appendix

A. Web3D – VRML/X3D

Nowadays PCs allow for powerful 3D graphics. 3D graphics are, until now, mainly used by dedicated applications such as CAD/CAM and, not to forget, games. It is to be expected that 3D graphics will also manifest themselves in other types of applications, including web applications. In the Multimedia Authoring I course, students are required to develop such applications:

Multimedia Authoring I – Web3D/VRML

- *product demo* – with descriptive information and animation(s)
- *infotainment VR* – in the areas of Culture, Commerce or Entertainment

The latter assignment, the *infotainment VR*, may result in either a virtual museum, a game, or an extended product demo with a suitable environment and interaction facilities.

The purpose of the Web3D/VRML course is not so much the modeling of 3D objects *per s&eaigu;* but rather the organisation of 3D material (using the PROTO construct) and the development of suitable interaction mechanisms and guided tours (using sensors and scripts). This course, as well as the other multimedia authoring course is focussed on a programmatic approach to 3D. Hence, no advanced tools are used. Not because they are too expensive (which is also true), but because students should learn the basics first!

Why did we choose for Web3D, and more in particular VRML? Some argue that VRML is slow. Moreover, navigation in VRML is not altogether pleasant. Why not a (more native) format such as OpenGL? The answer is simply that VRML offers the right level of abstraction for modeling and programming 3D worlds. OpenGL does not. In the timespan of one month, VRML allows you to develop rather interesting and complex worlds, whereas with OpenGL (using C or C++) you would probably still be stuck with very simple scenes.

As concerns the focus on Web3D, I simply state that delivery of (rich media) 3D content is the way to go. The web is our global information repository, also for multimedia and 3D content. And we should be optimistic about performance issues. Already Web3D is of much better quality than the native 3D in the beginning of the 1990s.

What will be the future of Web3D and VRML? I don't know. As concerns VRML, the 3D modeling concepts and programming model underlying VRML are sufficiently established (as they are also part of X3D) that VRML will very

likely survive in the future. The future of Web3D will depend on the success of the Web3D consortium of which a mission statement is given below.

www.web3d.org

> *The term Web3D describes any programming or descriptive language that can be used to deliver interactive 3D objects and worlds across the internet. This includes open languages such as Virtual Reality Modeling Language (VRML), Java3D and X3D (under development) - also any proprietary languages that have been developed for the same purpose come under the umbrella of Web3D. The Web3D Repository is an impartial, comprehensive, community resource for the dissemination of information relating to Web3D and is maintained by the Web3D Consortium.*

More in particular, the Web3D repository includes the X3D SDK to promote the adoption of X3D in industry and academia.

X3D SDK

> *This comprehensive suite of X3D and VRML software is available online at sdk.web3d.org and provides a huge range of viewers, content, tools, applications, and source code. The primary purpose of the SDK is to enable further development of X3D-aware applications and content.*

However, before downloading like crazy, you'd better get acquainted with the major concepts of VRML first. After all, VRML has been around for some time and VRML technology, although not perfect, seems to be rather stable.

Virtual Reality Modeling Language

VRML is a scengraph-based graphical format. A scenegraph is a tree-like structure that contains nodes in a hierarchical fashion. The scenegraph is a description of the static aspects of a 3D world or scene. The dynamic aspects of a scene are effected by routing events between nodes. When routing events, the hierarchical structure of the scenegraph is of no importance. Assuming compatible node types, event routing can occur between arbitrary nodes.

Below, an overview is given of the types of nodes supported by VRML as well as a number of browser-specific extensions introduced by *blaxxun*. The nodes that you might need for a first assignment are indicated by an asteriks. Additional information on the individual nodes is available in the online version.

abstraction and grouping

- *abstraction* – Inline Switch*
- *grouping* – Billboard, Collision Group, Transform*
- *scene* – Background LOD NavigationInfo Viewpoint* WorldInfo

geometry and appearance

- *geometry* – Box* Cone Coordinate Cylinder ElevationGrid Extrusion IndexedFaceSet IndexedLineSet Normal PointSet Shape* Sphere*

- *appearance* – Appearance* Color* Imagetexture* Material* MovieTexture PictureTexture TextureCoordinate TextureTransform
- *text* – FontStyle Text*

interaction and behavior

- *sensors* – Anchor CylinderSensor PlaneSensor ProximitySensor SphereSensor TimeSensor* TouchSensor* VisibilitySensor
- *behavior* – Script*
- *interpolators* – ColorInterpolator* CoordinateInterpolator NormalInterpolator OrientationInterpolator* PositionInterpolator* ScalarInterpolator

special effects

- *sound* – AudioClip Sound
- *light* – DirectionalLight Fog PointLight Spotlight

extensions

- *blaxxun* – Camera DeviceSensor Event KeySensor Layer2D Layer3D MouseSensor MultiTexture Particles TextureCoordGen

Not mentioned in this overview is the PROTO facility and the DEF/USE mechanism. The PROTO facility allows for defining nodes, by declaring an interface and a body implementing the node. Once a PROTO definition is given, instances of the PROTO can be created, in the same way as with built-in nodes. The DEF/USE maechanism may be applied for routing events as well as the reuse of fragments of code. Beware, however, that reuse using USE amounts to sharing parts of the scenegraph. As a consequence, one little change might be visible wherever that particular fragment is reused. In contrast, multiple instances of a PROTO are independent of eachother.

3D slides – the code

As you may have discovered, the material in this book is also available in the form of slides. Not Powerpoint slides but 3D slides, using VRML, with occasionally some graphic effects or 3D objects. At the Web3D Symposium 2002, I was asked *What is the secret of the slides?*. Well, there is no secret. Basically, it is just a collection of PROTOs for displaying text in VRML.[1]

protos

- *slideset* – container for slides
- *slide* – container for text and objects
- *slide* – container for lines of text
- *line* – container for text
- *break* – empty text

[1] The PROTOs were initially developed by Alex van Ballegooij, who also did the majority of the coding of an extended collection of PROTOs.

Note that for displaying 3D objects in a slide, we need no specific PROTO.

