A VRML-based user interface for an online digitalized antiquarian collection

Pierre Cubaud, Alexandre Topol
Centre d'Etudes et de Recherche en Informatique
Conservatoire National des Arts et Métiers
55, rue de Turbigo 75003 Paris
tel. : (33) 01 40 27 22 47 fax : (33) 01 40 27 22 96
{cubaud, topol}@cnam.fr

ABSTRACT

This paper presents a prototypal 3D application for the access to a digital library. It is mainly written in Java and relying on the 3D engine enclosed in a VRML browser. By describing the basic interactions we have included in our prototype and how they are implemented, we show that, with the actual Virtual Reality Modeling Language specification (VRML97) and even with the new Extented 3D (X3D) draft, it is too difficult to create some highly interactive virtual worlds. The effort needed to program basic behaviors with a script language raises some important problems. For solving them, we propose to work on adding some generic behavior nodes into the VRML specification.

Keywords

VRML, Navigation, Interaction, Behaviors, Digital Library.

1. INTRODUCTION

Digital library (DL) technologies have benefited during the last decade from the impressive increase of digital data capture, storage and transmission capabilities along with the consequent fall of their cost [8]. The widespread use of the World-Wide Web (WWW) also enables digital libraries to meet a very large, international, population. Relying on digital resources for academic researches is now a reality for many scientists, but it is usually restricted to journals, acts or bibliographic databases. For humanistic studies, a step forward is the digitalization of the historical sources themselves. In France, the digitalization project of the Bibliothèque Nationale de France (BNF) opened this trend. The BNF WWW server (http://gallica.bnf.fr) has demonstrated the feasibility of diffusing a large, image-based, digital repository over the internet and time has come for smaller academic institutions to undertake a digitalization program and a public diffusion of their own antiquarian collections.

Started in Jan. 1998, the "Conservatoire Numérique des Arts et Métiers" (CNum) is an internal, self-funded project of three CNAM services : the library (which holds a very important collection for French history of science and technology), the research center for history of technology (CDHT) and the computer science research laboratory (CEDRIC). The editorial team also includes history of science specialists from other european institutions. The first digitalization batch has been parts of the Sartiaux donation, 50 titles about electricity and general physics from the XVIIth to the XIXth century. This digital library is accessible online since feb. 2000 with a WWW-based interface (http://cnum.cnam.fr). The digitalization of Turgan's Grandes Usines - a very useful source for XIXth century european technology - and of the first thirty years of the periodical La Nature will double the total number of digitalized documents (approx. 60 000 pages) in the forthcoming months. A full description of the CNum software architecture can be found on the server.

According to emails sent to the webmasters, this straightforward WWW service seems to meet users needs. It requires only a low computational power for the users workstations and the size of the digitalized books pages is small enough to be downloaded with an average dial-up internet connectivity. We feel however that this situation may change in the near future. High bandwidth infrastructures for the internet are mushrooming between and within all major universities (where are located most of the CNum users). Low-cost 3D graphic hardware, combined with high resolution screens are commonly included in today's personal computer and will therefore become the dominant configuration for our users in the next two or three years. New visualization and interaction methods for on-line digital libraries must therefore be investigated.

We believe that 3D interaction can provide the three main functions of an on-line DL user's interface : (a) catalogue browsing and searching, (b) navigation within the selected documents, and (c) annotations and bookmarks archiving. Let us first describe how these three steps are handled within the CNum. Fig. 1 is a screen-shot of a probable user session, working on Pascal's arithmetic machine. Browsing through textual lists is the only method offered to the user for evaluating the DL corpus (fig. 1a). A book content (chapters and plates) is sequentially described in another window. The user opened multiple digitalized books pages (fig. 1b). He/she also compared the selected documents with findings from other related web services (such as the ABU french texts repository [3], also hosted by our laboratory since 1993 - fig. 1c). No authoring tool is provided and CNum users rely on their local bookmarking, archiving and word processing facilities (fig. 1d).

