8
Avatar Design

8.1 AVATARS

In vitual worlds, avatars are used to represent human users or autonomous agents. The complexities of avatars can range from the simple form, which may just consists of a group of boxes and spheres, to the most complicated ones, like humanoids with facial animation and sophisticated gestures. Based on their functionalities, avatars can be classified into the following different types:

- *Animated versus Non-animated*: Animated avatars have their own animation data. These animations are usually controlled by using ROUTE semantics to express built-in gestures. Non-animated avatars have no built-in animation data. However they may be controlled by using external facilities (i.e., programs) to make the animation. DLP can be used to control the animation if the body components of the avatars are properly defined by using DEF and those defined nodes have geometrical fields, like position and rotation. In this chapter, we are more interested in the design of non-animated avatars. However, we will discuss how DLP can be used to create and control the gestures of the avatars.

- *Texture-based versus non-texture-based*: Texture-based avatars use textures to cover or present parts of their bodies, especially, the face and clothes. These textures are usually presented by using picture files, like gif or jpeg. Non-texture-based avatars do not use any picture files. They use their own built-in appearance data to present special effects. The file sizes of the texture-based avatars are usually small, for they have embedded picture files. However, they are hard to be controlled by programs. Non-texture-based avatars have their own appearance data, which would increase the file size of the avatars, however, they provide the possibilities for the control from external programs. In this chapter we will discuss how DLP can be used to control the facial expression of non-texture-based avatars.

- *audio-embedded versus non-audio-embedded*: Audio-embedded avatars have their own embedded audio data in their files, whereas non-audio-embedded avatars have not any voice/sound data on them. The file sizes for good quality voice are usually extremely large. They are seldom embedded into avatars and virtual worlds.

- *H-anim compliant versus non-H-anim compliant.* H-anim1.1 is a specification for standard humanoids by the Humanoid animation working group.[H-anim, 2001] As the name implies, H-anim compliant avatars are designed according to the H-anim specification, whereas non-H-anim compliant avatars are not. In this chapter we will focus on the design and the control of H-anim compliant avatars.

8.2 H-ANIM 1.1 SPECIFICATION

As claimed by Humanoid animation working group in [H-anim, 2001], goals of H-anim specification are the creation of libraries of humanoids for reusable in Web-based applications, as well as authoring tools that make it easy to create humanoids and animate them in various ways. H-anim specifies a standard way of representing humanoids in VRML97. This standard will allow humanoids created using authoring tools from one vendor to be animated using tools from another. H-Anim humanoids can be animated using different animation systems and techniques.

An H-Anim file contains a set of Joint nodes that are arranged to form a hierarchy. Each Joint node can contain other Joint nodes, and may also contain a Segment node which describes the body part associated with that joint. Each Segment can also have a number of Site nodes, which define locations relative to the segment. Sites can be used for attaching accessaries, like hat, clothing and jewelry. In addition, they can be used to define eyepoints and viewpoint locations. Each Segment node can have a number of Displacer nodes, that specify which vertices within the segment correspond to a particular feature or configuration of vertices.

The Joint PROTO looks like this:

```
PROTO Joint [
    exposedField     SFVec3f      center             0 0 0
    exposedField     MFNode       children           []
    exposedField     MFFloat      llimit             []
    exposedField     SFRotation   limitOrientation    0 0 1 0
    exposedField     SFString     name                ""
    exposedField     SFRotation   rotation            0 0 1 0
    exposedField     SFVec3f      scale               1 1 1
    exposedField     SFRotation   scaleOrientation    0 0 1 0
    exposedField     MFFloat      stiffness           [ 0 0 0 ]
    exposedField     SFVec3f      translation         0 0 0
    exposedField     MFFloat      ulimit              []
]
```

The meanings of most fields of the Joint PROTO, like scale, translation, are straightforward from the names, however, the following fields need a further explanation

