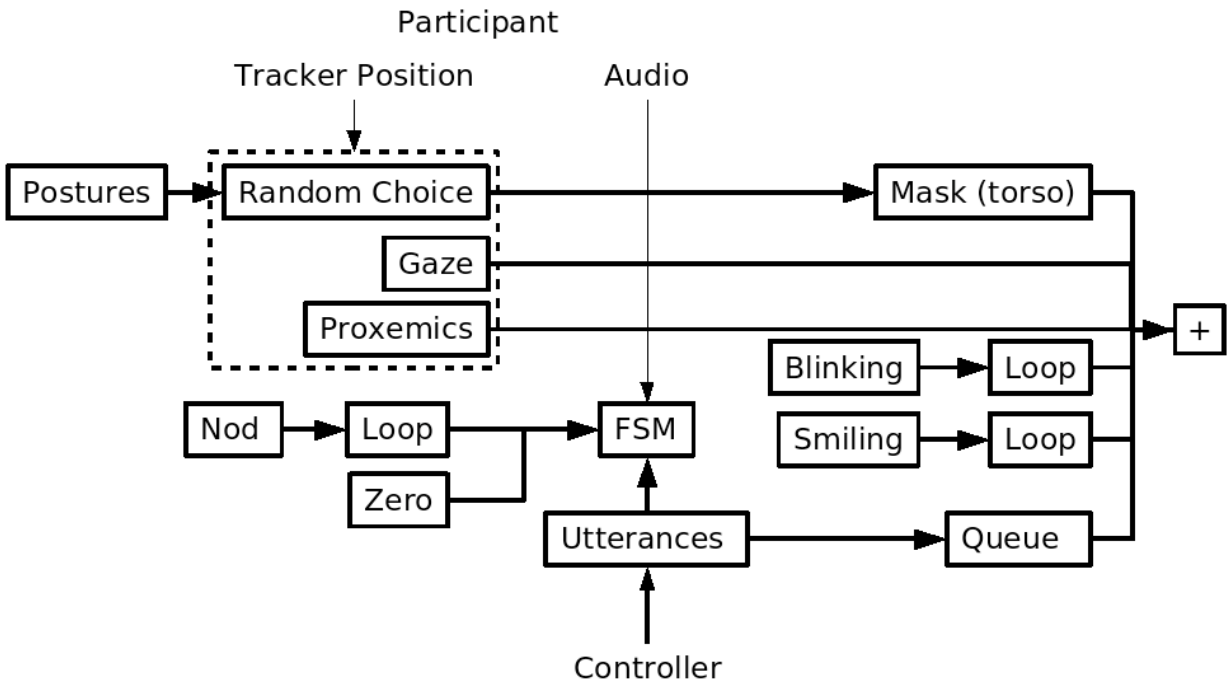

Control
Animation Generators
Rigs
Appearance

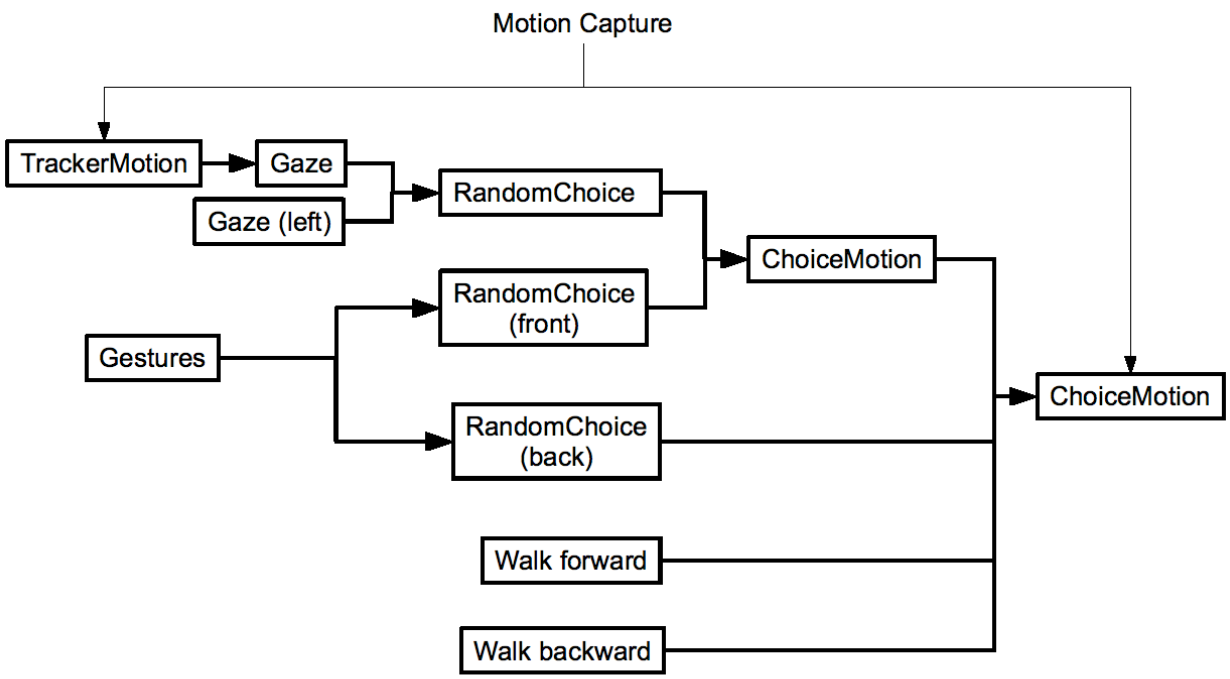

Virtual
Environment
Engine
Cal3d
Mixer
Model
GPU
Renderer
Bone
Transforms
Trigger
Actions
and
Cycles
Morph Weights
Software
Skinning
Vertices
Bone
Transforms