Since 1998 we settle down to the job of experimenting 3D interfaces for each of the previously described functions. Until

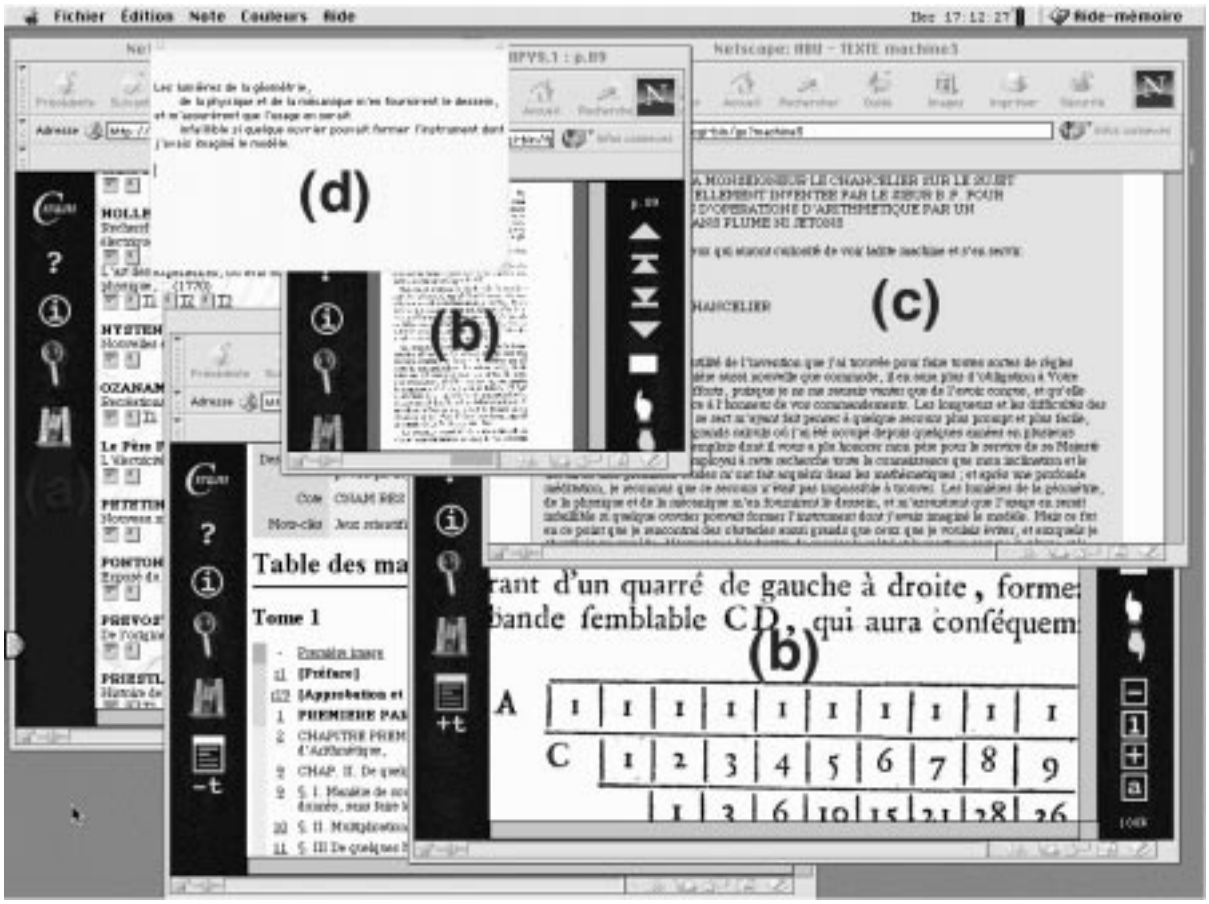

Figure 1 : Screenshot of a working session with CNUM

now we focussed on the benefits for the two former ones. One of our main concern was to pass beyond the unnecessary (because limiting) constraints of the real world (like gravity or buildings). Furthermore, in the particular case of a 3D interface for accessing a digital library, a desk is not needed to read books neither shelves to arrange them. However some real metaphors can give a precious help when creating a virtual copy of a real library. Patrons that already know a particular library can use this knowledge to find quickly books in shelves. This kind of interface like [7] may not apply for accessing an antiquarian collection since only the library staff can access to the books kept in a special reserve.

We feel that such an "hostile" environment for the reader should greatly benefit from virtual reality techniques. It is widely acknowledged that libraries patrons can discover interesting bibliographical information by casual glances to the book collections. In agreement with A. Manguel [9] who thinks that a book is distinguishable from others by its cover or shape as much as its authors or title, we decided to use the pictures of the book's back to offer a visual information. In [4], we have experimented on-the-fly generation of VRML scenes of textured books arranged in a cylinder or a sphere in order to minimize the navigation. The re-ordering of the bookshelves in accordance with the request made by the user proved to be a powerful way for titles selection. For the reading task itself we rely on the web forager interface described in a paper that pioneered the study of 3D immersion of 2D document browsers [2]. Focus+context environments like [10] overtake the common problems of 2D interfaces for visualizing large data sets. However, only 3D interfaces allow the rotation of the point of view (which is much more natural than panning) and the organization of the data in perspective.