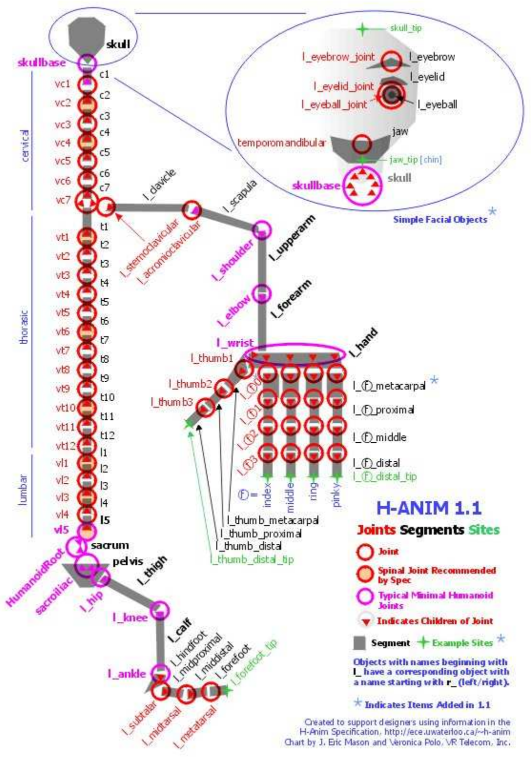
- ulimit and llimit: gives the upper and lower joint rotation limits. The ulimit field defines the maximum values for rotation around the X, Y and Z axes. The llimit field describes the minimum values for rotation around those axes.
- limitOrientation: describes the orientation of the coordinate frame in which the ulimit and llimit values are to be interpreted. This field specifies the orientation of a local coordinate frame, relative to the Joint center position described by the center exposedField.
- stiffness: specifies values ranging between 0.0 and 1.0 which give the inverse kinematics system hints about the ”willingness” of a joint to move in a particular degree of freedom.

The segment PROTO look like this:

```
PROTO Segment [
    field          SFVec3f     bboxCenter       0 0 0
    field          SFVec3f     bboxSize         -1 -1 -1
    exposedField   SFVec3f     centerOfMass     0 0 0
    exposedField   MFNode      children         [ ]
    exposedField   SFNode      coord            NULL
    exposedField   MFNode      displacers       [ ]
    exposedField   SFFloat     mass             0
    exposedField   MFFloat     momentsOfInertia [ 0 0 0 0 0 0 0 0 0 ]
    exposedField   SFString    name             ""
    eventIn        MFNode      addChildren
    eventIn        MFNode      removeChildren
]
```

An explanation on some of the fields above:

- mass: the total mass of the segment, however, it is usually not necessary.
- centerOfMass: the location within the segment of its center of mass.



skull
skullbase
H-ANIM 1.1
Joints Segments Sites
Simple Facial Objects
l_shoulder
l_elbow
l_wrist
l_hip
l_knee
l_ankle
HumanoidRoot
sacroiliac
Created to support designers using information in the H-Anim Specification. http://ece.uwaterloo.ca/~h-anim
Chart by J. Eric Mason and Veronica Polo, VR Telecom, Inc.

Fig. 8.1 A Standard Joints/Segment Diagram of H-anim 1.1

- momentsOfInertia: the moment of inertia matrix. The first three elements are the first row of the 3x3 matrix, the next three elements are the second row, and the final three elements are the third row.

A standard joints/segment diagram of H-anim 1.1 specification is shown in Figure 8.2

In H-anim specification, site nodes are designed for the following three purposes: First, it can be used to be an "end effector" location for an inverse kinematics system. Next, it defines an attachment point for accessories such as hat and clothing. Third, it provides a location for a virtual camera in the reference frame of a Segment (such as a view "through the eyes" of the humanoid for use in multi-user worlds).

Sites are located within the children exposedField of a Segment node. The children field of a site node is used to store any accessories that can be attached to the segment.

The Site PROTO looks like this:

```
PROTO Site [
    exposedField    SFVec3f     center           0 0 0
    exposedField    MFNode      children         []
    exposedField    SFString    name             ""

    exposedField    SFRotation  rotation         0 0 1 0
    exposedField    SFVec3f     scale            1 1 1
    exposedField    SFRotation  scaleOrientation 0 0 1 0
    exposedField    SFVec3f     translation      0 0 0
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
]
```

According to H-anim specification, if used as an end effector, the Site node should use the following consisting naming system: the name of the site should be with with the name of the segment which attached, like a "_tip" suffix appended. The end effector Site on the right index finger should have a name like "r_index_distal_tip", and the Site node would be a child of the "r_index_distal" Segment. Sites that are used to define camera locations should have a "_view" suffix appended. Sites that are not end effectors and not camera locations should have a "_pt" suffix. Sites that are required by an application but are not defined in this specification should be prefixed with "x_".