Before looking at the PROTO for a set of slides, let's look at the *slide* PROTO. It is surprisingly simple.

slide

```
PROTO slide [
   exposedField SFVec3f    translation 0 0 15
   exposedField SFRotation rotation    0 1 0 0
   exposedField SFVec3f    scale       1 1 1
   exposedField MFNode     children []
] {
   Transform {
      children    IS children
      translation IS translation
      rotation    IS rotation
      scale       IS scale
   }
}
```

The *slide* PROTO defines an interface which may be used to perform spatial transformations on the slide, like translation, rotation and scaling. The interface also includes a field to declare the content of the slide, that is text or (arbitrary) 3D objects.

The interface of the *slideset* PROTO allows for declaring which slides belong to the set of slides.

slideset

```
PROTO slideset [
   exposedField SFInt32    visible 0
   exposedField MFNode     slides [ ]
   eventIn SFInt32         next
] {
   DEF select Switch {
      choice      IS slides
      whichChoice IS visible
   }

   Script {
   ...
   }
}
```

Apart from the *visible* field, which may be used to start a presentation with another slide than the first one (zero being the first index in the array of slides), the *slideset* PROTO interface also contains a so-called *eventIn* named *next* to proceed to the next slide.

To select between the different slides a *Switch* node is used, which is controlled by a *Script*. The code of the script is given below.

script

```
Script {
directOutput TRUE
eventIn SFInt32 next IS next
field SFInt32 slide IS visible
field SFNode select USE select
field MFNode slides []
url "javascript:
function next(value) {
slides = select.choice;
Browser.print('=' + slide + ' ' + slides.length);
if (slide ¿= (slides.length-1)) slide = 0;
else slide += 1;
select.whichChoice = slide;
}"
}
```

In the interface of the script, we see both the use of *IS* and *USE* to connect the (local) script fields to the scenegraph. The function *next*, that implements the corresponding event, simply traverses through the slides, one step at a time, by assigning a value to the *whichChoice* field of the *Switch*.

example As an example of applying the *slide* PROTOs, look at the fragment below.

example

```
DEF slides slideset {
 slides [
   slide {
      children [
            text {
                  lines [
                    line { string ["What about the slide format?"] }
                    break { }
                    line { string ["yeh, what about it?"] }
                    break { }
                  ] # lines
            }
            Sphere { radius 0.5 }
      ] # children
   } # slide 1

   slide { # 2
      children [
            Sphere { radius 0.5 }
      ]
   } # slide 2
 ] # slides
}
```

In the online version you may see how it works. (Not too good at this stage, though, since we have not included a proper background and viewpoint.)

For traversing between slides, we need a mechanism to send the *next* event to the *slideset* instance. In the current example, a timer has been used, defined by the code below.

timer

```
DEF time TimeSensor { loop TRUE cycleInterval 10 }
DEF script Script {
eventIn SFTime pulse
eventOut SFInt32 next
url "javascript: function pulse(value) { next = 1; }"
}
ROUTE time.cycleTime TO script.pulse
ROUTE script.next TO slides.next
```

Obviously, better interaction facilities are needed here, for example a simple button (which may be implemented using a *TouchSensor* and a *Sphere*) to proceed to the next slide. These extensions, as well as the inclusion of a background and viewpoint, are left as an exercise.

Naturally, the actual PROTOs used for the slides in this book are a bit more complex than the collection of PROTOs presented here. And, also the way slides themselves, that is the content, is different from what we have shown in the example. In appendix we will see how we can use XML to encode (the content) of slides. However, we will deploy the PROTOs defined here to get them to work.

B. XML-based multimedia

XML is becoming a standard for the encoding of multimedia data. An important advantage of XML-based encodings is that standard XML tools, such as XSLT stylesheet-based processing, are available. Another advantage is that the interchange of data becomes more easy. Examples of XML-based media formats include SMIL, X3D, Speech ML, Voice XML.

In fact, to my mind, we should have a course on XML-based multimedia. Zhisheng Huang, who developed the STEP language (and its XML-encoding) which is described in the next section, has compiled a list of topics that you should know about XML-based multimedia.

XML-based multimedia

- *introduction*: Extensible Markup Language (XML). Extensibility and profiling of web-based multimedia. Streaming. Model of timing and synchronization of web-based multimedia.
- *processing XML*: XSLT stylesheets, Java-based XML Processing, SAX, DOM, Java XSL object APIs
- *SMIL*: (Synchronized Multimedia Integration Language) SMIL modules: animation, content control, layout, linking, media object, metainformation, timing, and profiles.
- *X3D*: (XML-based VRML) Extensible 3D: architecture and based components, profile reference, translation between VRML and X3D. X3D examples: case studies.
- *VHML*: (Virtual Human Markup Language) Virtual Human Markup Language, Humanoid, H-anim specification, Speech Synthesis Markup Language Specification for the Speech Interface Framework (Speech ML), Voice Extensible Markup Language (VoiceXML). Text to Speech Technology.
- *STEP*: Scripting Technology for Embodied Persona and XSTEP, the XML-encoding of STEP and its processing tools. Embodied agents and multimedia presentation: theory, model, and practice.

The course should emphasize practice and experience. An example assignment is the development of an information system, including multimedia data in the form of images, 3D objects and audio recordings. The content should be organized according conceptual criteria, in an XML format to be designed by the student. Additional processing tools should then be written, using XSLT, to create a web site and to generate presentations in which the material is displayed from

a particular perspective, for example a historic timeline, in one or more of the available presentation formats. See appendix **??** for a (more or less) concrete example.

As noted in the *research directions* of section **??**, XML comes with a set of related technologies. For processing XML we have XSLT, the transformation language which allows us to generate arbitrary text (including XML) from the information content of XML-encoded information. In the following, we will look at the use of XSLT to generate VRML-code from XML-encoded slides, using the collection of PROTOs developed in appendix **??**.

3D slides in XML

To refresh your memory, a slide set is a collection of slides that may contain lines of text and possibly 3D objects. Writing slides in VRML would be rather tedious. Besides, slides written in VRML could not be used in, say, HTML pages.

