A tool for constructing 3D Environments with Virtual Agents

Spyros Vosinakis, Themis Panayiotopoulos
Knowledge Engineering Laboratory, Department of Informatics,
University of Piraeus, 80 Karaoli & Dimitriou str., 18534, Piraeus, Greece
spyrosv@unipi.gr, themisp@unipi.gr

Abstract. The use of Virtual Environments as a user interface is essential for certain types of applications, both in education and entertainment. These worlds are even more attractive for the user when they are neither static nor pre-scripted, but have dynamic characteristics and are populated by autonomous entities, also called virtual agents. There has been a lot of research concerning visualization, animation and behavior of virtual agents, but there are no generic architectures, methodologies and tools for the development of intelligent virtual environments, i.e. 3D environments with autonomous virtual agents. In this paper, we present SimHuman, a tool for the construction of virtual worlds with autonomous entities, targeted for a specific group of applications, such as simple simulation systems, virtual environments, educational applications, multimedia presentations, etc. It consists of a programming library and two utilities and it is highly dynamic and configurable, as it is not based on fixed scenes and models. It has embedded characteristics such as Inverse Kinematics, Physically Based Modeling, Collision Detection and Response, and Vision. SimHuman incorporates some important features for designing and building virtual environments and turns out to be an effective tool for interactive 3D applications with virtual agents.

Keywords: virtual agents, believable agents, autonomous virtual humans, virtual environments, animation, simulation

1. Introduction

Recent advances in Multimedia and Internet technology are giving rise to interactive virtual worlds, which offer attractive opportunities both for entertainment and education [1]. The ability to navigate and interact in a 3D environment, where anything from small objects to large cities can be visualized, is essential for certain types of applications, such as virtual classrooms [2], on-line museums or shops [3], games, or even detailed models of the human body users can walk through [4].

The use of Virtual Environments as a user interface should not be limited to just browsing a beautiful 3D scene. However attractive a synthetic world may be, if it is static and empty of change and behavior, the immersive experience is of limited interest. Cities without people cannot make the user feel a real sense of presence. Animation can create dynamic environments, but its pre-scripted nature runs against the interactional freedom a Virtual Environment should have, and can only hold the user interest for a limited time. Therefore, Virtual worlds become more attractive, if there is dynamic sound and motion, and user-independent action takes place inside the environment. In other words, 3D environments would be more believable and interactive, if they were populated with virtual agents.

A virtual agent can be defined as an autonomous entity in a virtual environment [5]. It should not only look like, but also behave as a living organism (human [6], animal [7] or other fictional character [8]) in a synthetic three-dimensional world, and be able to interact with it and its inhabitants. These could be either real users in the form of avatars, or other virtual agents.

There are numerous applications that would require some form of virtual agents in their environments, especially in fields such as entertainment, education, virtual reality, simulation, etc. [9]. Nevertheless, there are no standard architectures and methods of implementing such ‘entities’ and this is due to the fact that different applications focus on different characteristics of synthetic creatures. A game, for example, emphasizes mainly in appearance and behavior, because it should combine appealing graphics with believability to attract the user. On the other hand, a simulation system aims at higher accuracy, because it is the numerical data that matter most (e.g. the total force applied on the driver in a car crash simulation).

In this paper we present a programming library called SimHuman, which is our approach towards the generation of virtual environments with synthetic characters for desktop VR applications. It has an embedded engine for rendering and animating 3D environments, which can use any 3D model for agents, avatars and other virtual objects and is therefore highly dynamic and configurable. It using physically based modeling, and its agents have an embedded vision model and can use inverse kinematics to interact with objects. SimHuman is designed in such a way that it maintains a balance between performance, autonomy and believability and can be used as a basis for simple virtual environments and simulation systems.

The paper is structured as follows: In section 2 we present the related work in the field of virtual environments and believable agents, while in section 3 we discuss about the available tools for creating interactive 3D environments. The general functionality and architecture of SimHuman are discussed in section 4, and the design issues of virtual agents is the subject of section 5. In the next section we present an example application, and in the final one we state our conclusions and possible extensions of this system.

2. Background

There has been a great amount of research in the field of visualization, motion and behavior of virtual humans or other articulated figures, and a number of different systems and approaches have been proposed. Each of these approaches varies in functionality, believability and autonomy according to the application field and the required detail and accuracy.

The process of displaying and animating a synthetic human involves three different stages. The most primitive one is the visualization of the body [10], which is the same process as displaying any other 3D object, and one can therefore use standard techniques from the field of computer graphics (curved surfaces, voxels, polygons, etc). To add more realism, one could model the body as a deformable object [11]. The next stage is the modeling of the skeleton [12], which defines the moving parts of the body and the type of motion that they can perform, and the last one is the modeling of skin, hair and clothes [13], so as to produce more believable animation. The calculation of the skin and cloth motion is the most computationally intensive task, which is why it is not suitable for real-time animation.

Believable animation of a living creature is a surprisingly hard task, because we are skilled at perceiving the subtle details of motion. A person can, for example, often recognize friends at a distance purely from their walk. Because of this ability, people have high standards for animation sequences that feature humans or other creatures. For computer-generated motion to be realistic and compelling, the virtual agents must move with a natural-looking style.

Animation techniques fall into three basic categories: keyframing [14], motion capture [15] and simulation [16]. Each of these has its own advantages and disadvantages, which involve the level of control that the animator has over the fine details of the motion, the production cost, the efficiency in real-time applications, and the ability to reuse the motion sequence in different environments.

In keyframing, the animator has to specify critical, or key, positions for the body parts, and the computer fills in the missing frames by smoothly interpolating between those positions. Body postures can be defined either with the low-level forward kinematics approach, or with the more elegant inverse kinematics one [17]. On the other hand, motion capture involves measuring a real person's / object's position and orientation in physical space, and then recording that information in a computer-usable form.

Unlike keyframing and motion capture, simulation uses the laws of physics to generate motion of figures and other objects. Virtual creatures are usually represented as a collection of rigid body parts. Although the models can be physically plausible, they are nonetheless only an approximation of the body, because they ignore the movement of muscle mass relative to bone. Recently, researchers have begun to build more complex physical models based on bio-mechanical data, and the resulting simulations are becoming increasingly lifelike [18].

There have been some significant approaches towards the introduction of synthetic figures in virtual environments, such as the work of Kalra et al.[19], which describes an interactive system for building realistic virtual humans for real time applications, and that of Aubel et al. [20], which presents techniques for rendering and animating a multitude of virtual humans in real-time. Both

approaches focus on algorithms for modeling and rendering virtual figures, without presenting a tool for embedding these figures in applications.

One important application that utilizes many aspects of human motion and simulation is a commercial system called Jack, developed at the University of Pennsylvania [6]. It contains kinematic and dynamic models of humans based on biomechanical data and displays several built-in behaviors including balance, reaching and grasping, walking and running. Jack is targeting at the fields of industrial simulation, human factor analysis, and similar environments that require accurate biological and mechanical modeling of the human body. Nevertheless, not much attention is paid on appearance, and the application features only one type of male and female bodies, which can be scaled to various sizes, without using particularly attractive graphics.

A similar environment is HUMANOID [21], which is additionally using metaballs for a more realistic representation of the skin and muscles. Focusing more on appearance, it features realistic skin deformation for the body and hands, and facial animation. Unfortunately, it uses its own type of human models, so one cannot import new models from commercial programs, but one has to design them manually. Additionally, like Jack, HUMANOID is running only on graphic workstations, such as SGI and SUN systems, meaning that one cannot create a simple application for average PC users. Furthermore, there is much detail on how the environment can render and animate deformable characters, but the paper does not describe how a programmer can use that environment to construct a virtual world with multiple agents.

Another application, which is using virtual agents in complex 3D environments, is Steve (Soar Training Expert for Virtual Environments) [22], an animated agent that helps students learn to perform physical, procedural tasks. Steve inhabits a virtual world, continuously monitoring its state and periodically manipulating it through virtual motor actions, using perception, cognition and motor modules. Steve is using a sophisticated intelligent agent implementation in a 3D environment, but it is limited in educational applications. Additionally, it is using Jack as a rendering and animation engine and has, therefore, all its limitations.