Sometimes the application may want to identify some specific vertices within a Segment. That would require a Displace to store the hint message. The Displacers for a particular Segment are stored in the displacers field of that Segment.

The Displacer PROTO looks like this:

```
PROTO Displacer [
    exposedField MFInt32  coordIndex     [ ]
    exposedField MFVec3f  displacements  [ ]
    exposedField SFString name           ""
]
```

The coordIndex field shows the indices into the coordinate array for the Segment of the vertices that are affected by the displacer. The displacements field describes a set of 3D values that should be added to the neutral or resting position of each of the vertices referenced in the coordIndex field of the Segment. These values correspond one to one with the values in the coordIndex array.

The H-anim file also has a single Humanoid node which stores human-readable data about the humanoid such as author and copyright information. That also stores additional information about all the Joint, Segment and Site nodes, and serves as a "wrapper" for the humanoid. Of course, it is used to describe the top-level Transform for positioning the humanoid in virtual worlds.

```
PROTO Humanoid [
    field           SFVec3f    bboxCenter            0 0 0
    field           SFVec3f    bboxSize              -1 -1 -1
    exposedField    SFVec3f    center                0 0 0
    exposedField    MFNode     humanoidBody          [ ]
    exposedField    MFString   info                  [ ]
    exposedField    MFNode     joints                [ ]
    exposedField    SFString   name                  ""
    exposedField    SFRotation rotation              0 0 1 0
    exposedField    SFVec3f    scale                 1 1 1
    exposedField    MFNode     segments              [ ]
    exposedField    MFNode     sites                 [ ]
    exposedField    SFVec3f    translation           0 0 0
    exposedField    SFString   version               "1.1"
    exposedField    MFNode     viewpoints            [ ]
]
```

8.3 CREATING H-ANIM COMPLIANT AVATARS

Based on H-anim 1.1 specification, we can design any humanoid with different body geometrical data and different levels of articulation, which can be anything we like. The appendix of the H-anim 1.1 specification also provides a suggest on the body dimension and three levels of articulation. The level of articulation zero is the minimum legal H-Anim humanoid, with the node HumanoidRoot and several default site translations. The level of articulation one is a typical low-end real-time 3D hierarchy. The level of articulation two is a more complex one, a body with simplified spine. Take the VRML files of the level of articulation, which are available from the H-anim web site, as templates for building H-anim 1.1 compliant avatars, we can design our own ones.

Here are some examples: First we can design a simple H-anim 1.1 compliant avatar, which just use simple geometrical data, like box, or sphere, to be the body parts, which is shown in Figure 8.3.

For the convenience of the test on the construction, we can design each body part as a seperated VRML file. For instances, the left-hand can be like this:

```
\#VRML V2.0 utf8

DEF hanim_l_hand Transform {
            translation 0.15 0.7 -0.025
            rotation 0 0 1  0
               children [
                     DEF hanim_l_hand_shape Shape {
```

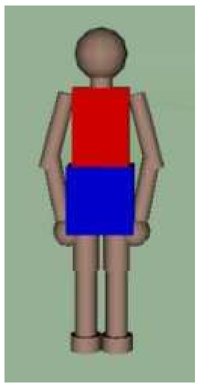


Fig. 8.2 A Simple H-anim 1.1 Compliant Avatar

```
appearance Appearance {
material  Material {
ambientIntensity 0.200
shininess 0.200
diffuseColor 0.76863 0.61961 0.54902
emissiveColor 0.0 0.0 0.0
specularColor 0.0 0.0 0.0
        }}
    geometry Sphere {
       radius 0.07
      } } ]}
```

Namely, the left-hand (without the left forearm and left upper-arm, which would be defined in a separated file), is just a sphere with a defined name ”hanim_l_hand”. Moreover, we need a translation field and a rotation field to put the body-part to an appropriate position. Suppose that this VRML file is saved with the file name ”l_hand0.wrl”. We can add it into the left-hand segment in the template file like this:

```
......
children [
               DEF hanim_l_hand Segment {
               name         "l_hand"
               children [

            Inline { url "l_hand0.wrl"}
.....
```

We use *Inline* to add the VRML file into the avatar file, however, note that VRML EAI does not support any referred nodes which is defined in Inline files. If we want to control the defined nodes in DLP, we should add the corresponding lines into the avatar file. Similarly, we can define other body parts, like forearm, thigh, upper-arm, skull, calf, etc.