In section 2, we present and describe the design choices and the navigation management. The application's architecture and implementation is detailed in section 3. In section 4 we discuss about some of the problems raised by a Java-extended VRML world and propose a solution to investigate for solving them. Section 5 introduces some of the future works this work permits.

2. DESIGN PRINCIPLES

With our VRML-based interface, an user can organize its 3D environment in two different ways. The first one is based on the standard navigation functions included in all plugins. The second method is supplied by the interactions we have allowed in our prototype. The implementation of those different actions described below will be discussed in the next section.

Navigating in a 3D content can be difficult for inexperienced or occasional users. To ease the interaction, users can manipulate the books of our 3D digital library without moving the point of view. Compared to real or "virtually real" libraries, no moves are required to find the shelves containing the books of interest. It avoids getting lost and/or wasting too much time with hazardous movements because of the lack of experience with 3D navigation. We consider that it is useless to faithfully reproduce the various steps of a book selection. In a virtual world there is no need to

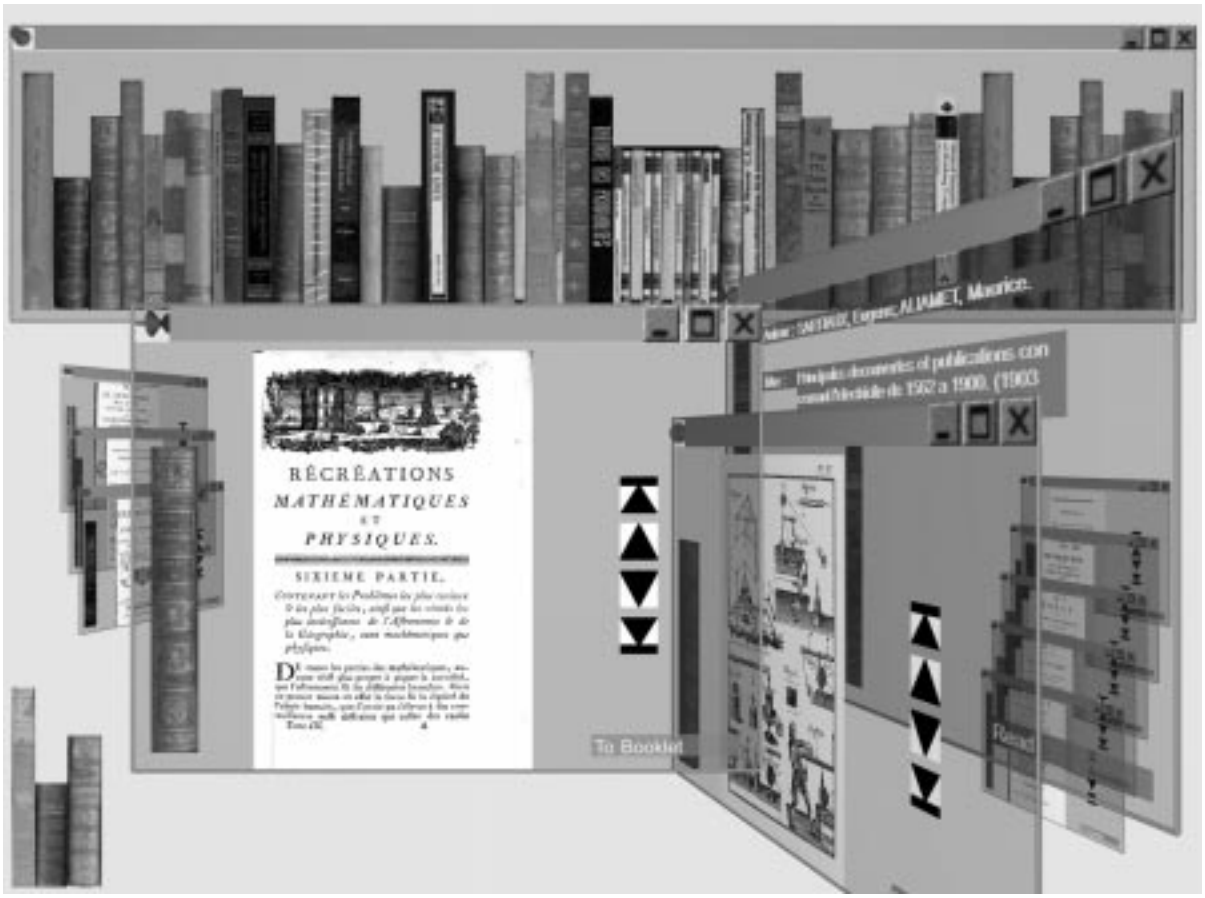

Figure 2 : Screenshot of the 3D prototypal application

"walk" to a bookshelf then "turn" to face the books for reading their titles before choosing one. On the contrary, we chose to speed this process up by minimizing the moves.