So the solution I came up with is to isolate particular pieces in a text as slides and to process these slides to create a presentation. In effect, both dynamic HTML-based and VRML-based presentations are supported. As a notation, an XML-based encoding seems to be the most natural, since it very close already to HTML, thus reducing the amount of processing needed to convert text containing slides to HTML. Now, how should the conversion to VRML take place. The answer is, simply, by using XSLT.

Let's first look at the XML-encoding of the example slides of appendix **??**.

slides in XML

```
<slideset>
<slide id="1">
<text>
<line>What about the slide format?</line>
<break/>
<line string="yeh, what about it">?</line>
</text>
<vrml>Sphere { radius 0.5 }</vrml>
</slide>
<slide id="2">
<vrml>Sphere { radius 0.5 }</vrml>
</slide>
</slideset>
```

One difference is that we introduced an *id* attribute in the *slide* tag, to allow for cross-referencing. These *id* attributes are, however, ignored in the conversion to VRML. Also, a *string* attribute has been introduced for the *line* tag. This is, however, just to illustrate how attributes are dealt with in processing XML files.

Before looking at the stylesheet used for the conversion to VRML, let me briefly say something about XSLT. The XSLT transformation language is a declarative language. It allows for processing an XML-encoded text by templates matching particular tags. In addition, the values of attributes of tags may be used when generating output.

The first part of our XSLT stylesheet looks as follows.

XSLT *stylesheet*

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
```

Apart from the obligatory declaration that the stylesheet itself is written in XML, there is also the indication that the file is a stylesheet written according to the rules and conventions that can be found in a file dating from 1999, as given in the url. Since we do want to generate VRML (and not XML), we need to indicate that our output method is *text*, to avoid having an XML header at the start of the output.

Now we are ready to define our first template.

slideset

```
<xsl:template match="/slideset">

... load (extern) proto(s)

DEF slides slideset {
slides [
<xsl:apply-templates/>
] # slides
}

... include timer or user interface

</xsl:template>
```

Everything that is not part of a tag containing the *xsl* prefix is literally copied to output. In this fragment, I have not included the full PROTO declarations nor the timer or user interface needed to traverse the slides. In the middle of the fragment we see the *xsl* tag *apply-templates*. This results in further processing the content that is contained between the *slideset* begin and end tag, using the template definitions given below.

The template for the *slide* tag is simple.

slide

```
<xsl:template match="*/slide">
slide { children [
<xsl:apply-templates/>
] }
</xsl:template>
```

You will recognize the structure, which is in agreement with the way we encoded slides in VRML, as illustrated in appendix **??**.

The template for the *text* is equally simple.

text

```
<xsl:template match="*/text">
text { lines [
<xsl:apply-templates/>
] }
</xsl:template>
```

For the *line* tag we need to do a bit more. Namely, we have to ask for the value of the *string* attribute, to obtain the complete result.

line

```
<xsl:template match="*/line">
line { string [ "<xsl:value-of select="@string"/>
<xsl:apply-templates/> " ] }
</xsl:template>
```

Note that, as mentioned above, the *string* attribute was just introduced to illustrate how to process attributes and is in itself superfluous. Actually, this way the *line* tag can be used as a closed tag, containing only the attribute and no contents, or an open tag with contents and possibly attributes.

Then, we are almost done.

etcetera

```
<xsl:template match="*/break">
line { string [ "<xsl:apply-templates/>" ] }
</xsl:template>

<xsl:template match="*/vrml">
<xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

We need to define a template for the *break* tag and a template for the *vrml* tag, which does nothing but copy what is beteen the *vrml* begin and end tag.

And that's it. Check the online version for the resulting slides obtained by processing this specification with the XSLT stylesheet given above.

You may have wondered why no mention was made of a DTD or *schema*. Simply, because we do not need such a thing when processing an XML-file using XSLT stylesheets.

When you want to use XSLT to process your own XML-encoded information, you will probably want to know more about XSLT. That is a good idea. Consult XSLT or one of the online tutorials.

D. a platform for intelligent multimedia

We have developed a platform for *intelligent multimedia*, based on distributed logic programming (DLP) and X3D/VRML. See Platform. Now, before giving a more detailed description of the platform, let's try to provide a tentative definition of *intelligent multimedia*.

intelligent multimedia

> ... intelligent multimedia *provides a merge between technology from AI, in particular agent-technology, and multimedia ...*

However shallow this definition might be, it does indicate that we are in a multidisciplinary field of research that investigates how we may approach multimedia in a novel manner, using knowledge technology developed in Artificial Intelligence. More pragmatically, *intelligent multimedia* characterizes a programmatic approach to multimedia making use of high-level declarative languages, in opposition to low-level third generation and scripting languages, to reduce the programming effort involved in developing (intelligent) multimedia systems. Does this make the application themselves more intelligent? Not necessarily. In effect, nothing can be done that could not have been done using the available programmatic interfaces. However, we may argue that the availability of a suitable programming model makes the task (somewhat or significantly) easier.

In our Multimedia Authoring II course, students become familiar with our *intelligent multimedia* technology.

Multimedia Authoring II – virtual environments

- *intelligent services in virtual environments*

Knowledge of Web3D/VRML, as taught in Multimedia Authoring I, is a prerequisite. The course gives a brief introduction to logic programming in Prolog and DLP and then continues with building virtual environments using agent-technology to control the dynamic aspects of these environments.

distributed logic programming

The language DLP has a respectable history. It was developed at the end of the 1980s, DLP, and was implemented on top of Java at the end of the 1990s. In

retrospect, the language turned out to be an agent-programming language *avant la lettre*. What does it offer? In summary:

DLP

- *extension of Prolog*
- *(distributed) objects*
- *non-logical instance variables*
- *multiple inheritance*
- *multi-threaded objects*
- *communication by rendez-vous*
- *(synchronization) accept statements*
- *distributed backtracking*

Basically, the language is a distributed object-oriented extension of Prolog. It supports multiple inheritance, non-logical instance variables and multi-threaded objects (to allow for distributed backtracking). Object methods are collections of clauses. Method invocation is dealt with as communication by rendez-vous, for which synchronization conditions may be specified in so-called *accept* statements. As indicated above, the current implementation of DLP is built on top of Java. See OO, appendix E for more details.