One interesting application towards a 3D environment with intelligent virtual agents are the Virtual Teletubbies [8]. The system uses a simple physics model to simulate gravity and the agents' personality is based on a novel behavioral architecture, the Behavioral Synthesis Architecture. The paper presents a novel approach, which applies a robot architecture to virtual agents, but it cannot be considered as a general tool, as there are no hints on how to construct other applications using this method.

Agents with personality are also the main subject of the work of D. Silva et al [23]. They propose a Synthetic Actor model that connects emotions and social attitudes to personality, providing a long-term coherent behavior. They present two games as case studies to their proposed architecture. The Synthetic Actor model is a very interesting behavioral model, but there is no indication on how to interface it with a virtual environment. The case studies they present have been implemented using the Java / VRML platform, which is not one of the most efficient ones.

The EXCALIBUR project [24] is a system that uses a generic architecture for autonomously operating agents that can act in a complex computer-game environment. The agents use planning in a constraint programming framework and the behavioral model is able to handle incomplete knowledge and information gathering. The project has been built for computer games and there is no specific information on how to connect the proposed architecture to a virtual world and how to interface it with the sensing and acting mechanisms of a virtual agent.

Another interesting system for the creation of real-time behavior-based animated actors is Improv [25]. It consists of an Animation Engine that uses procedural techniques to generate layered, continuous motions and transitions between them, and a Behavior Engine that is based on rules governing how actors communicate and make decisions. The combined system provides an integrated set of tools for authoring the 'minds' and 'bodies' of interactive actors. Improv seems, however, to be more suitable for interactive storytelling rather that virtual environments, since there is no indication that the system is using collision detection and physics, while the actors have total freedom to manipulate the world (even to change other actors' properties), so they cannot be considered virtual agents.

Our research team has also done some preliminary work on virtual environments with autonomous entities during the last years. We have worked towards the introduction of logic programming in virtual worlds and we designed and implemented an intelligent agent framework in VRML environments [26]. After a few years, we abandoned the Java / VRML platform and

started using C++ and OpenGL to build a generic tool for rendering and animating virtual agents, a research, which lead to SimHuman.

3. Creating Interactive 3D Environments for Multimedia and Educational Applications

There are no general methodologies and architectures for the development of 3D interactive environments, because it is usually the target application, which sets the requirements. Some systems may have to use models with great detail, while others may need to render very large scenes. There are cases where accurate modeling of the physical laws is needed, and others where the emphasis is put in visual appearance. In general, the fact that high-quality graphics and animation decrease the performance dramatically, makes it very hard to set standard features for all possible applications with virtual environments.

We believe, nevertheless, that there are certain types of applications, such as educational environments, multimedia presentation systems, multi-user chat rooms and simple simulation systems, which have similar requirements. Such environments usually involve relatively small spaces and need visually appealing graphics and animation combined with a simplified physical model. The combination of attractive graphics, complex animation and physics should make the environments more realistic and believable to the user. Furthermore, such applications should be dynamic, i.e. to allow the user to interact with the environment, through navigation, manipulation of objects, or assignment of orders to virtual agents, using the appropriate interface, e.g. speech processing, natural language commands, joystick, mouse, etc. In all these cases, the use of commercially available products such as Macromedia Flash or QuickTime VR is not enough, because they are restricted either to 2D graphics or to static 3D environments with limited interaction.

As seen in the previous section, there has been a lot of research in the field of virtual worlds with synthetic characters and all these approaches have many important features to offer, but in all cases, the systems seem to be targeting at a specific subgroup of applications. There are sophisticated tools for designing and animating virtual agents, which are running only in expensive graphical workstations, and in some cases they focus more on accuracy than on visual appearance. On the other hand there are some interesting approaches that use intelligent agent architectures in virtual environments, but are presented as case studies and not as a way to design and implement custom worlds.

There are cases where complex behavioral architectures for 3D environments have been proposed, lacking, however, details on how these approaches are interfaced with a graphical environment, and whether it is possible to use them in a custom application. Finally, there are tools such as Improv, made especially for the design of synthetic actors, which focus more on storytelling and lack features, such as collision detection and physics, that would allow the development of simple simulation systems with virtual agents.

There are commercially available tools for rendering and animating virtual agents, such as the 3D Game engines, but due to the fact that these engines are built especially for games, they also have a number of limitations. In most of the cases they use their own file format for 3D models (e.g. the MD3 files used by Quake III engine), which is not widely supported, so the user may have to use specific editors to design a scene. Furthermore, the agents have cannot use an arbitrary body hierarchy (they usually have simple skeletons) and there are few actions that they can implement, such as running, jumping and picking objects. It is very hard (if not impossible due to the simplicity of the skeleton) to design more complex actions using game engines, e.g. to have an agent sit on a chair, or open a door. Finally, game engines tend to run in full screen, or at least to cover the whole application window, limiting the interaction with the user through messages and keyboard input. Therefore, one cannot use such an engine to have a virtual world as a part of a larger application.

As a conclusion, we believe that there is a lack in general tools for designing and implementing single-user interactive 3D environments with multiple virtual agents that can be embedded in applications for average PC users. Therefore, we have designed and created SimHuman as a tool to build such environments.

4. SimHuman

The field of character modeling, simulation and behavior is an area of continuous research and many different approaches have been proposed. In our case, we tried to combine algorithms that balance between efficiency and accuracy in order to produce believable motion in real-time environments and to create a useful tool for the development of interactive 3D applications.

SimHuman is a tool that can aid programmers in the design, rendering and animation of virtual environments with single or multiple agents and avatars. These synthetic worlds have characteristics such as Inverse Kinematics, Physically Based Modeling, Collision Detection and Response, and Vision, and can be easily embedded in larger multimedia applications.

SimHuman consists of a programming library and two utilities. The library is implemented in C++ and allows users to define and animate three-dimensional scenes with an arbitrary number of objects, virtual humans and user-controlled avatars. The utilities are *Agent Designer*, a program that loads agent models and helps users design the skeleton and create animation libraries, and *Scene Designer*, a program that lets users place objects in a scene and visualize sequences of agents' actions.

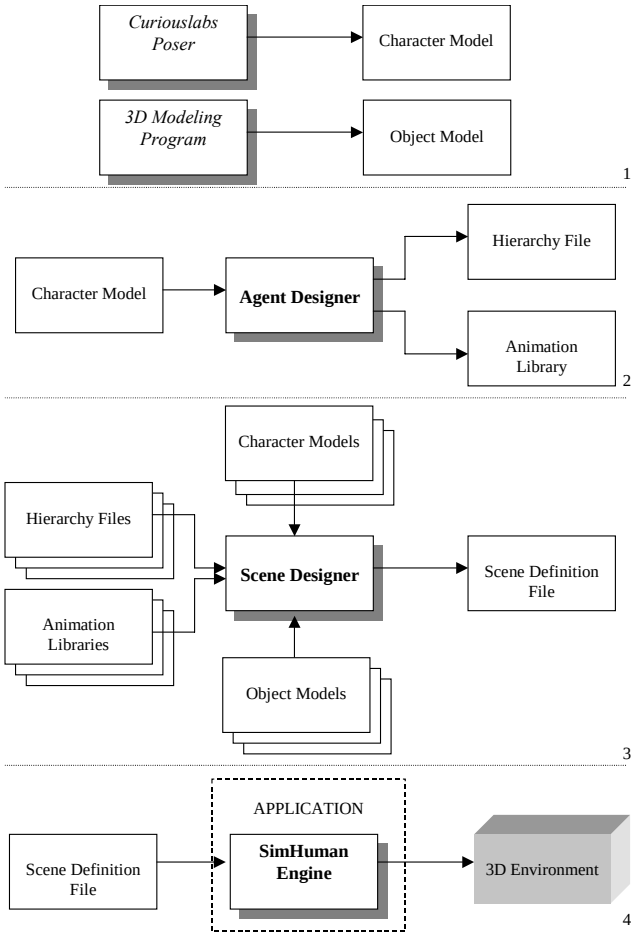

Figure 1: The process of creating a 3D Environment with SimHuman

4.1. Creating an application with SimHuman

The process of using SimHuman for the generation of an interactive 3D environment as part of a larger application consists of four stages (Figure 1). First of all, the user has to define the geometry of the objects and agents that will be part of the world. Then, a more detailed description of the

virtual agents is needed, mainly their skeleton hierarchy with the joint limits, as well as an animation library, which contains the actions that the agents can perform. After that, one has to design the scene, i.e. to define how the objects and agents will be initially arranged inside the environment. Agent Designer and Scene Designer can help the user construct the agents' skeleton and animation libraries, as well as place the objects and define the initial settings of the 3D environment.

