1

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.

http://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.