In order to obtain more realistic humanoid avatars, we need more sophisticated geometrical data on the body parts and some necessary accessories, like hairs, clothes, etc. The upper body and the clothes are normally located at the joint *vl*5. Suppose that the corresponding VRML file is stored as "l5.wrl". Add the data of upper body and the clothes can be like this:

```
......
DEF hanim_l5 Segment {
       name           "l5"
       children [
         DEF hanim_l5 Inline { url "l5.wrl" }
......
```

Moreover, the hairs should be located at the skull_tip site of the skull base joint as follows:

```
.....
 DEF hanim_vl5 Joint {
       name           "vl5"
      center         0.0028 1.0568 -0.0776
       children [
       DEF hanim_skullbase Joint {
         name         "skullbase"
         center       0.0044 1.6209 0.0236
         children [
          DEF hanim_skull Segment {
           name         "skull"
          children [
          DEF hanim_skull
            Inline { url "skull.wrl" }

            DEF hanim_skull_tip Site {
             name         "skull_tip"
             translation    0.0050 1.7504 0.0055
             children [
            Inline { url "hair.wrl" }
             ]
          }
```

A H-anim 1.1 compliant avatar with hairs and clothes is shown in Figure 8.3.

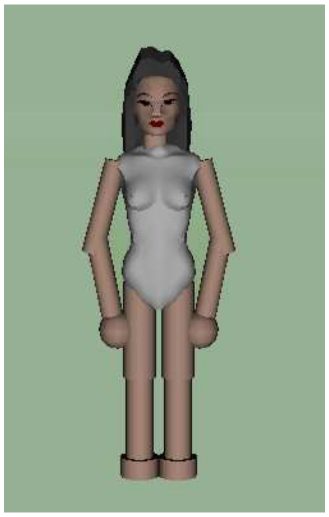


Fig. 8.3 A H-anim 1.1 Compliant Avatar with Hairs and Clothes

8.4 AVATAR AUTHORING TOOLS

8.4.1 Curious Labs Poser 4

Poser 4 by Curious Labs is a 3D-character animation and design tool for avatars design. [Curious Labs] The program provides plentiful libraries of pose settings, facial expressions, hand gestures, and swappable clothing. Users can create images, movies, and posed 3D figures from a diverse collection of fully articulated 3D human and animal models. More usefully, poser 4 supports the export of avatar files with VRML format or H-anim format. However, the sizes of the exported files from poser are usually too large, say, 2 MB or more, which would be a problem for any significant use over the Web. A screenshot of Poser 4 is shown in Figure 8.4.1.

8.4.2 Blaxxun Avatar Studio

Blaxxun Avatar Studio is a tool for design of animated and texture-based VRML avatars. Blaxxun Avatar studio has not yet support of the export of the avatars file with H-anim compliant format. The avatars designed by Avatar Studio are non H-anim compliant ones. Using Avatar Studio, avatars can be designed with large range of body sizes, individual proportions, skin, hair and eye color. Once the avatar's basic properties are determined, she/he can be dressed from a large wardrobe and furnished with an extensive selection