The final step is to construct an application using the SimHuman library. The library's main feature is the *SimHuman engine*, a global engine that coordinates the synthetic characters' activities and animates the virtual world. The user has to define how the agents 'behave' in the 3D environment and how the world changes during time. These behaviors are assigned to the engine as callback functions.

We will now present in more detail the files that are used by the SimHuman engine to render and animate the environment.

Character geometry and motion

Each virtual agent is using three different files for rendering and executing its actions, a *geometry model*, a *hierarchy file* and an *animation library*. The geometry model can be imported from Curiouslabs Poser, a commercial program for modeling, posturing and animating synthetic characters. Character motion is defined by the hierarchy file, which represents the virtual agent's skeleton, i.e. the joints and segments of the body and their limits. The animation library holds a set of keyframed animation sequences and can be used by synthetic characters to implement various actions.

Object geometry

The geometry models for the objects that will compose the scenery can be imported from commercial 3D Modeling programs provided that they are in VRML97 format.

The Virtual Reality Modeling Language may not be as popular as it used to be a few years ago, but it is still a universal file format for 3D objects, and one can find numerous free VRML objects on the Internet. Many commercial 3D Modeling programs, such as 3D Studio Max, can export to VRML97 files, and there are also file conversion utilities (e.g. 3D Exploration), that can convert almost any popular 3D file format to VRML97.

Scene Definition

The most primitive step in using the SimHuman library is to define a scene, i.e. a 3D environment with an arbitrary number of objects and virtual humans. This scene can be loaded from a *scene definition file*, which describes the objects and their attributes, or created dynamically during the execution of the program by constructing new C++ objects. Each object in the scene definition file has the following form:

```
OBJECT classname name modelname
   Position px py pz
   Orientation rx ry rz

   Spot name px py pz
   Spot name px py pz
   …
END OBJECT
```

Each object has a unique name and its geometry is described in a VRML model. The *classname* variable is for the class that the object belongs to, which may be useful for the implementation of certain actions (e.g. one can sit on an object of class *chair* but not on an object of class *window*). Each object has also a position and orientation in the 3D space (denoted by the variables (*px*, *py*, *pz*) and (*rx*, *ry*, *rz*) respectively), and an arbitrary number of spots which are again used for the implementation of human-object interactions as we shall describe later. Each spot has a unique name and its position in the local coordinate space of the object is (*px*, *py*, *pz*). The reason for using the object's coordinate space instead of the global one is to give the user the

ability to easily change the position and orientation of the object in the scene without having to readjust all the spots. All VRML models of the scene are declared in the beginning of the file as:

```
MESH modelname filename sx sy sz
```

Filename is the name of the wrl file, *modelname* is a unique name assigned to the model, which is referenced in the OBJECT declaration, and (*sx, sy, sz*) is a global scale factor of the model.

A sample screen definition file for a scene with one table and two chairs is the following:

```
MESH smallchair chairnew.wrl 0.4 0.4 0.5
MESH bluetable metaltable.wrl 0.5 0.5 0.5
END

OBJECT chair chair1 smallchair
Position 0.0 0 0.0
Orientation 0 90 0
Spot s1 0.0 0.22 0.0
Spot s2 0.19 0 0
Spot s3 1.19 0 0
END

OBJECT table table1 bluetable
Position 0.0 0 0.3
Orientation 0 0 0
END

OBJECT chair chair2 smallchair
Position -0.3 0 0.3
Orientation 0 0 0
END
```

The 3D scene generated by this scene definition file is depicted in figure 2.

Figure 2: A sample scene with a table and two chairs

Virtual Creatures can be either computer controlled characters (Agents) or user controlled (Avatars). Their definition resembles that of the virtual objects, but in addition to the geometry model, one has to add the name of the hierarchy file and the animation library, which describe the virtual character's joint structure and the actions it can perform. We shall describe the definition and functionality of virtual agents in more detail in the next section.

4.2. Using the library

The SimHuman library is a C++ library that can be linked to a larger application and render and animate an interactive virtual environment with one or more virtual agents. The method of using this library in an application is to define the initial objects and their properties and then to initialize

the SimHuman engine using the appropriate callback functions. The initial scene definition can be loaded from the scene definition file. Alternatively, the library allows the dynamic creation or destruction of objects and agents, so an application could possibly have an interactive object placement and initialization of the environment. Another issue that the programmer can control is whether an object (or agent) will take part in the physically based modeling and collision detection process or not. This is mainly for performance reasons, e.g. static objects such as walls do not change their position over time and have always a zero velocity, so there is no need of applying gravity on them and testing their collision with the ground. On the other hand, moving objects do have a velocity and it is possible for them to collide with other moving or static objects.

The SimHuman engine is responsible of animating and rendering the virtual agents and the other objects of the environment. During each timeframe, all agents follow a Sense – Decide – Act sequence, while the world is causing changes to the objects based on any global laws that may exist in the environment. After these changes in the object properties, the physically based modeling module applies the laws of physics to the objects and tests for possible collisions. This process may also alter some geometric properties of the world's entities, which may be sensed by the agents only in the next timeframe. Using the final positions and orientations of the objects, the visualization module renders the scene on the screen. The process of rendering a frame in the SimHuman engine is displayed in Figure 3.

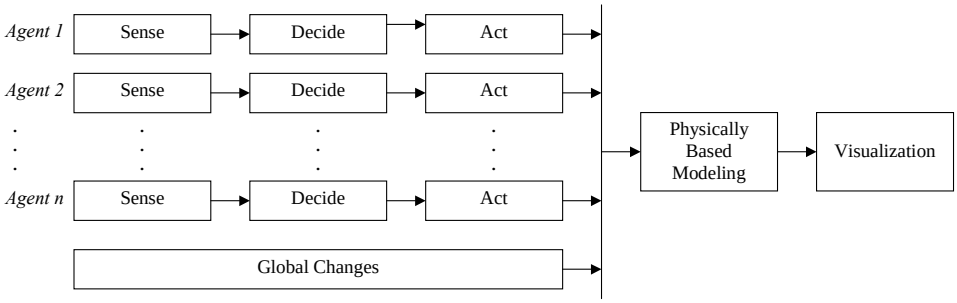

Figure 3: Operation of the SimHuman engine

The engine is using processes for agents' sensing, deciding and acting, but, unfortunately, the way that an agent should react to the environment depends strongly upon the application. Therefore, the user should assign callback functions that will determine how each agent should 'behave' based on the current state of the world. There is an embedded sensing mechanism, which may be used for additional believability, and there is also a set of different actions that the agents can perform to interact with the environment. To create an avatar in the environment, one can simply use a callback 'Decide' function that maps user input (keyboard, joystick or mouse) to specific agent actions.

The global changes that will be applied during time (e.g. time of day, weather, other laws) are also determined by a callback function that the user should assign to the engine.

4.3. Physically Based Modeling

Traditional animators rely upon a mixture of visual memory and drawing skills to simulate various physical behaviors. When a line test reveals that the animator is not correct, it is redrawn and re-tested until it satisfies the animator. Unfortunately, such techniques cannot be used in real-time computer graphics systems. If the aim is realism, one must rely upon deterministic procedures that encode knowledge of the physical world, which is the physically based modeling process. Moreover, in such a simulated environment all objects have to behave as rigid bodies, i.e. no object should be allowed to penetrate another. The process of collision detection is to examine if the whole or part of the geometry of an object is within another. Associated with this is another equally important process, that of collision response, which should determine the new position and velocity of the two (or more) objects that collided.