DLP+X3D platform

Our platform is the result of merging VRML with the distributed logic programming language DLP, using the VRML External Authoring Interface. This approach allows for a clear separation of concerns, modeling 3D content on the one hand and determining the dynamic behavior on the other hand. As a remark, recently we have adopted X3D as our 3D format. The VRML profile of X3D is an XML encoding of VRML97.

To effect an interaction between the 3D content and the behavioral component written in DLP, we need to deal with two issues:

- control points: *get/set* – position, rotation, viewpoint
- *event-handling* – asynchronous accept

We will explain each of these issues separately below. In addition, we will indicate how multi-user environments may be realized with our technology.

control points The control points are actually nodes in the VRML scenegraph that act as handles which may be used to manipulate the scenegraph. In effect, these handles are exactly the nodes that may act as the source or target of event-routing in the 3D scene. As an example, look at the code fragment below, which gives a DLP rule to determine whether a soccer player must shoot:

```
findHowToReact(Agent,Ball,Goal,shooting) :-
   get(Agent,position,sfvec3f(X,Y,Z)),
   get(Ball,position,sfvec3f(Xb,Yb,Zb)),
```

```
   get(Goal,position,sfvec3f(Xg,Yg,Zg)),
   distance(sfvec3f(X,Y,Z),sfvec3f(Xb,Yb,Zb),DistB),
   distance(sfvec3f(X,Y,Z),sfvec3f(Xg,Yg,Zg),DistG),
   DistB =< kickableDistance,
   DistG =< kickableGoalDistance.
```

This rule will only succeed when the actual distance of the player to the goal and to the ball satisfies particular conditions, see section **??**. In addition to observing the state of the 3D scene using the *get* predicate, changes to the scene may be effected using the *set* predicate.

event handling Our approach also allows for changes in the scene that are not a direct result of setting attributes from the logic component. Therefore we need some way to intercept events. In the example below, we have specified an observer object that has knowledge of, that is inherits from, an object that contains particular actions.

```
:- object observer : [actions].
var slide = anonymous, level = 0, projector = nil.

observer(X) :-
   projector := X,
   repeat,
     accept( id, level, update, touched),
   fail.

id(V) :-  slide := V.
level(V) :- level := V.
touched(V) :- projector←touched(V).
update(V) :- act(V,slide,level).
:- end_object observer.
```

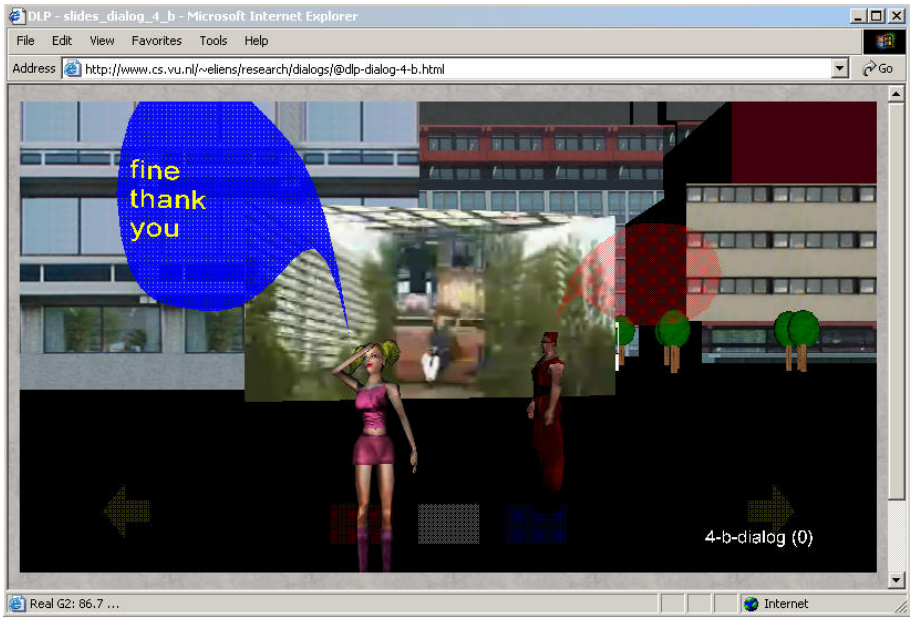
The constructor sets the non-logical variable *projector* and enters a repeat loop to accept any of the incoming events for respectively *id*, *level*, *update* and *touched*. Each event has a value, that is available as a parameter when the corresponding method is called on the acceptance of the event. To receive events, the *observer* object must be installed as the listener for these particular events.

The events come from the 3D scene. For example, the *touched* event results from mouse clicks on a particular object in the scene. On accepting an event, the corresponding method or clause is activated, resulting in either changing the value of a non-logical instance variable, invoking a method, or delegating the call to another object.

An observer of this kind is used in the system described below, to start a comment (dialog) on the occurrence of a particular slide.

case studies

To illustrate the potential of our DLP+X3D platform, we will briefly sketch two additional case studies deploying embodied agents, respectively the use of dialogs in VR presentations (fig. a), and a scripting language for specifying gestures and movements for humanoids (fig. b).



dialogs in virtual environments

Desktop VR is an excellent medium for presenting information, for example in class, in particular when rich media or 3D content is involved. At VU, I have been using *presentational VR* for quite some time, and recently I have included dialogs using balloons (and possibly avatars) to display the text commenting on a particular presentation. See figure (b) for an example displaying a virtual environment of the VU, a propaganda movie for attracting students, and two avatars commenting on the scene. The avatars and their text are programmed as annotations to a particular scene as described below.

Each presentation is organized as a sequence of slides, and dependent on the slides (or level within the slide) a dialog may be selected and displayed. See the *observer* fragment presented above.

