

Humanóide, à la VRML

Artur Marques
arturmarques@mail.telepac.pt

Carlos Ferreira
cpdf@isr.ist.utl.pt

Miguel Antunes
Miguel.Antunes@inesc.pt

João Pereira
jap@inesc.pt

Sumario

Os ambientes virtuais multi-utilizador têm vindo a aumentar de popularidade. Tem sido evidente o esforço para o desenvolvimento de ferramentas que permitam a criação desses ambientes com o maior grau de realismo. Uma característica comum nesses ambientes é a representação de cada utilizador por um objecto denominado avatar, possuindo na maior parte das vezes uma forma humanóide. A utilização do VRML para a criação e manipulação desse tipo de objectos, e a especificação Hanim 1.1 constituíram os fundamentos do trabalho realizado.

1 Introdução

Este trabalho foi desenvolvido no âmbito da cadeira de Realidade Virtual do curso de mestrado de Engenharia Electrotécnica e Computadores no Instituto Superior Técnico. Trata-se de um mundo 3D em VRML onde o utilizador pode navegar utilizando um Avatar implementado segundo a especificação *Hanim* [1].

Os ambientes virtuais multi-utilizador têm vindo a aumentar de popularidade, não só por parte de potenciais utilizadores, mas também na própria comunidade científica. Têm sido feitos grandes esforços com o intuito de desenvolver as técnicas e tecnologias que permitem o desenvolvimento de ambientes virtuais que suportam um número de utilizadores cada vez maior, bem como um maior realismo, (por exemplo, através da utilização de imagens 3D, som e vídeo em tempo real). Existem já vários sistemas (plataformas) que permitem o desenvolvimento deste tipo de aplicações, por exemplo: *Dive*[2], *Massive*[3], *ActiveWorlds*[4], *Blaxun*[5]. Sendo alguns desses sistemas comerciais. Todos estes sistemas suportam a representação 3D dos mundos e dos objectos neles contidos, sendo o mundo, na maior parte das vezes, modelado utilizando formatos proprietários, apesar de alguns deles permitirem a importação de VRML. Uma característica comum a estes ambientes é que cada utilizador é representado no mundo por um objecto denominado *avatar*, possuindo na maior parte das vezes uma forma humanóide. O utilizador navega no mundo controlando o seu *avatar*, e tem consciência da presença de outros utilizadores através da visualização dos respectivos *avatars*. Em alguns sistemas os *avatars* são estáticos enquanto noutros possuem animações que o utilizador pode activar enquanto navega (animação para andar, correr) ou com o objectivo de comunicar com outros utilizadores (saudar, acenar, etc.). O objectivo deste trabalho foi o de verificar até que ponto a especificação actual do VRML suporta o desenvolvimento de *avatars*, principalmente no aspecto do controle dos mesmos por parte dos utilizadores.

2 Avatar

A implementação do Avatar baseou-se na especificação *Hanim 1.1 (Humanoid animation)* para modelar o corpo de um humanóide. A escolha desta especificação deveu-se ao facto de a mesma definir uma forma padrão para representar humanóides, e de ser de implementação bastante acessível e flexível. Nesta secção a especificação e sua implementação são brevemente descritas.

corpo. Quanto mais articulações forem implementadas, mais articulado será o humanóide o que permitirá uma maior mobilidade das várias partes do corpo. Foi esta simplicidade de manipulação que levou à sua utilização neste trabalho.

2.2 Implementação do Avatar

2.2.1 Nó Joint

Cada articulação no corpo é representada por um nó *Joint* que é utilizado para definir relações entre segmentos do corpo. A definição do nó *PROTO Joint* é a seguinte:

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

O campo *name* serve para identificar as articulações de forma a que as aplicações possam manipulá-las em *run-time*.