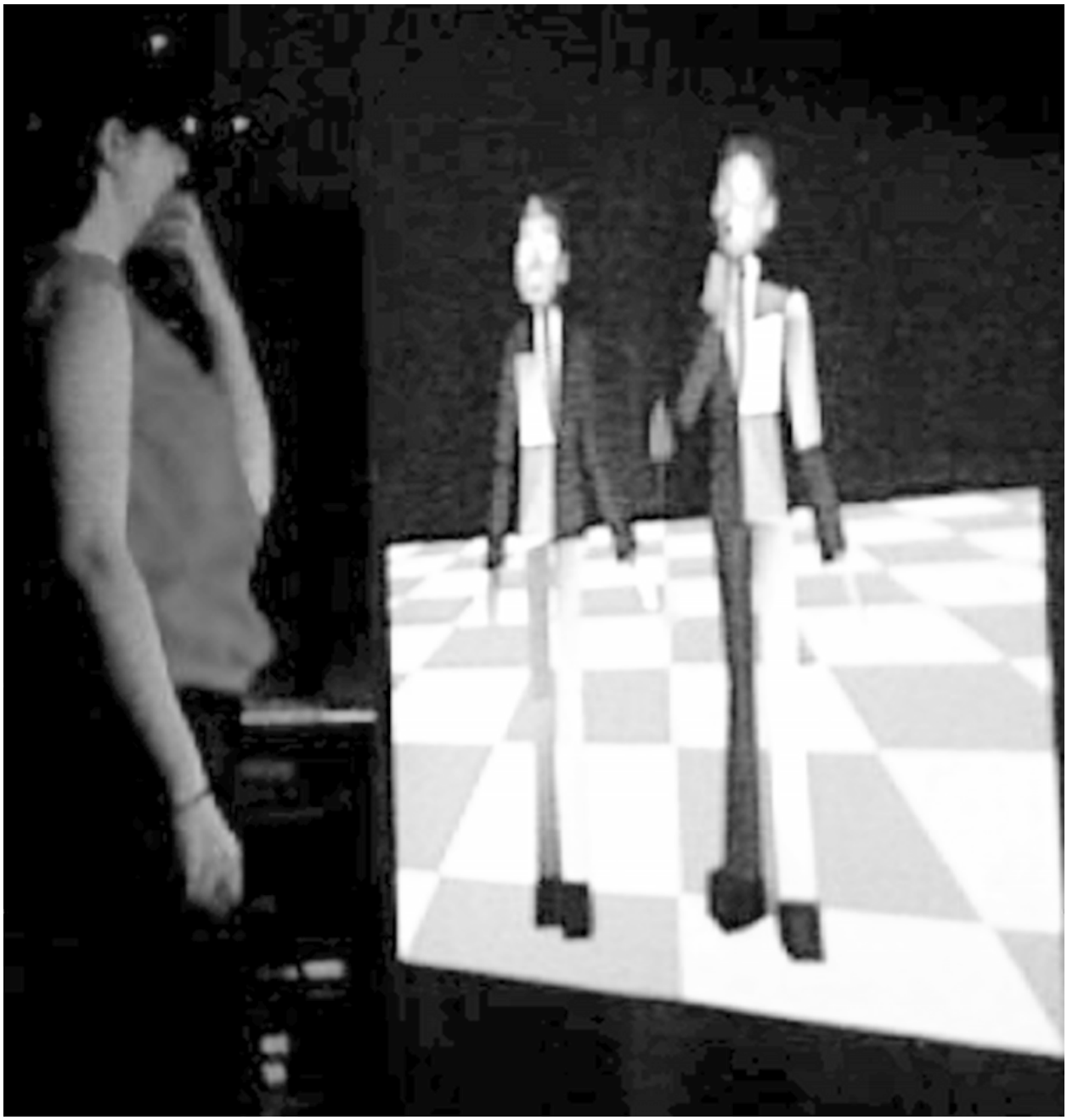


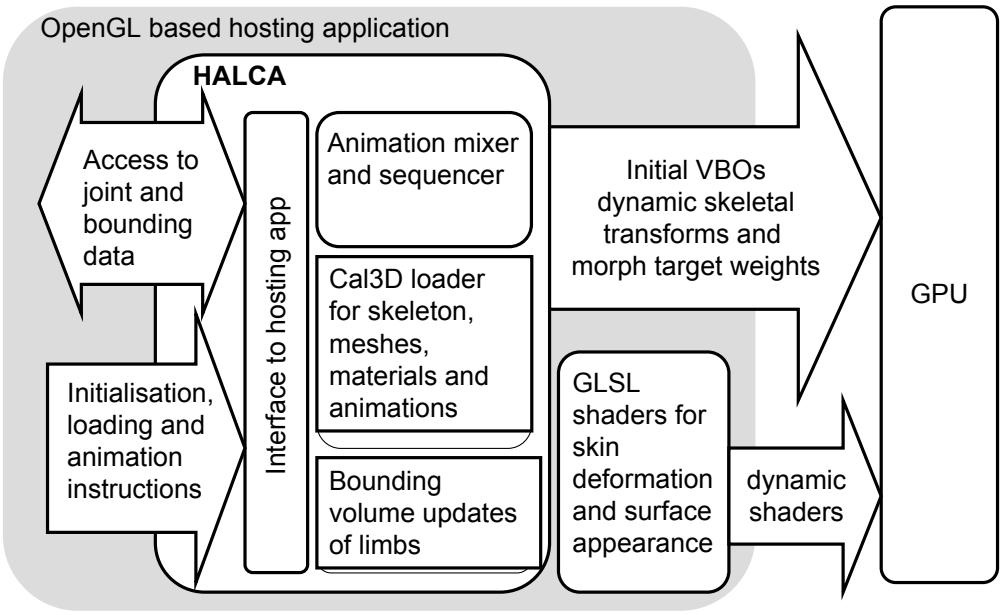

Virtual
Environment
Engine
Piavca
Python
API
C++ API
Motions
Avatar
Cal3D
GPU
Vertices
Events
Morph Weights
Continuous control
Local Bone
Transforms
Bone
Transforms


Piavca Designer
SubMotion gesture_4
KeyframeMotion discussion_gerard
SubMotion gesture_4
Start 12.3000001907
End 13.8000001907
setRange
resetRange
Avatar
Motions
Motions Types
Events
Parameters
AutoCreators
|<
>
>|
758333325386
<>


Participant
Tracker Position
Audio
Postures
Random Choice
Mask (torso)
Gaze
Proxemics
+
Blinking
Loop
Nod
Loop
FSM
Smiling
Loop
Zero
Utterances
Queue
Controller


Motion Capture
TrackerMotion
Gaze
Gaze (left)
RandomChoice
ChoiceMotion
RandomChoice (front)
Gestures
RandomChoice (back)
ChoiceMotion
Walk forward
Walk backward




OpenGL based hosting application
HALCA
Access to joint and bounding data
Initialisation, loading and animation instructions
Interface to hosting app
Animation mixer and sequencer
Cal3D loader for skeleton, meshes, materials and animations
Bounding volume updates of limbs
Initial VBOs dynamic skeletal transforms and morph target weights
GLSL shaders for skin deformation and surface appearance
dynamic shaders
GPU

	Appearance	Rigs	Animation Generators	Control
Cal3D	a basic polygon renderer	skeleton and morph targets	animation data and basic blending	N/A
Piavca	as Cal3D	as Cal3D + hardware morph targets	generic motion generation and combining system	event based and continuous control methods
HALCA	as Cal3D + plugable shader architecture	as Cal3D + dual quaternion skinning	numerous data driven and procedural generators	event based, real time tracking and physics based interaction
UNREAL	State of the art Graphics Engine	skeleton and morph targets	data driven, physics and inverse kinematics based generators	Scripting language with built in finite state machine and path planning AI
Jack	a basic polygon renderer	skeleton	numerous procedural generators	various, including AI and speech driven interaction
VHD++	fully featured renderer including hair and cloth	skeleton, facial bones and morph targets	data driven and procedural generators + real time tracking	real time input and AI
Havok	N/A	skeleton and morph targets	extensive data driven, inverse kinematics and physics based generators	N/A
Natural Motion	N/A	skeleton	extensive data driven and physics based generators	N/A
SAIBA + GRETA	polygon renderer	complex facial rig	complex procedural controllers tailored to non-verbal communication	via the BML language