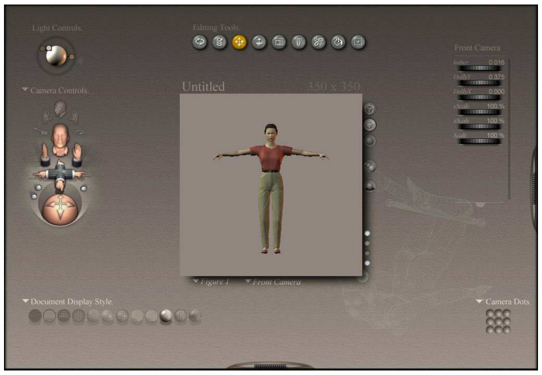


Fig. 8.4 A Screenshot of Poser 4

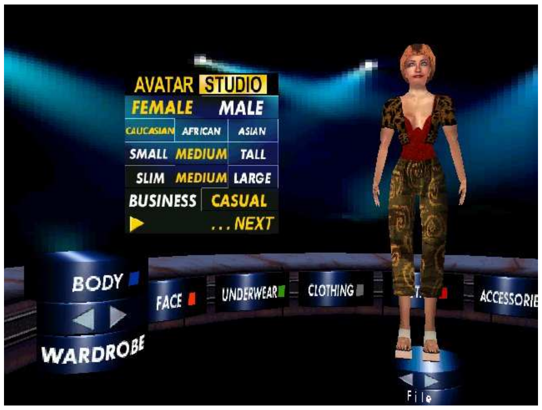


Fig. 8.5 A Screenshot of Avatar Studio

of accessories, like sunglasses or handbags. Avatar Studio also supports the "animations editor," a tool for creating gestures and movements which are then assigned to certain key-words. The typical keys for the gestures are: hello, hey, yes, smile, frown, no, and bye. A screenshot of Blaxxun Avatar Studio is shown in Figure 8.4.2. Avatars designed by Blaxxun Avatar Studio are texture-based ones. Therefore, the faces and clothes of the avatars can be changed by editing the corresponding texture files. For example, Figure 8.4.2 shows the soccer player avatar "blue2", whose texture file for the face and the clothes is shown in Figure 8.4.2.

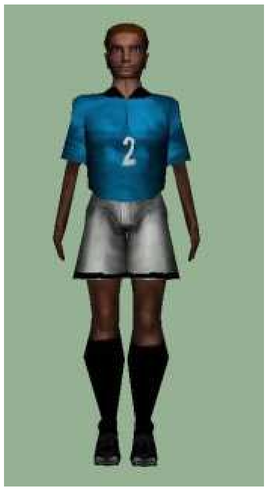


Fig. 8.6 Soccer Player Avatar blue2

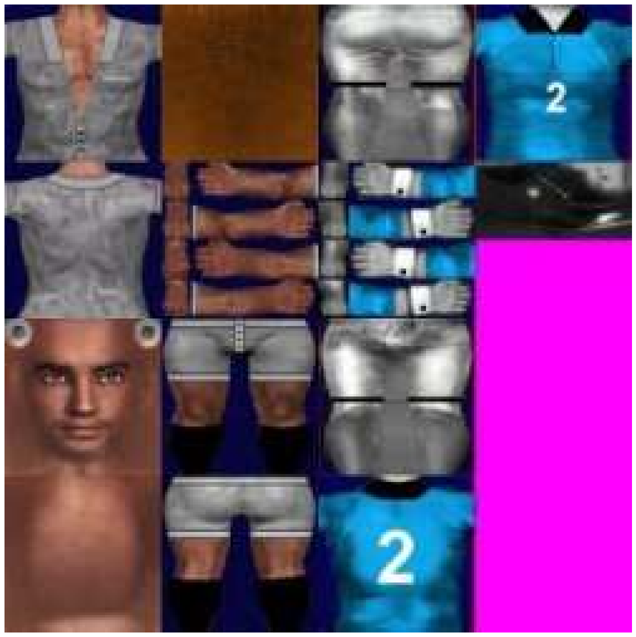


Fig. 8.7 Texture of Soccer Player Avatar blue2

8.5 AVATAR ANIMATION CONTROL IN DLP

Humanoid avatars can be controlled by using the get/set-predicates in DLP, like those are shown in WASP soccer games. They can move to certain positions by the set-position-predicates, or turn to certain orientation by the set-rotation-predicates, on those humanoid avatars. These humanoid avatars can be built based the H-anim specification. The avatar animation can be achieved by setting the positions/rotations of the body parts of the humanoid avatars with different time intervals.