Um nó *Joint* também é utilizado para conter informação específica das articulações. Essa informação pode ser utilizada por aplicações para animar o humanóide. Os campos comentados servem para conter informação acerca de constrangimentos de rotação que uma dada articulação pode ter. Esses campos estão comentados uma vez que não fazem parte da implementação. Os restantes campos possuem o mesmo significado que os campos de um nó *Transform* tendo sido esse o nó utilizado para implementar o *PROTO Joint*. No entanto convém notar que o campo *center* contém a posição do centro de rotação da articulação, relativa à raiz do da descrição do corpo do humanóide. E como as localizações dos centros das articulações estão todas no mesmo sistema de coordenadas, o comprimento de cada segmento pode ser determinado calculando a distancia entre o centro do nó *Joint* pai e o centro do nó *Joint* filho. A única excepção será para os segmentos terminais como as pontas dos dedos das mão e dos pés, por exemplo. As localizações dos centros das articulações utilizadas nesta implementação estão de acordo com o que é aconselhado na especificação.

2.2.2 Nó Segment

Cada segmento do corpo é guardado num nó *Segment*. Segundo a especificação um nó *Segment* deverá ser tipicamente implementado utilizando um nó *Group* de forma a conter um número de figuras ou de transformações que posicionem a parte do corpo correspondente dentro do sistema de coordenadas do corpo. No entanto e uma vez que o avatar tem a sua dimensão pré-definida pelas posições dos centros das articulações implementadas, e pretendíamos que o aspecto do avatar pudesse ser facilmente alterado, optou-se por implementar o *PROTO Segment* utilizando um nó *Transform*. Desta forma a nossa implementação do humanóide terá à partida todas as articulações e segmentos devidamente posicionados. Note-se no entanto que a dimensão das figuras associadas aos segmentos terá de estar de acordo com as dimensões definidas pelas distâncias dos centros das articulações associadas a esse segmento. Para que se possa alterar facilmente o aspecto do avatar todos os segmentos têm como «filho» um nó *Inline* que deve conter a figura da parte do corpo correspondente. Esses nós *Inline* são parametrizados no *PROTO Avatar* descrito em 4.2.4.

```
PROTO Segment {
  exposedField SFString name ""
# exposedField SFFloat mass 0
# exposedField SFVec3f centerOfMass 0 0 0
# exposedField SFNode coord NULL
# exposedField MFNode displacers []
}
```

```

exposedField SFVec3f bboxCenter 0 0 0
exposedField SFVec3f bboxSize -1 -1 -1
exposedField MFNode children [
]
exposedField SFVec3f translation 0 0 0
exposedField SFVec3f scale 1 1 1
exposedField SFRotation rotation 0 0 1 0
exposedField SFRotation scaleOrientation 0 0 1 0
# eventIn MFNode addChildren
# eventIn MFNode removeChildren
}
{
Transform {
translation IS translation
rotation IS rotation
scaleOrientation IS scaleOrientation
scale IS scale
}
children IS children
}
}

```

Tal como no *PROTO JOINT* só foram implementados alguns dos campos descritos na especificação, uma vez mais, os campos não utilizados estão comentados.

2.2.3 Nó Humanoid

O nó *Humanoid* é utilizado para guardar informação legível como os dados dos autores, guardar as referências para as articulações e segmentos, servindo como contentor para todo o humanóide. Também oferece uma forma de se mover o humanóide através dos mundos.

```

PROTO Humanoid [
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 SFRotation scaleOrientation 0 0 1 0
exposedField MFNode segments [ ]
# exposedField MFNode sites [ ]
exposedField SFVec3f translation 0 0 0
exposedField SFString version "1.1"
exposedField MFNode viewpoints [ ]
field SFVec3f bboxCenter 0 0 0
field SFVec3f bboxSize -1 -1 -1
]
{
Transform {
center IS center
rotation IS rotation
scale IS scale
scaleOrientation IS scaleOrientation
translation IS translation
children [
Group {
children IS humanoidBody
}
Group {
children IS viewpoints
}
]
}
}