SimHuman has an embedded physically based modeling engine that is used to enhance the naturalness of the animation. The engine uses a simple and fast approach to simulate forces and to

check for collisions. There are more sophisticated approaches that can, however, be implemented in the future, and easily replace the current one, due to SimHuman's modular architecture.

The physical simulation is conducted in discrete timesteps, where each object has its own mass, position and velocity. In each timestep objects' positions and velocities are recalculated following the laws of kinematics and collision. It is optional whether an object will participate in the physically based modeling engine or not, and it depends on the target application. Collision detection is also optional and it is using the bounding boxes of the objects, which are automatically calculated when the geometry models are loaded.

The Physically based modeling process in each timestep *dt* is as follows:

For each object

1. Calculate the total force **F** applied to it. This will give rise to acceleration ***a*** such that: $\mathbf{F} = \mathbf{m} \cdot \mathbf{a}$. The value of **a** is calculated.
2. Assuming the object's velocity in the previous timestep was $\mathbf{v_0}$, then the current velocity **v** is: $\mathbf{v} = \mathbf{v_0} + \mathbf{a} \cdot dt$
3. Assuming the object's position in the previous timestep was $\mathbf{p_0}$, its new position **p** is calculated as if the object had moved with a constant velocity, the mean between that of the previous and the current timestep. So: $\mathbf{p} = \mathbf{p_0} + \frac{\mathbf{v} + \mathbf{v}_0}{2} \cdot dt$
4. Check if the object collides with any other object in the scene. If not, assign the new position **p** and the new velocity **v** to the object and proceed to the next one.
5. Assuming that the other object did not move during *dt* find the position $\mathbf{p_c}$ of the object exactly when the collision occurred. Find the relation λ between the impact position and the current one to approximate the exact time of impact and the object's velocity at that time.

$$\lambda = \frac{|\mathbf{p_c} - \mathbf{p_0}|}{|\mathbf{p} - \mathbf{p_0}|}, \; 0 < \lambda \le 1 \qquad \mathbf{v_c} = \lambda \cdot \mathbf{v} \qquad dt_c = \lambda \cdot dt$$

6. Assign the position p_c and velocity $\mathbf{v_c}$ to the object and repeat the process (go to step 1) for the rest of the time $dt - dt_c$

4.4 SimHuman Utilities

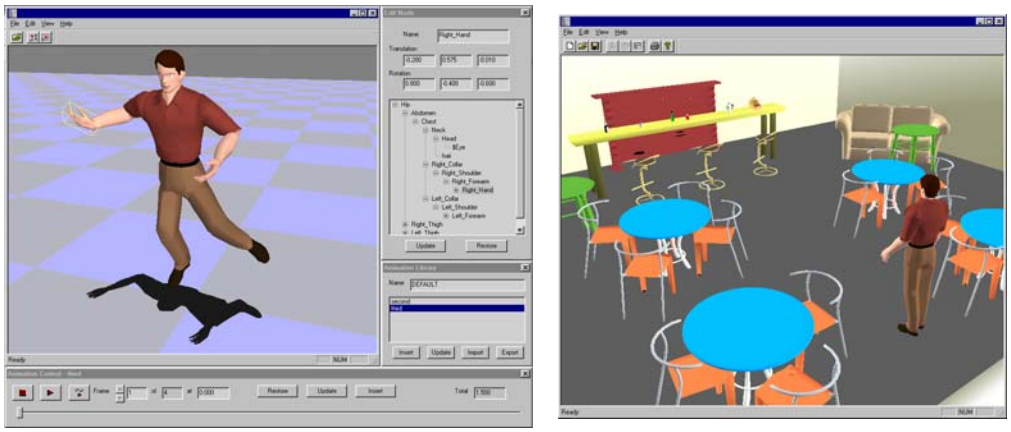

Figure 4: Agent Designer (left) and Scene Designer (right)

The SimHuman library is accompanied with two additional programs that render and test 3D Scenes and agents, and aid the programmer during the development of applications based on virtual environments.

The first one is Agent Designer, a program for constructing the skeleton and creating animation libraries for virtual agents. It has the ability to load and render agent models and visualize their skeleton, where one can rotate the camera and zoom in or out to examine the model from different views. The user can select a body segment and rotate it in any direction to test and readjust the

joint limits and the centers of rotation. He/she can also change the joint's parent and children in the tree hierarchy using drag and drop in the appropriate dialog. With this method, one can create the appropriate joint hierarchy file or adjust an existing one to the needs of the application.

Multiple segment rotations create body postures, which the user can export to text files. Additionally, one can define or load series of postures and create keyframed animation sequences, which can, then, be used by the agent as actions in the 3D Environment. The SimHuman engine is able to select animation sequences from library files and execute them during runtime.

The other program is Scene Designer, which is built for visualizing and testing 3D Scenes. The user can load and examine any 3D environment from the scene definition file and easily readjust the objects' positions and rotations. He / she can also select avatars and directly manipulate them to test the walking algorithm, the collision detection with objects and the physically based modeling engine. Additionally, the program can visualize the agent's sensing algorithm with ray-casting and display the objects that are visible to the agent during each timeframe. One of the most important features of the program is that it can execute a sequence of actions for each one of the agents, and the user can therefore setup different environments and test the outcome of certain actions or action combinations.

Using these two tools, the programmer can refine the agent models and scenes and create all the necessary input files that will be used by the SimHuman library.

5. Virtual Agents in SimHuman

In this section we will focus on the design issues and functionality of the virtual agents that are part of SimHuman. We will present issues such as primitive motion, walking, collision detection, sensing, dynamic actions, and behavioral control.

5.1. Display and primitive motion

In the SimHuman library, all objects of the scene (including virtual agents' bodies) are defined in terms of 3D Polygons. Agents and avatars are not based on a fixed model, but the library gives the ability to load models dynamically, from a geometry file that contains the details of their appearance. The models can be loaded directly from Curiouslabs Poser, a commercial program for the design, posturing and animation of virtual humans or animals. Therefore, it is practically very easy to design new virtual agent models and embed them in an application based on SimHuman.

A synthetic character's body could be just one big polygon mesh, but it would be very difficult to animate it. If, for example, a virtual human had to raise its arm, the program would have to find which vertices and polygons belong to the arm and only rotate these. A much better approach is to define the body in terms of joints and segments, where each segment should be the body part that connects two joints and should maintain its geometry throughout the animation. The number of joints and segments that should be defined on the human body depends on the application and the animation detail required. So does the number of degrees of freedom allowed per joint. A real human has in total over two hundred degrees of freedom, but efficient animation can be produced with significantly less. A simple walking animation may need less than a dozen joints. On the other hand, a virtual human that can grasp various objects requires a lot of additional joints and segments (for the fingers of each hand).

In our approach a virtual character can have an arbitrary number of joints and segments and any possible hierarchy tree (or skeleton) to connect them. This information is stored in a supplementary file called hierarchy file, which gives the user / programmer the ability to adjust the program to the needs of different applications having any possible detail.

The hierarchy file has the following structure:

```
NODE nodename
Centre cx cy cz
Max rx1 ry1 rz1
Min rx2 ry2 rz2

   NODE nodename
```

```
    …
    END
    NODE nodename
    …
    END
    …

END
```

Each node has a unique name (*nodename* variable), which is the name of the body part it controls. The center of rotation expressed in global coordinates of the body is (*cx, cy, cz*), while (*rx1, ry1, rz1*) and (*rx2, ry2, rz2*) are the maximum and minimum rotation values allowed, the joint limits. Between the declaration of the Node and the END keyword, one can declare an arbitrary number of joints, which are the descendant nodes of the hierarchy tree.