Our annotation for dialog text in slides looks as follows:

```
<phrase right="how~are~you">
<phrase left="fine~thank~you"/>
<phrase right="what do~you think~of studying ..."/>
...
```

```
<phrase left="So,~what~are you?"/>
<phrase right="an ~agent" style="[a(e)=1]"/>
<phrase left="I always~wanted to be~an agent" style="[a(e)=1]"/>
```

In figure (b), you see the left avatar (named *cutie*) step forward and deliver her phrase. This dialog continues until *cutie* remarks that she *always wanted to be an agent*. The dialog is a somewhat ironic comment on the contents of the movie displayed, which is meant to introduce the VU to potential students.[2]

Furthermore, there are a number of style parameters to be dealt with to decide for example whether the avatars or persona are visible, where to place the dialogs balloons on the display, as well as the color and transparancy of the balloons. To this end, we have included a *style* attribute in the *phrase* tag, to allow for setting any of the style parameters.

Apart from phrases, we also allow for gestures, taken from the built-in repertoire of the avatars. Below we discuss how to extend the repertoire of gestures, using a gesture specification language.

Both phrases and gestures are compiled into DLP code and loaded when the annotated version of the presentation VR is started.

STEP – a scripting language for embodied agents

Given the use of humanoid avatars to comment on the contents of a presentation, we may wish to enrich the repertoire of gestures and movements to be able, for example, to include gestural comments or even instructions by gestures.

Recently, we have started working on a scripting language for humanoids based on dynamic logic. The STEP scripting language consists of basic actions, composite operators and interaction operators (to deal with the environment in which the movements and actions take place).

The basic actions of STEP consist of:

- *move* – `move(Agent,BodyPart,Direction,Duration)`
- *turn* – `turn(Agent,BodyPart,Direction,Duration)`

These basic actions are translated into operations on the control points as specified by the H-Anim 1.1 standard.

As composite operators we provide sequential and parallel composition, as well as *choice* and *repeat*. These composite operators take both basic actions and user-defined actions as parameters.

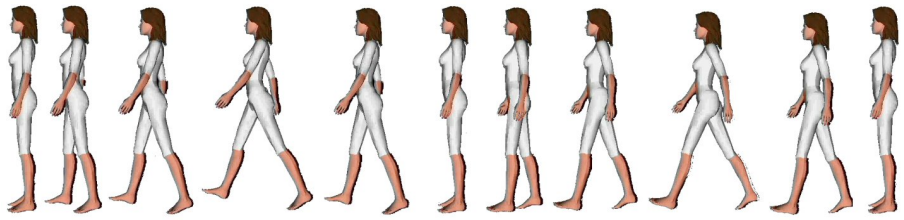
Each action is defined using the *script*, by specifying an action list containing the (possibly compound) actions of which that particular action consists. As an example, look at the definition of *walking* below.

```
script(walk(Agent), ActionList) :-
ActionList = [
  parallel([turn(Agent,r_shoulder,back_down2,fast),
         turn(Agent,r_hip,front_down2,fast),
```

[2] Clearly, our approach is reminiscent to the notorious *Agneta & Frida* characters developed in the Persona project. See the *research directions* of section **??**.

```
            turn(Agent,l_shoulder,front_down2,fast),
            turn(Agent,l_hip,back_down2,fast)]),
     parallel([turn(Agent,l_shoulder,back_down2,fast),
            turn(Agent,l_hip,front_down2,fast),
            turn(Agent,r_shoulder,front_down2,fast),
            turn(Agent,r_hip,back_down2,fast)])
], !.
```

Notice that the *Agent* that is to perform the movement is given as a parameter. (Identifiers starting with a capital act as a logical parameter or variable in Prolog and DLP.)



Interaction operators are needed to conditionally perform actions or to effect changes within the environment by executing some command. Our interaction operators include: *test*, *execution*, *conditional* and *until*.

Potentially, an action may result in many parallel activities. To control the number of threads used for an action, we have created a scheduler that assigns activities to a thread from a thread pool consisting of a fixed number of threads.

As a demonstrator for STEP, we have created an instructional VR for *Tai Chi*, the Chinese art of movement.

XML encoding Since we do not wish to force the average user to learn DLP to be able to define scripts in STEP, we are also developing XSTEP, an XML encoding for STEP. We use *seq* and *par* tags as found in SMIL, as well as *gesture* tags with appropriate attributes for speed, direction and body parts involved. As an example, look at the XSTEP specification of the *walk* action.

```
<action type="walk(Agent)">
<seq>
 <par speed="fast">
 <gesture type="turn" actor="Agent" part="r_shoulder" dir="back_down2"/>
  ...
 </par>
 <par speed="fast">
  ...
 <gesture type="turn" actor="Agent" part="r_hip" dir="back_down2"/>
 </par>
</seq>
</action>
```

Similar as with the specification of dialog phrases, such a specification is translated into the corresponding DLP code, which is loaded with the scene it belongs to. For XSTEP we have developed an XSLT stylesheet, using the Saxon package, that transforms an XSTEP specification into DLP. We plan to incorporate XML-processing capabilities in DLP, so that such specifications can be loaded dynamically.

related work

There is an enormous amount of research dealing with virtual environments that are in one way or another inhabited by embodied agents. By way of comparison, we will discuss a limited number of related research projects.

As systems that have a comparable scope we may mention Environments and DIVE, that both have a client-server architecture for realizing virtual environments. Our DLP+X3D platform distinguishes itself from these by providing a uniform programmatic interface, uniform in the sense of being based on DLP throughout.

The Parlevink group at the Dutch University of Twente has done active research in applications of virtual environments with agents. Their focus is, however, more on language processing, whereas our focus may be characterized as providing innovative technology.

Both Jinni and Scripts deal with incorporating logic programming within VRML-based scenes, the former using the External Authoring Interface, and the latter inline logic scripts. Whereas our platform is based on distributed objects, Jinni deploys a distributed blackboard to effect multi-user synchronisation.

Our scripting language may be compared to the scripting facilities offered by Alice, which are built on top of Python. Also, *Signing Avatar* has a powerful scripting language. However, we wish to state that our scripting language is based on dynamic logic, and has powerful abstraction capabilities and support for parallelism.