```

O campo *humanoidBody* serve para conter o nó *HumanoidRoot*, que é o nó articulação raiz de toda a hierarquia de articulações. Os campos *joints* e *segments* servem para conter referências (i.e *USES*) para os nós *Joint* e *Segment* do humanóide. O campo *viewpoints* serve para conter referências para os nós *Viewpoints* que possam estar associados a partes do corpo do humanóide. Estes nós podem ser utilizados para que o utilizador possa observar as animações de vários pontos de vista, por exemplo. Na implementação realizada os únicos nós *Viewpoint* são referidos em 4.5. No entanto esses nós não podem ser inseridos no campo *viewpoints* do humanóide porque todos os nós inseridos nesse campo estão sujeitos a todas as transformações efectuadas no humanóide, facto esse que interferiria com os mecanismos utilizados para controlar o avatar.

2.2.4 Nó Avatar

Como já foi referido anteriormente o nosso objectivo era definir um avatar (humanóide) que o utilizador pudesse controlar e mover por um mundo. A utilização da especificação *Hanim* permitiu definir um humanóide capaz de ser movido, com animações associadas. No entanto, para que o utilizador pudesse controlar o avatar, foi necessário definir mais alguns nós. A especificação *Hanim* não aborda aspectos de controle dos humanóides, no entanto, a forma como estão definidos os nós *PROTO* permite desenvolver aplicações que manipulem em *run-time* os nós do humanóide. Essas aplicações deverão recorrer, muito possivelmente, a interfaces oferecidas pelos browsers (EAI[7], por exemplo). Mas pretendia-se controlar o

avatar utilizando simplesmente mecanismos do VRML97, pelo que foi definido um novo nó *Avatar*, com o objectivo de estender o nó *Humanóide*. A definição do nó avatar é a seguinte:

```
#Segmentos do Avatar.
field MFString cabeca "cabeca/cabeca.wrl"
field MFString pescoco "pescoco/pescoco.wrl"
field MFString tronco "tronco/tronco.wrl"
field MFString ant_braco_dir "antbraco_dir/antbraco.wrl"
field MFString braco_dir "braco_dir/braco.wrl"
field MFString mao_dir "mao_dir/mao.wrl"
field MFString ant_braco_esq "antbraco_esq/antbraco.wrl"
field MFString braco_esq "braco_esq/braco.wrl"
field MFString mao_esq "mao_esq/mao.wrl"
field MFString pelvis "pelvis/pelvis.wrl"
field MFString coxa_dir "coxa_dir/coxa.wrl"
field MFString tibia_dir "tibia_dir/tibia.wrl"
field MFString pe_dir "pe_dir/pe.wrl"
field MFString coxa_esq "coxa_esq/coxa.wrl"
field MFString tibia_esq "tibia_esq/tibia.wrl"
field MFString pe_esq "pe_esq/pe.wrl"
}
{
Group{
children[
DEF TheAvatar Humanoid {
HumanoidBoyd[ # Arvore de Joints ... ]
}
# Cameras
# ...
#No AvatarControl
# ...
#Animacoes
# ...
#ROUTES
# ...
}
}
```

As figuras associadas aos nós segmentos do humanóide são definidas recorrendo a nós *Inline*. A definição do humanóide propriamente dito é feita dentro do campo *humanoidBody*, do nó *Humanoid*. Para cada segmento do humanóide implementado existe um campo definido no prototipo, que serve para guardar o *URL* do ficheiro que define a parte do corpo correspondente. Os nós *Segment* são criados utilizando nós *Inline* em que o *URL* possui o valor do campo correspondente a parte do corpo. Por exemplo o nó para a coxa direita é definido da seguinte forma:

```
DEF hanim_r_thigh Segment {
name "r_thigh"
translation -.125 .7 0
children [
Inline {
url IS coxa_dir
}
]
}
```

Desta forma é possível ao criar nós do tipo *Avatar* especificar as formas de todas as partes do corpo do avatar.