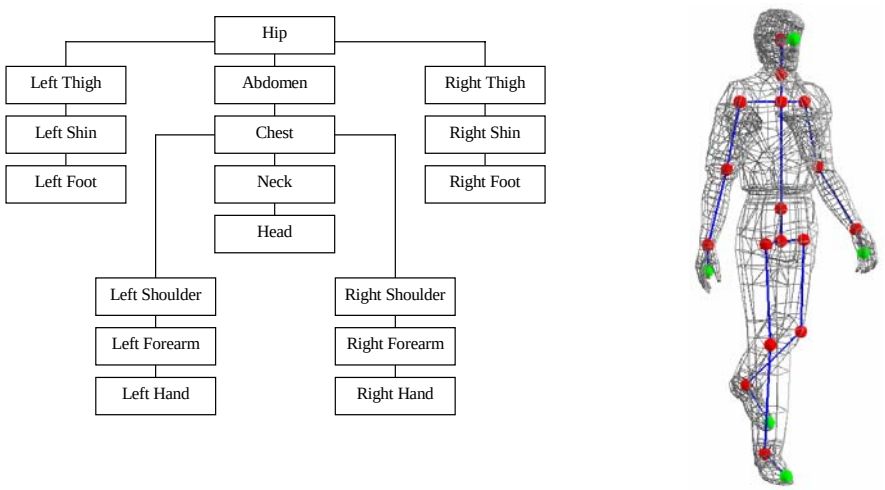

Figure 5: A sample hierarchy tree and its respective representation in the human body

Figure 5 displays a hierarchy tree and shows a simple human body mesh in wireframe and the underlying skeleton, which is defined by that tree. The red spheres are the joints of the skeleton, which transform the body parts to generate animation sequences, while the green spheres are dummy joints, since they do not cause any transformation on the human body. The dummy joints on the hands and feet are used as end-effectors in inverse kinematic chains to implement actions such as catching an object. The joint on the head is the 'eye' of the virtual human and is used for first person view in the case of avatars and for sensing the environment in the case of agents.

A classic problem of articulated figures (i.e. figures that are based on a skeleton model) is that the rotation of a segment causes a crack around the joint, because the faces of adjacent segments are not adjoining anymore. One way to deal with this is to have fixed primitives (usually spheres) on the joints, but one drawback is the fact that these primitives do not always fit perfectly with the model's meshes, thus distorting the appearance of the model. Another problem is that they add a lot more polygons to the scene, which may decrease the program's performance.

In our engine we have the ends of adjacent segments always connected with each other and, whenever a segment is rotated, the vertices that are common with any adjacent segment are not transformed. The effect of this method is demonstrated in Figure 6. With SimHuman, one can generate arbitrary animation sequences using keyframing, where the engine produces a smooth transition between two or more predefined states (poses). With this simple process of keyframing, one can load various body postures stored in an animation library and use them to create more complex animation sequences, such as walking, sitting on a chair etc.

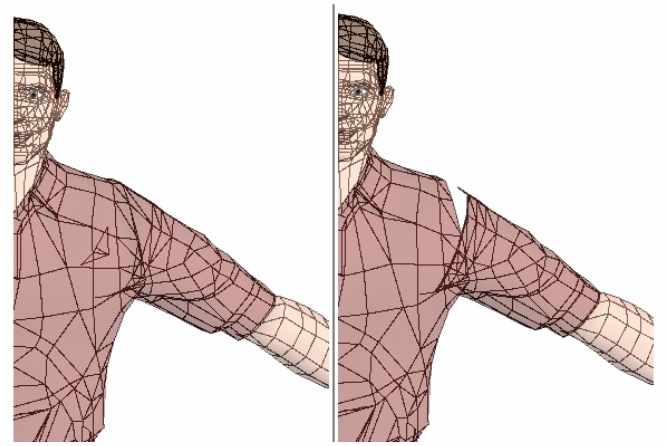

Figure 6: Shoulder rotation with and without common vertices

5.2. Walking animation

The animation of a walking human is a typical keyframing example. This is because of the nature of walking, which is a continuous transition between several states. During walking a person changes between the following states:

- State 1: the left leg touches the ground and pushes the body forward
- State 2: the right leg is on the ground and drives the body

Nevertheless, a walking animation should contain two more states. One is the transition from current position to state 1 to start walking, and the other is the transition from any state to a rest position to stop walking. This state – transition process is only capable of animating walking along a straight line. However, in most of the cases, the virtual human may have to change its direction while walking. One cannot do this by simply interpolating the global rotation, because the motion will not look natural. In reality, humans use their feet to rotate; they cannot just turn their bodies while walking. The body rotation is a motion that has two extra states, as shown in figure 7. These two states may not be enough, because one cannot usually rotate the body more than ninety degrees using just two steps. If the rotation angle is bigger the rotation process is split into two smaller ones.

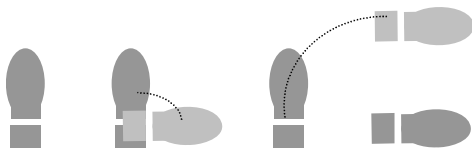

Figure 7: The two states for rotating the body

If the human body is resting, the rotation process can start immediately. This is, however, not the case if the virtual human is walking. If the rotation starts on arbitrary time, there might be a case where no leg touches the ground, and the animation will look unnatural. Therefore, the rotation has to be coordinated with the walking to achieve better visual results.

When the human is walking and the body has to be rotated, the program waits until one of the two legs is on the ground and the body is standing on it. The body is then rotated around that leg, so that the whole process is animated smoothly and realistically enough. The state diagram for the walking process is shown in figure 8.

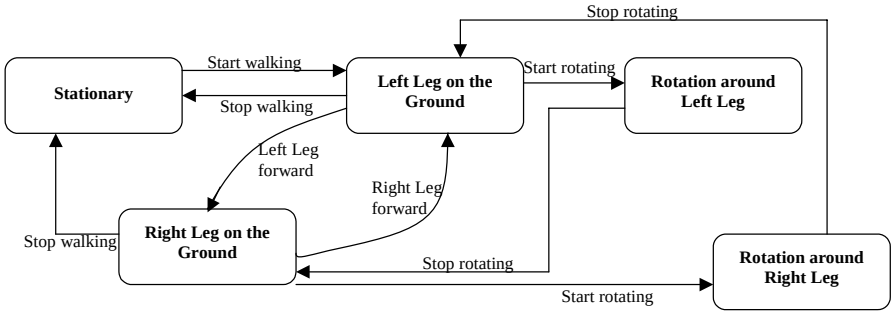

Figure 8: State diagram for walking

5.3. Collision detection and response

Calculating an accurate collision detection and response for a synthetic character's body mesh is not an easy task. The program has to check each polygon of the mesh against all the other objects of the scene and determine if it is penetrating another polygon. The approach that we use in our engine is somehow different. We use bounding primitives around the human body segments and perform all collision detection checks with them. This reduces the computational time significantly, making it possible to run in acceptable speeds in real-time applications. The primitives that we have chosen are cylinders, mainly because their geometry is more symmetrical compared to bounding boxes and they still fit well around the human body (figure 9). Once a model is loaded, SimHuman automatically calculates the bounding cylinders of each body segment.

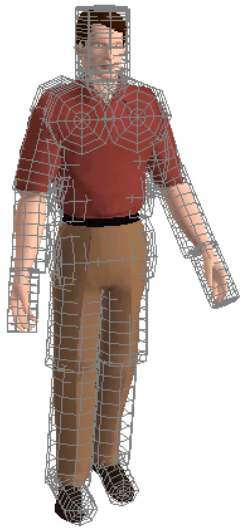

Figure 9: A human body mesh and the respective bounding cylinders

The bounding cylinders need to be calculated once and then they are only translated and rotated following the motion of the limbs they correspond to. This calculation is done initially, when the human model is loaded. The process is simple and follows the next three stages for each limb (mesh) of the body:

1. **Calculation of the bounding box**. The program checks all vertices of the mesh and finds the maximum and minimum *x*, *y* and *z* values. The result will be two vectors (**max** and **min**)

which will define the corners of the bounding box. The size and center of the box can be found very easily: **size** = **max** – **min** and **center** = $\frac{\mathbf{max}+\mathbf{min}}{2}$.