Consider a humanoid avatar which is based on H-anim specification. Turning the left arm of the avatar to front can be realized by simplely setting the rotation of the left shoulder joint *hanim_l_shoulder* to $\langle 1, 0, 0, 1.57 \rangle$ as follows,

```
setRotation(hanim_l_shoulder, 1, 0, 0, 1.57)
```

However, in order to achieve a smooth change of movement of the body part, we have to introduce several interpolations between two rotations with certain time interval control. Turning Object to a rotation $\langle X, Y, Z, R \rangle$ within $Time$ milliseconds with I interpolation can be achieved by the following DLP program[1]:

```
turn_object(Object, rotation(X,Y,Z,R), Time, I):-
            getRotation(Object,X1,Y1,Z1,R1),
            count := 0,
            incrementr := (R-R1)/I,
            incrementx := (X-X1)/I,
            incrementy := (Y-Y1)/I,
            incrementz := (Z-Z1)/I,
            sleeptime := Time*1000/I,
            repeat,
                 Rnew is R1+incrementr*(count+1),
                 Xnew is X1+incrementx*(count+1),
                 Ynew is Y1+incrementy*(count+1),
                 Znew is Z1+incrementz*(count+1),
                 setRotation(Object,Xnew,Ynew,Znew,Rnew),
                 sleep(sleeptime),
                 ++count,
            abs(Rnew-R) =< abs(incrementr),
            setRotation(Object,X,Y,Z,R).
```

Animation of avatars usually involves the movements of several body parts simultaneously. That would make the maintenance of the timing and sycroniza-

[1]In order to achieve a natural transition between two rotations, we need the slerp interpolation on quatenions, which is explained the section 9.6.3

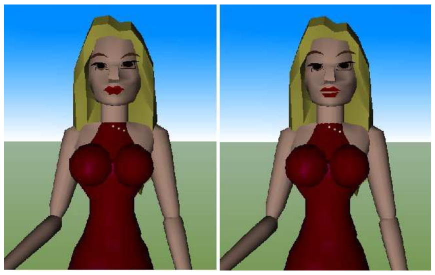


Fig. 8.8 Facial Animation

tion of the multiple threads which controls the body movements more complicated. In Chapter 9 we will discuss a scripting language which can be used to simplize the control of the animation of humanoid avatars.

One of the important issues of the avatar animation is the facial expression. H-anim specification adopts the facial animation parameters (FAP), which are first proposed in MPEG4. The following is a simple example which shows how some facial expression, like eyebrow movement and smile can be realized by basing on some ad hoc facial geometrical data. The facial expression is shown in Figure 8.8.

```
eyebrow_move(l,Range):-
getSFVec3f(l_eyebrow,translation,X,Y,Z),
Y1 is Y + Range,
setSFVec3f(l_eyebrow,translation,X,Y1,Z).
eyebrow_move(r,Range):-
getSFVec3f(r_eyebrow,translation,X,Y,Z),
Y1 is Y + Range,
setSFVec3f(r_eyebrow,translation,X,Y1,Z).


smiling(Time):-
smile_point(2, Point2),
setMFVec3f(lower_lip_coordinate, point, Point2),
sleep(Time),
smile_point(1, Point1),
setMFVec3f(lower_lip_coordinate, point, Point1).

smile_point(1,SmilePoint) :-
SmilePoint = [[0.0381, 0.0312, -5.0E-4],
   [0.015, 0.0162, -0.0204],
[0.015, 0.03, 0.0],
[0.0215, 1.0E-4, -8.0E-4],
 [-0.015, 0.0162, -0.0204],
[-0.0381, 0.0312, -5.0E-4],
 [-0.015, 0.03, 0.0],
[-0.0215, 1.0E-4, -8.0E-4]].

smile_point(2,SmilePoint):-
SmilePoint = [[0.04810,0.03920,-0.00050],
[0.01500,0.00020,-0.02040],
[0.01500,0.01400,0.00000],
[0.02150,-0.01610,-0.00080],
[-0.01500,0.00020,-0.02040],
[-0.04810,0.03920,-0.00050],
[-0.01500,0.01400,0.00000],
[-0.02150,-0.01610,-0.00080]].
```

Exercises

8.1 Design a H-anim 1.1 compliant avatar for the soccer playing agents. The avatar should be able to be controlled in DLP to show the following gestures: kicking, greeting, shouting, and ball-holding.

8.2 Design a texture-based humannoid avatar, by using your own photo as the texture of the avatar's face.

8.3 Extend the facial animation example with more facial expressions.