O nó *Avatar* não serve somente para permitir criar avatars com aspectos diferente - é no nó avatar que são definidos todos os comportamentos do avatar e toda a lógica de controle do mesmo. Alguns desses aspectos são descritos nas secções seguintes.

2.3 Animações

Devido a se ter utilizado a especificação *Hanim*, a implementação de animações para o avatar tornou-se uma tarefa bastante simplificada. Uma vez definidas as articulações, animar o avatar é uma questão de efectuar rotações às articulações associadas às partes do corpo que se pretende mover. Por exemplo, se efectuar uma rotação no eixo dos *XX* com um angulo negativo à articulação do ombro do braço direito, o avatar irá levantar o braço direito esticado.

Foram implementadas várias animações que podem ser activadas pelo utilizador :

- **Andar** – Esta animação simula o andamento do avatar. Note-se que esta animação só afecta o corpo do avatar, e não a sua posição. O controle da posição é feito pelo script *AvatarControl*.
- **Correr** – Esta animação simula o andamento do avatar em modo corrida. Tal como a animação andar, esta animação só afecta o corpo do avatar.

- **Sentar** – Esta animação coloca o avatar na posição sentado. O utilizador pode accionar esta animação para se sentar em cadeiras ou sofás existentes no mundo. Note-se no entanto que a animação não depende da existência desses objectos.
- **Levantar** – Esta animação coloca o avatar de pé.
- **Adeus** – O avatar vira-se para o utilizador e diz adeus.
- **Não** – O avatar diz não. Esta animação é utilizada quando o avatar está sentado e o utilizador tenta mover-se. Pretende indicar ao utilizador que este (o avatar) deve-se levantar antes de andar.

2.4 Barra de Comandos

Para o utilizador poder controlar o(s) comportamento(s) do avatar foram definidos dois novos nós: *CommandMenu* e *Command*. Estes nós permitem definir menus de comandos para controlar o avatar.

2.4.1 Nó *Command*

Cada no comando representa um comando que o utilizador pode activar. O utilizador activa um comando premindo o rato sobre o nó.

```
PROTO Command {
  field MFString name ""
  field SFVec3f position 0 0 0
  field SFNode active_appearance Appearance { ... }
  field SFNode inactive_appearance Appearance { ... }
  eventOut SFTIME click_time
  eventOut SFBool click
}
```

O campo *name* define o nome do comando que é mostrado ao utilizador. Os campos *active_appearance* e *inactive_appearance* representam o aspecto do botão quando activo e inactivo, respectivamente. Um comando está activo enquanto o utilizador mantiver o rato sobre o comando. Quando o utilizador prime o botão do rato sobre o comando são lançados dois eventos: *click_time* e *click*. O primeiro contém o tempo em que o utilizador premiu o comando e pode ser utilizado para iniciar animações (fazendo ROUTE para nós *TimeSensor*, por exemplo); o segundo informa simplesmente que o comando foi activado. O nó *Command* é implementado utilizando um nó *TouchSensor* para detectar as acções do utilizador, uma *shape* de texto para mostrar o nome do comando ao utilizador e um nó *Script* para alterar o aspecto do texto consoante o utilizador activa ou desactiva o comando.