2. **Cylinder type**. Each cylinder can be aligned on the x, y or z axis. The use of arbitrary cylinders has been avoided because it would complicate the calculations significantly. This is not a drawback since they can be rotated and translated as any other object in the scene, it just affects their initial definition. The axis on which the cylinder will be aligned is be the one with the highest value in the **size** vector of the bounding box.
3. **Centre, radius and height**. The cylinder's center is the center of the bounding box. Its height (or length, or width, depending on the axis) is the **size** value of the axis on which it is aligned. The cylinder's diameter is the mean between the **size** values of the two other axes. For example, if the cylinder is aligned on the x-axis and the size vector of the bounding box is [*sx sy sz*], then the length will be *sx* and the diameter $\frac{sy+sz}{2}$. So the radius is going to be $\frac{sy+sz}{4}$.

After defining the initial position and size of each cylinder the next step is to check for collision with other objects in the scene. In each timestep the cylinders have the translation and rotation of the limbs they correspond to. Let us for example check if a cylinder collides with a sphere. Instead of transforming the cylinder, the program inverse transforms the position and velocity of the sphere and checks for collision with the initial cylinder. If the objects collided the new position and velocity of the sphere has to be transformed back to the original coordinate system. The steps are:

1. Find the matrix of the corresponding mesh and calculate its inverse.
2. Multiply the inverse matrix with the center and velocity of the sphere
3. Check if the sphere collides with the initial cylinder
4. If they do, calculate the new position and velocity values and multiply them with the transformation matrix.

The collision detection of a primitive with a cylinder is not a very complex task. Let us consider the case of an x-aligned (horizontal) cylinder and test if a point is inside it. Let the centre of the cylinder be [*cx cy cz*], its radius be *R* and its length *l*. Let also the point be [*px py pz*]. The point is inside if and only if:

- $cx - \frac{l}{2} \le px \le cx + \frac{l}{2}$ and
- the distance of the point from the cylinder's axis is less than the radius. In other words: $\sqrt{(cy-px)^2(cz-pz)^2} \le R$.

Similarly one can check if an edge is penetrating a cylinder, and with these two primitives (points and edges) one can test boxes, polygons or any kind of mesh.

5.4. Sensing the Environment

A virtual agent has to have a way of interacting with its environment, i.e. to perform actions on objects or other agents and cause a change in the state of the scene. A necessary precondition for such an interaction is to have a mechanism to sense the environment. The agent has to be able to know other objects' position and attributes and to use this information for the execution of its actions. A virtual agent could be omniscient, i.e. be able to read the attributes directly from scene, or could have a limited view of the environment, thus making its behavior more believable to the user.

The sensing mechanism in SimHuman is based on ray casting. Whenever an agent 'looks', it casts a number of rays into the scene, and reads the objects that these rays intersect with. With this mechanism, it is able to know the position, size and type of the objects that are in its field of view, and use this information to navigate inside the scene and to perform actions on the objects. The agent keeps its own, internal map of the scene, which is memory and is updated periodically, depending on the sensing frequency. In figure 10, the agent is casting rays into a complex scene.

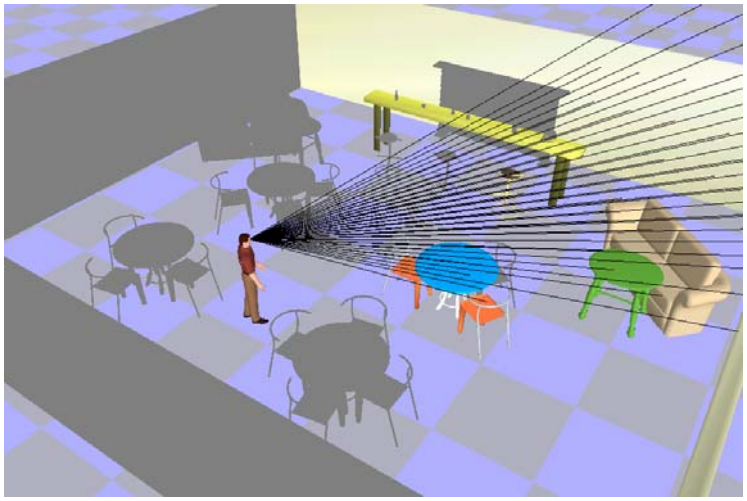

Figure 10: A virtual agent ray-casting a scene

5.5. Dynamic Actions

The actions that the agent can perform are either predefined, such as the execution of an animation sequence using keyframing, or dynamic, meaning that they depend on the environment and their outcome cannot be predicted. Consequently, dynamic actions may fail, because the current state of the environment may not allow its success, e.g. in the case of using the Inverse Kinematics engine for a target that cannot be reached. In SimHuman, the physically based modeling engine as well as the co-existence of more than one agents and avatars inevitably 'generate' a dynamic environment, and a virtual agent has to be able to execute dynamic actions to interact with it.

One such action is path planning and obstacle avoidance, i.e. the ability to move to a specified target without bumping into other objects, where the agent uses its own memory map to calculate a safe path to the target. More complicated dynamic actions involve catching a moving object, hitting it, or leaving an object on a specified location. These actions must use a form of inverse kinematics, because they involve more than one joints that have to be coordinated to succeed.

Path finding

SimHuman is having a simple embedded algorithm for path finding and obstacle avoidance. Using the Ray Casting engine, it tests if the agent will penetrate an object while walking a straight line from the initial to the target position. If it does, the segment is subdivided using a point on the perpendicular line that passes though its center. The algorithm, then, works recursively for each of the two new segments. In case it fails, it moves the point of subdivision further along the perpendicular line (both to the left and right position). The step of moving the subdivision point, as well as the depth of recursion, are both adjustable. Figure 11 shows a case where the algorithm succeeds in finding a path in a scene with three different obstacles using four steps with equal subdivisions. The use of this algorithm is of course optional; the programmer could implement other methods of controlling the agents' motion, such as a graph, a grid, etc., or use the objects' bounding boxes to perform other path finding techniques.

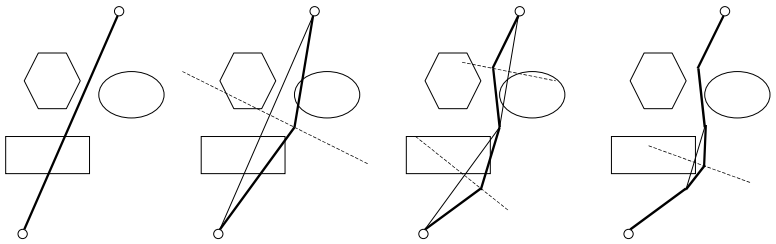

Figure 11: Path finding with the subdivision algorithm

Inverse Kinematics

Instead of using a generic inverse kinematics solver, a different approach is introduced. This approach tests at every step the best rotation for each joint to achieve the target. The set of joints (or the joint chain) that is going to be used by the Inverse Kinematics engine is first defined. Then, a function has to be assigned and return on how close to the target is the current state of the virtual human. The process (in pseudocode) is:

```
For each joint in the chain
   For each degree of freedom of the joint
      increase and decrease angle by a value v
      check which of the two states improves the
        function
      if that state is different from the
        previous one
         decrease the value of v
      else increase it
      assign the value to the angle
```

This process is repeated in each frame and the chain always corrects itself towards the target. The speed of rotation change per joint depends on the success of the move. If one segment is 'shaking' (the angle is increased in one step and decreased in the next, or vice versa), the joint speed is decreased to provide finer approach to the target. On the other hand, if an angle change is always heading towards the correct direction, the speed is increased until it reaches the maximum value.

This continuous correction sequence has the advantage that the animation looks more natural and human-like compared to applying a transition from the current state of the chain to the solution of the problem. Additionally, it works with moving targets and can avoid collisions with objects that lie between the agent and the target.

Let us consider the example of the pickup action, where an agent has to pick up an object. The agent uses the dummy joint on the left or right hand and tries to equate its position with a point on the surface of the object. This has the effect that the hand seems to touch the object. The next step is to link the object's position with the joint so that the object inherits the joint's transformation and remains attached to the hand. With most of real-world objects, there is a standard way of catching them, and therefore we store this information inside the object declaration in the scene definition file using the *spot* declaration (see paragraph 4.1). To implement the pickup action we employ the Inverse Kinematics engine and use the distance between the dummy joint on the hand and the spot on the target object's surface as an objective function.