Finally, we seem to share a number of interests with the VHML community, which is developing a suite of markup languages for expressing humanoid behavior. We see this activity as complementary to ours, since our research proceeds from technical feasibility, that is how we can capture the semantics of humanoid gestures and movements within our dynamic logic, which is implemented on top of DLP.

future research

In summary, we may state that our DLP+X3D platform is a powerful, flexible and high-level platform for developing VR applications with embodied agents. It offers a clean separation of modeling and programming concerns. On the negative side, we should mention that this separation may also make development more

complex and, of course, that there is a (small) performance penalty due to the overhead incurred by using the External Authoring Interface.

Where our system is currently lacking, clearly, is adequate computational models underlying humanoid behavior, including gestures, speech and emotive characteristics. The VHML effort seems to have a rich offering that we need to digest in order to improve our system in this respect.

Our choice to adopt open standards, such as XML-based X3D, seems to be benificial, in that it allows us to profit from the work that is being done in other communities, so that we can enrich our platform with the functionality needed to create convincing embodied agents in a meaningful context.

D. resources, tools and technology

What do you need to have to start working on your multimedia project? that depends, naturally, on what you want to do. In the following, I will give a brief overview of resources, tools and technolgies that you might find useful or that you might want to explore. This overview consists mainly of urls and a brief characterization and in some cases an indication of a price range.

This overview is definitely not meant to be complete, and is only included for your convenience, so that you don't have to *google*[3] it yourself. In the online version of the book more (online) resources are given, as well as a (clickable) list of all urls that appear (as a footnote) in the book.

resource(s)

This section contains a variety of itmes, including a selection of online tutorials and thesauri. Some examples are given of online museum tours and listings are included of the media art and cultural heritage institutes mentioned in the book. But we will start with introducing briefly with what you need for 3D authoring and rendering, since this is what we have primarily focused on in theis book.

3D authoring & conversion

- vrmlpad – www.parallelgraphics.com/products/vrmlpad
- polytrans – www.okino.com/products.htm
- maya – www.alias.com
- 3dsmax – www.discreet.com
- sketchup – sketchup.google.com/download.html
- flux studio – www.mediamachines.com/products.html

The *polytrans* tool from Okino has been included, since it allows you to convert almost any format into you format of choice, which is a great asset for (re) using models.

[3] www.google.com

3D rendering

- blaxxun – www.blaxxun.com/en/products/contact
- virtools – www.virtools.com
- flux web3d – sourceforge.net/projects/flux
- mediamachines flux – www.mediamachines.com/products.html

As concerns price, VRML-based solutions for authoring and rendering are clearly low-cost, whereas tools such as *Maya* and *Studio Max* require more investment, not only in money but also in learning time. Also *Virtools* is in the higher price range.

tutorials

- html – www.mcli.dist.maricopa.edu/tut
- javascript – www.javascriptkit.com
- php – www.php.net/docs.php
- rdf – www.w3.org/TR/rdf-primer
- vrml – web3d.vapourtech.com/tutorials/vrml97
- java – java.sun.com/docs/books/tutorial
- 3D modeling – www.raph.com/3dartists/tutorials/t-3dsmax.html
- games in VRML – www.3dezine.com/3DEZine/gamestory.html
- ria – www.macromedia.com/resources/business/rich_internet_apps/whitepapers.html

In many cases it is (more) convenient to have working examples at hand. Personally, I advice my students to learn using HTML, VRML, Javascript and the like from one of the online tutorials, which do provide such examples. The *php* documentation is not really a tutorial but does provide useful help and examples.

visual design

- collage – www.artlex.com/ArtLex/c/collage.html
- storyboard – www.thestoryboardartist.com/links.html
- drawing – www.thestoryboardartist.com/tutorial.html

For *visual design* it might be worthwhile to look at some examples, or even take a complete course in drawing.

museum

- van gogh – www.vangoghmuseum.nl
- rijksmuseum – www.rijksmuseum.nl
- canada – www.virtualmuseum.ca/English/index_flashFT.html
- zkm – www.zkm.de
- tate – www.tate.org.uk
- louvre – www.louvre.fr

More inspiration can perhaps be obtained from looking at what musea have to offer. It also gives you an opportunity to update your knowledge of the history of art.

media art

- montevideo – www.montevideo.nl
- V2 – www.v2.nl
- electronic arts intermix – www.eai.org/eai
- cinemanet – www.cinemaneteurope.com
- variable media – www.variablemedia.net
- net art – wwwwwwwww.jodi.org/100cc/index.html
- mediamatic – www.mediamatic.net

Listed above are institutions that play a role in the preservation and dissemination of contemporary media art. Not an institution, but an early pioneer of art on the internet, is *jodi* from *net art*.

virtual tours

- amsterdam – www.channels.nl
- panoramic amsterdam – www.panoramsterdam.nl
- rijksmuseum – www.rijksmuseum.nl/collectie/meesterwerken/?lang=en
- groningen – www.kalamiteit.nl/world/no_cache/museum/vrml/connect.html
- mondriaan – www.artmuseums.harvard.edu/mondrian

Many cities nowadays have virtual tours. And also many musea allow the (online) visitor to have a look at their collection.

cultural heritage

- incca – www.incca.org
- delos – www.delos.info
- echo – echo.mpiwg-berlin.mpg.de/home
- eu – www.iue.it/ECArchives
- cidoc – www.cidoc.icom.org
- collate – www.collate.de
- cimwos – www.xanthi.ilsp.gr/cimwos
- library of congress – www.loc.gov./
- scriptorium – sunsite.berkeley.edu/scriptorium
- tei – www.tei-c.org
- open archives – www.tei-c.org
- topia – topia.telin.nl

Above is a mixed collection of references to organizations and institutions that are in some way involved in cultural heritage projects, either related to traditional art or contemporary art.