2.4.2 Nó *CommandMenu*

O nó *CommandMenu* serve para agrupar comandos num menu. O campo *commands* serve para indicar os comandos que fazem parte do menu e o campo *position* indica a posição da barra de comandos. Como se pode ver na implementação do protótipo os comandos são agrupados utilizando um nó *Billboard* desta forma os comandos estão sempre «virados» para o utilizador desde que o eixo de rotação do utilizador seja o eixo dos *YY*. A escolha do eixo de rotação prende-se com o facto da barra de comandos ser tipicamente colocada ao lado do avatar.

```
PROTO CommandMenu {
  field MFNode commands []
  field SFVec3f position 0 0 0
}
{
  Transform {
    translation IS position
    children [
      Billboard {
        axisOfRotation 0 1 0
        children IS comands
      }
    ]
  }
}
```

Na implementação do avatar foram definidos dois menus *menu1* e *menu2*. O primeiro é utilizado quando o utilizador tem controle sobre o avatar e permite que o utilizador possa mudar do modo correr para o modo andar, e colocar o avatar na posição sentado ou de pé. O segundo é utilizado quando o utilizador não controla o avatar, e permite que o utilizador visualize todas as animações do avatar. Para que só seja mostrado um menu de cada vez, foi utilizado um nó *Switch* que é controlado pelo script *AvatarScript* de forma a aparecer o *menu1* ou *menu2* consoante o utilizador controla ou não o avatar.



Figura 2. (em cima) Barra de comandos com o avatar sobre controle do utilizador. (em baixo) Barra de comandos quando o utilizador não controla o avatar.

2.5 Controle do Avatar

O objectivo da implementação do avatar foi, não só exercitar a especificação *Hanim*, mas também definir um avatar que o utilizador possa controlar e utilizar, para navegar no mundo. Além disso optou-se por apenas utilizar VRML, aceitando as limitações existentes.

O avatar é controlado através dos movimentos do utilizador, isto é, o avatar efectua exactamente todos os movimentos que o utilizador, mantendo-se sempre à frente e de costas para este a uma distância de aproximadamente 3 unidades. Para controlar o avatar foi necessário definir diversos nós:

2 nós Viewpoint – Estes nós foram utilizados para definir as duas câmaras que o utilizador pode utilizar: *UserView* e *NormalView*. A primeira na qual o utilizador controla o avatar e que se desloca com ele; a segunda com a qual o utilizador pode explorar o mundo sem ter o controlo do avatar.

Um nó ProximitySensor – Para detectar os movimentos do utilizador que por sua vez são traduzidos em movimentos do avatar pelo script *AvatarScript*. O sensor está activo sempre que o *viewpoint UserView* está activo e o seu centro é alterado de forma a ser igual à posição actual do avatar. Dessa forma evita-se que o utilizador saia do alcance do sensor e perca controle do avatar.

Um nó PositionInterpolator - Este nó é utilizado para interpolar o deslocamento do avatar à medida que o utilizador se movimenta.

Um nó Script – O script *AvatarScript* controla diversos aspectos do comportamento do avatar, entre eles controla se o avatar está a correr, controla a posição das câmaras associadas ao avatar, o movimento do avatar, etc.

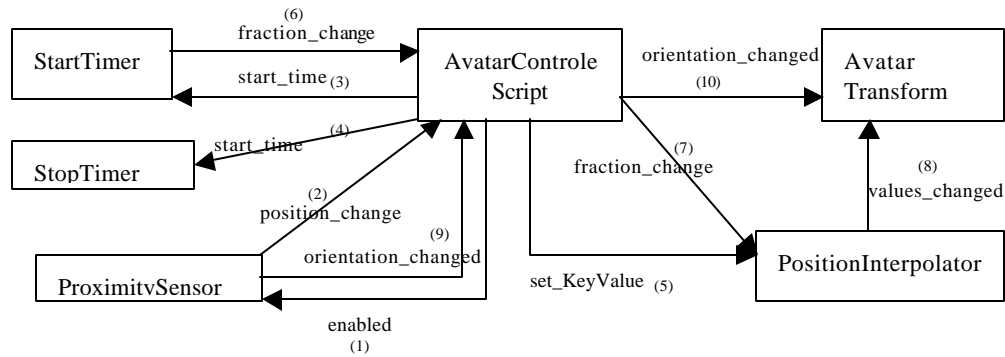


Figura 3 – Nós VRML para o controlo do movimento do Avatar

A Figura 3 mostra os principais nós utilizados no controlo do *avatar* bem como algumas das ligações (*ROUTES*) existentes entre eles. A figura mostra ainda a sequência de eventos que ocorre quando um utilizador se movimenta.