In figure 12 we display an agent trying to catch a moving ball with the use of the Inverse Kinematics algorithm.

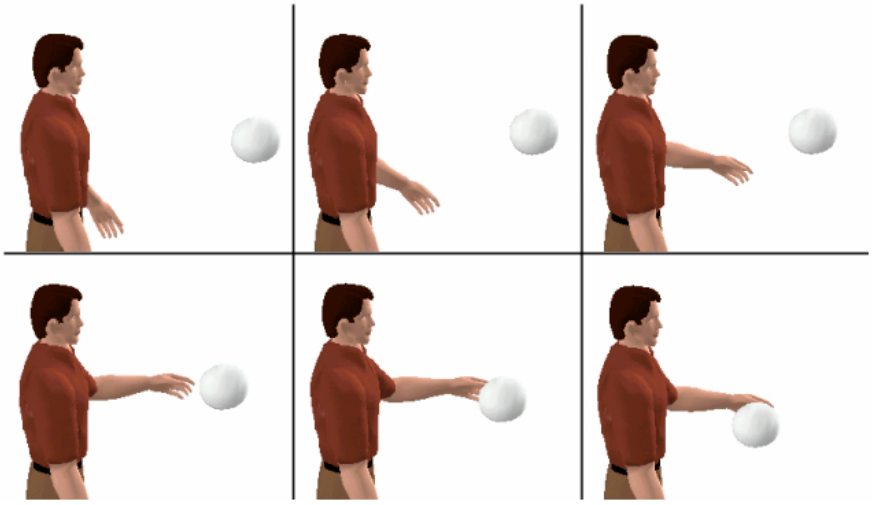

Figure 12: A virtual agent tries to catch a moving ball

5.6. Behavioral Control

So far we have presented a synthetic character with the ability to sense and execute various actions in a 3D environment. Such features may be more than enough for the implementation of user-controlled avatars or for synthetic actors that follow a predefined set of orders. Nevertheless, a virtual agent should demonstrate a more believable and dynamic behavior, and should therefore have an additional mechanism to select the appropriate actions and execute them according to its goals. This is the agent's equivalent of a mind, the part that gives orders to the virtual body.

The agents in SimHuman do not have a fixed method for implementing their behavior, because the way that an agent controls its actions depends much upon the application, and therefore it would not be useful to restrict the decision function to a specific action selection mechanism. The library provides all the necessary tools to the programmer to build a behavioral controller as a callback function, which is then used to control the agent during runtime. These tools are:

Perception: Knowledge about the environment comes from the embedded vision system, which is using ray-casting and the collision detection mechanism. Alternatively one could directly access the properties of objects.

Interaction with the environment: The keyframing mechanism, the inverse kinematics engine and the direct manipulation of virtual objects allow programmers to implement a variety of actions.

The behavioral mechanism could either be implemented with simple if-then rules or with more sophisticated methods, such as the use of a planner.

5.7. Architecture of SimHuman agents

The architecture of the Virtual Agents used in SimHuman is depicted in figure 13. Virtual Agents consist of three different modules. There is a motion controller, which is using the geometry file, the joint hierarchy file and the animation library to generate motion, i.e. to implement the desired actions. For most of the actions, the motion controller needs to be aware of the other objects' position and orientation, and is, therefore, using a memory, which holds this information. The sensing mechanism uses the information it gets from the environment and updates the memory. Finally, there is a behavioral controller, which takes all the high-level decisions of the agent. This module, which is defined by the programmer, decides about the next action that the agent should perform and sends a command (such as "pickup the ball") to the motion controller, which then handles the implementation details.

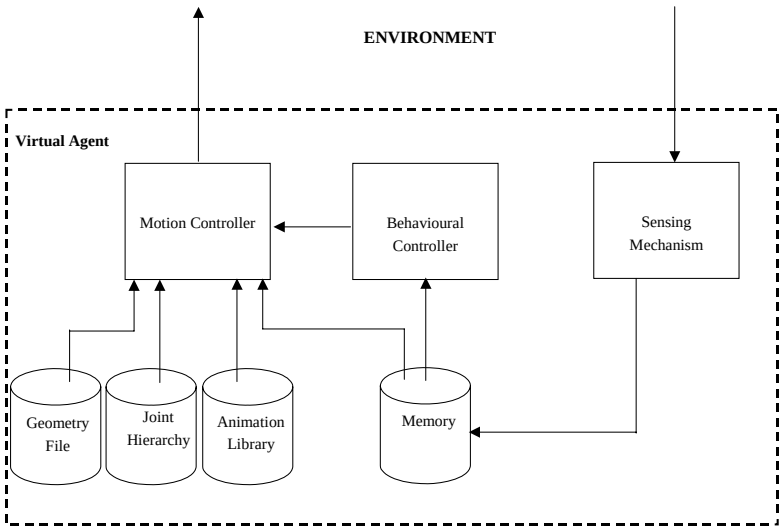

Figure 13: The architecture of SimHuman agents

6. A Simple Example: Virtual University

As an example, we have created a multimedia application for the interactive presentation of the University of Piraeus (Figure 14). It lets the user navigate inside a virtual representation of the university's main building and get information about its places and faculty. The user can navigate freely using the keyboard and the collision detection mechanism prevents him / her from penetrating the walls and objects. Additionally, by clicking on doors, he / she can read about the respective places and offices of the university.

The application also features a virtual agent, who serves as a guide and can assist the user to locate specific areas or offices of members of the staff. The user can ask about an office number or the name of a faculty member, and the virtual agent will walk towards the user and guide him / her to the desired destination. The user's point of view will automatically follow the agent's path. When the destination is reached, the program will display the appropriate information. The agent can also present guided tours in specific departments or entire floors.

The application has been created using the SimHuman library as a basis for the visualization of the 3D environment and the animation of the agent. The university's model has been created based on the building's ground plans, and the agent model has been exported from Curiouslabs Poser. The Virtual University example is an extension of an earlier project [27] that was built with the VRML – Java platform.

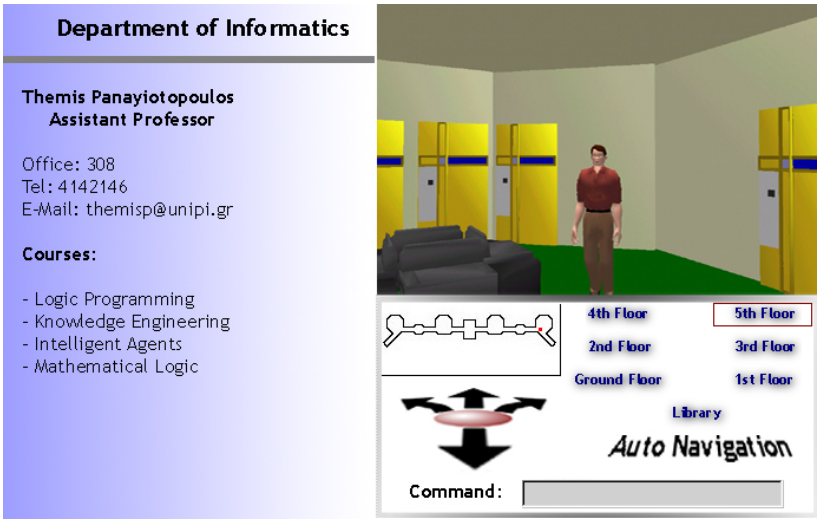

Figure 14: A sample screenshot of the Virtual University example

7. Conclusions and Future Work

In this paper we have presented our approach towards designing and implementing custom virtual environments with agents. We reviewed the literature and explained why we believe that there is a lack of general tools for constructing 3D environments as parts of multimedia applications, educational systems or simple simulations, and presented SimHuman, our tool for designing and building such environments. Virtual Environments and simple Simulation Systems do not need the best possible accuracy; they require natural looking motion, as well as acceptable execution speed on an average PC, and this is the aspect that our approach is focusing on.