thesaurus

- webopedia – www.webopedia.com
- visual – www.visualthesaurus.com
- 3D glossary – www.nvidia.com/page/pg_20010527107687.html
- art & architecture – www.getty.edu/research/conducting_research/vocabularies/aat/
- modern art – en.wikipedia.org/wiki/Modern_art
- (new) media art – en.wikipedia.org/wiki/New_Media_art
- art online – www.art-online.com
- multimedia – www.insead.fr/CALT/Encyclopedia/Media/multimedia.html
- virtual reality – www.insead.fr/CALT/Encyclopedia/ComputerSciences/VR
- gaming – www.insead.fr/CALT/Encyclopedia/ComputerSciences/Gaming
- mathematics – www.cs.brown.edu/people/scd/facts.html
- mpeg – www.m4if.org/mpeg4
- wikipedia – en.wikipedia.org/wiki/Multimedia

There is a wealth of online information sources, including glossaries and thesauri. Beware, not all of them are properly authorized. Nevertheless, it might be interesting to note that the online version of this book is referred to in the *wikipedia*, for the entry *multimedia*.

games

- gamasutra – www.gamesutra.com
- gamedev – www.gamedev.net
- developer – www.gdmag.com/resources.html
- and more – www.lostlogic.com/postnuke
- games at school – www.freewebs.com/schoolgamemaker
- gamemaker – www.gamemaker.nl/
- game learning – www.gamelearning.net
- scripting – htpp://www.lua.org
- open source – www.delta3d.org
- free source – www.thefreecountry.com/sourcecode/games.shtml

For games, there are several popular sites providing information about new upcoming games, as well as developer's resources, including software available for download.

A recommended open source game engine is *Delta3D*. This package contains a variety of open source software, well-integrated due to the efforts of a dedicated team at the Naval Postgraduate School in Monterey, CA/USA.

serious games

- play2learn – www.play2learn.nl
- nitrogenius – www.serc.nl/play2learn/products/nitrogenius
- at school – rla.oakland.edu/∼ist_699
- primary games – www.primarygames.com

- games at school – www.freewebs.com/schoolgamemaker
- arcade – www.educationarcade.org
- never winter – nwn.bioware.com

Serious games are a new brand of games. Not really new in terms of technology, but new with respect to focus and intent.

tool(s)

There is a great variety of tools, with huge differences in prize. Often, however, you can download a fully functional trial version that will last for a month, and thus may determine the length of your project. A number of tools, however, come with a free (such as Maya) or limited price (such as 3DSMax) student version.

imaging and graphics

- photoshop – www.adobe.com/products/photoshop
- illustrator – www.adobe.com/products/illustrator
- snagit – www.techsmith.com/products/snagit
- camtasia – www.techsmith.com/products/studio

Perhaps the most popular tools among designers are *photoshop* and *illustrator*. Both for capture and image catalogue maintenance I have benefited from *snagit* and *camtasia*, both from *techsmit*.

3D modeling

- vrmlpad – www.parallelgraphics.com/products/vrmlpad
- polytrans – www.okino.com/products.htm
- maya – www.alias.com
- 3dsmax – www.discreet.com
- houdini – www.sidefx.com
- bodystudio – www.reiss-studio.com
- poser – www.curious-labs.com

In addition to the modeling tools already mentioned before, there are many additonal tools and add-ons, such as *houdini* for procedural modeling, *bodystudio* for importing poser models in maya, 3dsmax and other tools, and *poser*, a somewhat outdated tool voor modeling humanoids, with a large collection of ready-made feature material.

Alias Wavefront Maya

- information – www.alias.com
- tutorials – www.alias.com/eng/community/tutorials
- community – www.alias.com/eng/community

A high end 3D modeling tool, with a respectable history and a large community of users. It is in the high end price range and requires significant effort to master.

Discreet 3D Studio Max

- information – www.discreet.com
- tutorials – www.pixel2life.com/tutorials/3dsmax.php?tut=16
- vrml – www.dform.com/inquiry/tutorials/3dsmax

Popular within the game community, *studio max* which includes *character studio* appears to be somewhat more straightforward than maya.

technology

Again, the technology overview is certainly not exhaustive. There are many commercial game engines that are well worth looking at when you engage in a real project. I have included a limited number of open source libraries and toolkits to provide you with a starting point for further exploration.

DirectX SDK 9

- information – www.microsoft.com/directx
- show + 3d – msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwmt/html/vmr_d3d.asp
- SDK – msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/directx9cpp.asp
- frames – www.jkarlsson.com/Articles/loadframes.asp
- animation controller – www.jkarlsson.com/Articles/animation.asp

Direct X is an advanced, yet complicated multimedia platform. The managed code version is significantly less powerful than the C++ version. As indicated in section 4.2 there is a great many of books about DirectX. Some helpful online tutorials are listed above.

Wild Tangent

- information – www.wildtangent.com
- developers – www.wildtangent.com/developer

Wild Tangent is very appropriate for developing games. It provides conveniece layer around DirectX 7, and enables applications to be run via a Web browser, by deploying the COM interfaces for DirectX. It allows for authoring content and dynamics in Javascript and Java. However, also the original X meshes, the file format for DirectX, can be used.

Virtools Software Suite

- information – www.virtools.com

Virtools is announced to be a *comprehensive development platform, for games, virtual reality/simulations and marketing/multimedia applications.*

OpenML

- information – www.khronos.org/openml

OpenML might be the candidate platform for those that wish to develop platform-independent (read non Microsoft windows-tied) multimedia applications. It is *a royalty-free, cross-platform programming environment for capturing, transporting, processing, displaying, and synchronizing digital media - including 2D/3D graphics and audio/video streams. OpenML 1.0 defines professional-grade sample-level stream synchronization,*

OpenGL extensions for accelerated video processing, the MLdc professional display control API and the ML framework for asynchronous media streaming between applications and processing hardware.

open source technology

- plib – plib.sourceforge.net
- OpenSceneGraph – www.openscenegraph.org
- OpenSound – www.cnmat.berkeley.edu/OpenSoundControl
- ARToolkit – artoolkit.sourceforge.net
- Mixed Reality Toolkit – www.cs.ucl.ac.uk/staff/rfreeman
- OpenNap – opennap.sourceforge.net
- ImageMagick – www.imagemagick.org
- *cygwin* – www.cygwin.com

There are many open source software toolkits and libraries. My experience with these is mixed. Anyway, when you start working with these make sure that you have sufficient programming skills, and patience. But then the results might be better than you could obtain with more expensive commercial technology. If you run Linux, then open source is probably the only way to go. For windows users, with a unix background, there is *cygwin*, which is a linux-like environment for windows.