Os nós *StartTimer* e *StopTimer* são nós do tipo *TimeSensor* e servem para controlar o movimento de translação. Quando o utilizador controla o *avatar* o nó *AvatarControle* está activo e como consequência o nó *ProximitySensor* também está activo - (1). Sempre que o utilizador se movimenta, o sensor detecta esse movimento e informa o nó *AvatarControle* - (2). De seguida, os dois temporizadores são activados - (3) e (4). O primeiro (*StartTimer*) irá controlar o movimento do *avatar* enquanto o segundo serve para activar a animação parar de forma a que o *avatar* se imobilize quando o utilizador parar de se movimentar. Isto é conseguido, inicializando temporizador de forma a este só se activar algumas unidades de tempo mais tarde. Assim, o tempo de activação só será alcançado quando o utilizador para de se movimentar e o nó *AvatarControle* deixar de receber eventos *position_changed* - (2). Após activados os temporizadores, o nó *PositionInterpolator* é inicializado de acordo com o deslocamento do utilizador e de forma a que a distancia entre o utilizador e o *avatar* no eixo dos *zz*, não exceda um determinado valor - (5). Finalmente, os eventos *fraction_changed* do temporizador *StartTimer* são encaminhados para o *PositionInterpolator* - (6), (7) -, via o *AvatarControle*, que por sua vez muda a posição do *avatar* - (8).

2.5.1 Nó *AvatarControle*

Dada a importância do nó *AvatarControle* para o comportamento do *avatar* este é descrito um pouco mais em detalhe.

```
DEF AvatarControle Script {
    ...
    field SFBool isMoving TRUE
    field SFNode Avatar USE WalkingAvatar
    field SFNode AvatarCamera USE UserView
    field SFNode NormalCamera USE NormalView
    field SFNode MenuSelector USE MenuSelector
    field SFNode TimeWalk USE TimeWalk
    field SFNode TimeRun USE TimeRun
    field SFBool walk TRUE
    directOutput TRUE
    eventIn SFVec3f set_position IS set_position
    eventIn SFRotation set_orientation IS set_orientation
    eventIn SFFloat set_fraction
    eventIn SFBool set_bind IS set_bind

    #Eventos recebidos para activar as animações
    # correr, andar, sentar e levantar, respectivamente

    eventIn SFBool set_run
    eventIn SFBool set_walk
    eventIn SFTIME sit
    eventIn SFTIME up

    # Eventos que activam as animações não e adeus
    eventOut SFTIME say_no
    eventOut SFTIME say_bye

    # eventos que controlam os TimeSensors associados
    # as animacoes correr e andar.
    eventOut SFTIME startTime
    eventOut SFTIME walkStartTime
    eventOut SFTIME runStartTime
    eventOut SFTIME stopTime
}
```

```

        url "vrmlscript:"
        ..
        " // codigo javascript ....
    }

```

Este nó é responsável por implementar a movimentação no mundo consoante a medida que o utilizador se move. Existe um *TouchSensor* cuja posição é constantemente actualizada, de forma a coincidir com a posição actual do avatar. O sensor possui um alcance considerável por forma a conseguir detectar o utilizador, mesmo quando um pouco afastado do avatar. A movimentação do avatar está feita de forma a ele se manter sempre, cerca de 3 unidades à frente do utilizador. Assim garante-se que o utilizador não perde o controle do avatar ao executar movimentações muito rápidas.