The aim of SimHuman is to combine simple yet effective design algorithms and implementation techniques for the creation of synthetic humans for desktop VR applications. One basic advantage of our approach is that it is suited for Windows applications on average PCs, because it is based on the C++ programming language and the OpenGL graphics library. Due to the latter, it is also taking full advantage of the graphics card acceleration, making it possible to render large 3D scenes with a good frame rate. Table 1 shows some typical rendering times for three different scenes with and without physically based modeling. The times have been measured

while a single agent was walking around the scene, using a PIII – 500 processor and a Riva TNT2 graphics card.

	No PBM	PBM
~ 4000 polygons	17 msec	18 msec
~ 10,000 polygons	31 msec	34 msec
~ 25,000 polygons	68 msec	72 msec

Table 1: A table of typical rendering times

Featuring a library, SimHuman can be part of a larger project and be combined with other multimedia features, such as images, sound and video. A very important characteristic of our engine is that it is fully configurable, since it can render a 3D scene using models in VRML format and it can import synthetic humans from Curiouslabs Poser, a commercial program built especially for modeling and animating virtual humans and animals. Consequently one can use any 3D modeler to design the scene, export the objects in VRML format, and use that scene in a SimHuman-based application.

We have already used SimHuman as a tool to present multimedia documents in an educational application [28] and we are planning to extend it as a general platform for scripting interactive presentations. There are, nevertheless, a lot of features that could be added, such as a language to define custom actions, the ability to select between different embedded behavioral architectures, etc. We are currently working on integrating spatio-temporal reasoning and planning techniques into SimHuman's behavioral control and on allowing concurrent execution of agents' actions. Our future plans are to develop a generic visual tool for setting up a scene and describing the agents' behavior to serve as a basis for the automatic generation of virtual environments with believable agents.

Acknowledgements

This work has been supported by the Greek Secretariat of Research and Technology under the PENED'99 project entitled "Executable Intensional Languages and Intelligent Multimedia, Hypermedia and Virtual Reality applications", Contract No. 99ED265.

References

1. C. Dede, M. Salzman, B. Loftin, “The development of a virtual world for learning Newtonian mechanics”, in P. Brusilovsky, N. Streitz (Eds.), Multimedia, Hypermedia and Virtual Reality. Springer Verlag, Berlin, 1996.
2. M. J. Van Gorp, P. Boysen, “ClassNet: Managing the Virtual Classroom”, In WebNet World Conference of the WWW, Internet & Intranet, 1996.
4. The Virtual Human, http://www-unix.mcs.anl.gov/~hudson/htmlbase/vhuman.html
3. I.Varlamis, M. Vazirgiannis, S. Lazaridis, M. Papageorgiou and T. Panayiotopoulos, “Distributed Virtual Reality Authoring Interfaces for the WWW: the VRSHOP case”, Multimedia Tools and Applications, accepted for publication.
5. M. Soto and S. Allongue, “Modeling Methods for Reusable and Interoperable Virtual Entities in Multimedia Virtual Worlds”, Multimedia Tools and Applications, vol. 16, pp. 161-177, 2002.
6. N. Badler, C. Phillips, B. Webber, Simulating Humans: Computer Graphics Animation and Control, Oxford University Press, 1993.
7. D. Terzopoulos, T. Rabie and R. Grzeszczuk, “Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a simulated physical world”, Artificial Life, vol. 1(4), pp. 327 – 351, 1994.
8. R. Aylett, A. Horrobin, J. O’Hare, A. Osman and M. Polshaw, “Virtual Telebubbies: reapplying a robot architecture to virtual agents”, in Proc. of the Third International Conference on Autonomous Agents, New York, 1999, pp. 514-515.
9. N. Badler, “Virtual humans for animation, ergonomics, and simulation”, IEEE Workshop on Non-Rigid and Articulated Motion, Puerto Rico, 1997.
10. J. Shen, D. Thalmann, “Interactive Shape Design Using Metaballs and Splines”, Proc. Implicit Surfaces 1995, Eurographics, Grenoble, France, 1995.
11. D. Thalmann, J. Shen, E. Chauvineau, “Fast Human Body Deformations for Animation and VR Applications”, Proc. Computer Graphics International 96, IEEE Computer Society Press, pp.166-174, 1996.
12. N. Badler, B. Barsky, D. Zeltser, “Making them Move: Mechanics, Control and Animation of Articulated Figures”, Morgan-Kaufmann, 1991.
13. N. Magnetat-Thalmann, S. Carion, M. Courchesne, P. Volino and Y. Wu, “Virtual Clothes, Hair and Skin for Beautiful Top Models”, in Proc. Computer Graphics International '96, Pohang, Korea, 1996, pp. 132-141.
14. J. Hodgins, J. O’Brien, “Computer Animation”, Encyclopedia of Computer Science, 1998.
15. G. Pourazar, “A method to capture and animate the dynamics of human motion”, in Proc. of COMPUGRAPHICS '91, vol. I, pp. 181-197, 1991.
16. D. Brogan, R. Metoyer, and J. Hodgins, “Dynamically Simulated Characters in Virtual Environments”, IEEE Computer Graphics and Applications, vol. 15, no.5, pp. 58-69, 1998.
17. D. Tolani, A. Goswami and N. Badler, “Real-time inverse kinematics techniques for anthropomorphic limbs”, Graphical Models vol. 62, pp. 353-388, 2000.
18. M. Bret, “Virtual Living Beings”, Lecture Notes in Artificial Intelligence, vol. 1834, pp. 119-134, 2000.
19. P. Karla, N. Magnenat-Thalmann, L. Moccozet, G. Sannier, A. Aubel, and D. Thalmann, “Real-time Animation of Realistic Virtual Humans”, IEEE Computer Graphics and Applications, vol.18, no.5, pp. 42-55, 1998.
20. A. Aubel, R. Boulic and D. Thalmann, “Real-time Display of Virtual Humans: Level of Details and Impostors”, IEEE Trans. Circuits and Systems for Video Technology, Special Issue on 3D Video Technology, 2000.
21. R. Boulic, Z. Huang, J. Shen, T. Molet, T. Capin, B. Lintermann, K. Saar, D. Thalmann, N. Magnetat-Thalmann, A. Schmitt, L. Moccozet, P. Kalra and I. Pandzic, “A system for the parallel integrated motion of multiple deformable human characters with collision detection”, Computer Graphics Forum, vol. 13(3), pp. 337-348, 1995.

22. J. Rickel and W.L. Johnson, “Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control”, Applied Artificial Intelligence, vol 13, pp. 343-382, 1999.
23. D. Silva, C. Siebra, J. Valadares, A. Almeida, A. Frery and G. Ramalho, “Personality-Centered Agents for Virtual Computer Games” in Proc. of Virtual Agents 99, Workshop on Intelligent Virtual Agents, Salford, UK, 1999.
24. A. Nareyek, “Intelligent Agents for Computer Games”, in Proc. of the Second International Conference on Computers and Games, 2000.
25. K. Perlin and A. Goldberg, “Improv: A system for scripting interactive actors in virtual worlds”, in Proc. of ACM Computer Graphics Annual Conf, 1996, pp. 205-216.
26. T. Panayiotopoulos, G. Katsirelos, S. Vosinakis and S. Kousidou, “An Intelligent Agent Framework in VRML worlds”, in S. Tzafestas (Ed.), Advances in Intelligent Systems – Concepts, Tools and Applications, pp. 219-230, 1999.
27. T. Panayiotopoulos, N. Zacharis and S. Vosinakis, “Intelligent Guidance in a Virtual University”, in S. Tzafestas (Ed.), Advances in Intelligent Systems – Concepts, Tools and Applications, pp. 33-42, 1999.
28. Belessiotis, S. Vosinakis and T. Panayiotopoulos, “The use of the Virtual Agent SimHuman in the ISM scenario system”, in Proc. of WSES International Conference on Automation and 3Information: Theory and Applications, (AITA '01), Skiathos Island, Greece, 2001, pp. 97-101.