XML

- XML Entities – tech.irt.org/articles/js212
- W3C – www.w3.org/Style/XSL
- resources – www.xml.org/xml/resources_cover.shtml
- saxon – saxon.sourceforge.net
- online tutorial – www.zvon.org/HTMLonly/XSLTutorial/Books/Book1/index.html
- Xeena XML editor – www.alphaworks.ibm.com/tech/xeena
- X3D Edit setup – sdk.web3d.org/spring2002disk2/tools/X3D-Edit/index.html

For XML there is a number of generic editors, such as Xeena, which has been adapted for X3D, see appendix B. There are also XSLT processing tools, such as *saxon*, which is the only one I have experience with.

Java

- information – http://www.javasoft.com
- art with Java – http://java.khm.de
- java media framework – http://java.sun.com/products/java-media/jmf/2.1.1/guide/JMFTOC.html
- slide show – http://developer.java.sun.com/developer/technicalArticles/Threads/applet/index.html
- basics – http://developer.java.sun.com/developer/onlineTraining/Programming/BasicJava1/compile.html
- tutorial – http://java.sun.com/docs/books/tutorial/index.html
- advanced – http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/
- sound API – http://java.sun.com/products/java-media/sound/samples/JavaSoundDemo
- imaging – http://developer.java.sun.com/developer/technicalArticles/Media/AdvancedImage

Java is the programming language of choice for many computer science curricula. It is a well-documented, relatively easy to use language and the java framework provides a rich collection of libraries. There is also a version for mobile platforms.

student multimedia facilities

To conclude this overview of resources, tools and technologies, I have included a brief description of the student facilities we have for multimedia work at the Vrije Universiteit, spring 2005.

computers 14 fujitsu siemens scenico P320, AMD64 3400+ MHz, 1G memory, 80 GB serial ATA disk, 6 x USB, XFX Geforce 6600 GT 128 Mb AGP, dual display, 2 LCD monitors.

software

- Parallel Graphics VrmlPad – site license
- Alias Maya Complete (5.0 & 6.0) – 10 floating licenses
- 3D Studio Max 7 – 15 floating licenses
- DirectX9c SDK – http://www.microsoft.com/directx
- WildTangent WebDriver & SDK – http://www.wildtangent.com/developer
- CG Toolkit – http://developer.nvidia.com/page/tools.html
- RenderMonkey & SDK – http://www.ati.com/developer/rendermonkey
- Llustrator & Photoshop CS – http://www.adobe.com

There is no need to say that this is not the end of the story. More software will be installed, among which *virtools*, hopefully soon. And whenever the opportunity is there, we will no doubt upgrade to more powerful hardware as well!

E. write an essay!

Even when you prefer to do practical work, it might well pay off to take a step back, reflect on your approach and and study one aspect of multimedia in more detail. When you plan to work in an academic situation, it is very likely that at some point you must report about your work and provide some theoretical context to it. These few closing paragraphs are meant to give you some hints about how to approach writing a paper or report.

Independent of how you tackle the process of collecting material, organizing notes and writing it all down, keep in mind that the end result must consist of:

outline

title – *indicating the topic*
name – *to tell who you are*
abstract – *giving the 'message' of your efforts*
introduction – *clarifying the approach and structure*
background – *explaining the context of the subject*
sections – *to elaborate on the subject*
related work – *characterizing related approaches*
conclusion(s) – *summarizing the main point(s)*
references – *listing the literature you consulted*
appendices (optional) – *providing extra information*

It is surprising how often students forget, for example, an abstract or a proper introduction. Often the familiarity with the material, built up when working with it, seems to make them forget that for the reader these items are important and cannot be missed to grasp the point(s) of their efforts. Also, I wish to note that, although the discipline of giving references is in computer science much less strict than in, for example, philosophy, sufficiently clear references are necessary for the reader to check and verify your claims.

AS I already indicated I do not wish to elaborate on how to gather material, how to organize your collection of potentially useful notes, or how to convert these notes into readable text. Rather, I wish to to discuss the distinction, or tension, between form and content. Form, I would say, is determined by the perspective from which you approach the material and the goal you set yourself when writing the paper or report. Possible perspectives, or if you prefer forms, are:

perspective(s)

- review/background – *sketch perspectives, history, viewpoints*
- case study – *analyse assumptions, gather empirical data, and explain!*
- technical analysis – *technology-oriented, work out the details*

- formal study – *clarify in a formal manner, conceptualize and formalize*
- tutorial – *explain for the laymen, but do it very good*

To be clear, the phrase perspectives as used here is only vagely related to the use of perspectives when used to introduce the parts, where it meant to indicate the scientific discipline or point of view from which to look at a particular topic.

Content, as opposed to form, may be characterized as the collection of possible subjects, which in the area of multimedia include authoring, digital convergence, standards and information retrieval. Obviously, some subjects are better matched with particular forms or perspectives than others. For example, a formal study is suitable for discussing standards, but, to my mind, less so for explaining multimedia authoring. To get an idea of how I look at the problem of reconciling form and content when writing a paper about multimedia, consult the matrix:

	authoring	convergence	standards	retrieval
review/background	-	++	++	+
case study	+	+	+	+
technical analysis	-	++	++	++
formal study	-	-	++	-
tutorial	–	-	?	-

You may wonder why I don't think of tutorials as a suitable form for writing about multimedia. Well, in fact I do think that the form of a tutorial is an excellent way to write about multimedia technolgy, but it is not a very rewarding form for getting academic credits. When you want to be an acdemic, you'd better learn to write a technical analysis or case study. However, by that time perhaps the scientific paper generators[4] might have matured to the extent that writing has become a superfluous activity.

[4] www.pdos.lcs.mit.edu/scigen