Os campos *eventOut position_changed* e *orientation_changed* do sensor foram direccionados para os campos *eventIn set_position* e *set_rotation* do script, respectivamente. Por sua vez o script actualiza a posição e orientação do avatar utilizando a referência guardada no campos *Avatar*. A razão porque os eventos do sensor foram direccionados para o script e não para o avatar directamente, deve-se ao facto de ser necessário efectuar algumas operações além de actualizar a posição (orientação) do avatar. Por exemplo, no caso da actualização da posição é necessário restringir a posição do avatar no eixo dos ZZ de forma ao avatar estar sempre à frente do utilizador e a uma determinada distância. Noutros casos, quando o avatar está sentado, por exemplo, a animação *Não* é activada para indicar ao utilizador que o avatar só se pode mover se estiver em pé. Além disso, a posição e orientação do nó *ViewPoint NormalView* (campo *NormalCamera*) são actualizadas de forma a que quando o utilizador decidir largar o controle do avatar, a camara *normal* esteja na mesma posição (e orientação) que o utilizador.

O campo *eventIn set_bind* recebe eventos consoante o nó *NormalView* está, ou não, activo. Quando o nó deixa de estar activo, o nó *UserView* que representa a câmara que o utilizador usa quando controla o avatar, é activado. Quando o nó fica activo então *set_bind* é verdadeiro e a animação *Adeus* é activada. É também activado o menu *menu1* (ver 4.4.2).

3 Conclusões

Devido a «limitações» do VRML não foi possível implementar o controlo do avatar que seria de desejar. Uma dessas limitações é o facto de as colisões só serem detectadas entre o utilizador e os objectos existentes numa cena. O VRML não oferece nenhum mecanismo para detectar colisões entre objectos da cena. Quando se pretendem efectuar animações complexas, seria útil por vezes saber quando um dado objecto colide com outro e impedir, dessa forma, eventuais sobreposições. A ausência desta funcionalidade tem como consequência que não é possível impedir que o avatar atravesse os objectos existentes no mundo. A única forma de o conseguir seria programaticamente fazer a detecção de colisões entre o avatar e os outros objectos da cena, opção essa que decidimos não tomar, dada a sua complexidade.

Outro dos problemas encontrados, está relacionado com o nó *Anchor* da actual especificação VRML. Este nó permite que num mundo sejam definidas ligações para outros mundos descritos em ficheiros VRML diferentes. Utilizando um avatar seria desejável que quando o utilizador seleccionasse um nó *Anchor*, que o seu avatar aparecesse no novo mundo carregado pelo *browser*. Acontece que não é possível, por exemplo, especificar que certos objectos do mundo actual devem ser «transportados» para o novo mundo. Utilizando a especificação actual, só se consegue a funcionalidade pretendida recorrendo às APIs dos *browsers*.

Ambos os problemas referidos poderiam ser resolvidos se a especificação VRML fosse alterada de forma a por exemplo, permitir associar nós à informação sobre o avatar. Presentemente o VRML permite que sejam dadas aos *browsers* algumas informações acerca de o avatar que servem, por exemplo, para o controle de detecção de colisões, ou para simular o efeito do utilizador se posicionar em cima de certos objectos. Seria interessante que, para além dessas informações, fosse possível (opcionalmente) indicar um nó que representaria fisicamente o avatar no mundo. A detecção de colisões entre objectos e o utilizador, seria

feita utilizando a geometria do nó indicado. Além disso, na navegação entre mundos, o avatar seria também «transportado» para o mundo destino.

4 Bibliografia

- [1] Especificação HANIM 1.1. <http://ece.uwaterloo.ca/~h-anim/>
- [2] Especificação VRML97. <http://www.web3d.org/VRML2.0/DRAFT3/>
- [3] Dive. <http://www.sics.se/dive/dive.html>
- [4] Massive. <http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-3/>
- [5] ActiveWorlds. <http://www.activeworlds.com>
- [8] Blaxun. <http://www.blaxun.com/>
- [7] External Authoring Interface. <http://www.vrml.org/WorkingGroups/vrml-eai/>