A fixed point of view would have been however too limiting and frustrating for experienced users. We want to allow them to move the camera in order to organize their 3D workspace. In [6] the viewpoint is fixed so that users only have to focus on interaction tasks and not navigation ones. Even if it is complex to deal with both tasks, we believe that experienced users are capable of doing it. The compromise we made was to authorize these users to navigate with the standard metaphors but without forcing less-experienced ones to use them. By following this principle of different interaction levels, we also chose to use container objects that inherits from 2D interfaces to ease the interaction.

2.1 Plugin Navigation

The most useful navigation metaphors are "walking" or "flying" forward and backward. They correspond to interactive zoom in and zoom out allowing users to see books or pages with more or less details. The other traditional navigation metaphors like "pan", "turn", "study" can be used to organize the 3D environment. These movements are mostly like switching-tools for moving to different part of the space where the user can (or has) spread books. Viewpoints, that are also managed by the VRML plugin, could also be created and deleted automatically to offer another switching-tool to each region containing a set of information.

Whatever the metaphors are intended (or used) for, the real-time aspect of the interaction is of great significance. The zooming process allows users to link dynamically their navigation with the filtering of the data. The fluidity of interaction helps the user to keep working at a perceptive level. On the contrary, when there is an important elapsed time between two frames, the cognitive processor is also required to understand and anticipate what is perceived as jumps. All benefits provided by visual clues can, in this case, be annihilated. Robertson et al. [11] have described continuous animation as a manner to provide object constancy and a way to replace this cognitive effort by a simple perceptual activity.

2.2 Interactions

Each element in the scene is included in a transparent 3D traditional window (fig. 2). The two terms "3D" and "traditional" seem contradictory at first sight, since traditional windows are known to be 2D shapes. But they express the fact that we are using 2D windows in a 3D world (therefore manipulated with 3D interactions). We chose a transparency appearance because it is a well known clue for depth perception and because it helps for the retrieval of hidden objects.

A unique window giving access to the entire collection constitutes the initial scene. Each book in this collection-window is described as a rectangle textured with the scanned picture of its back. The size of each `IndexedFaceSet` geometry is computed to be proportional to the texture's size associated. Deformations are

therefore avoided and the book's format is respected. At first glance one can evaluate the number of volumes available and its publication date with the binding style. Each book catches events to deal with the user's interaction. A roll-over event creates a tooltip underneath the book giving a short bibliographic data. A mouse click opens the book in a new window.

In our VRML scene, all graphical elements are basically flat: a book is represented by a textured rectangle, information is given by text labels and pages are also rectangular. They all can be inserted in a 2D container. A flat window offers sufficient interactions for the manipulation of this kind of content. Any user should obviously find by himself the possible actions for managing a 3D window:

- moving it by dragging and dropping it with its title bar (movements of the mouse are mapped in local coordinates of the window),
- minimizing it (in our case it is more like putting it away automatically in perspective than iconifying it),
- maximizing it (by computing the rotation and position of the window in order to align it with the view frustum and fit it entirely with a maximum scale),
- using a pin (in some WIMP interfaces like OpenLook when these pins are active they forbid windows moves. The 3D generalization is to prohibit the user from moving the window with its title bar and to annihilate the movements of the viewpoint for the pinned widows in order to produce an effect of immobility),
- closing it with a traditional push button.

A window is truly iconified (as opposed to minimized) like those one can see in the bottom-left corner of the screenshot (fig. 2) when they do not appear anymore in the view frustum. These icons have the same appearance than the book objects in the collection window. This visual clue given by the book's texture helps the user to remember which ones are iconified. The user should also recall how to interact with them since the behaviors are implemented much like the ones for the books in the collection window:

- a ToolTip appears when the mouse is over the "icon",
- when the "icon" is clicked the corresponding window is maximized.

When a window enters back in the view frustum, its icon is deleted. These icons help the user to remember which books are already opened even if they are not in the current view.

A book-window has two distinct and concurrent parts that are activated with a `Switch` node. The unique common element in these two parts is the book texture that helps visually to recall which book is opened. The first one is the booklet composed with different 2D elements (that one can see in a window in perspective on the right-hand side in fig. 2):

- the first label (at the top) gives the name of the author(s),
- the second one (underneath) gives the title,
- the label(s) on the bottom-right corner are "links" (a text label + a `TouchSensor` node) to switch to the reading part of the window. One link is given by volume (four volumes in this case).

The second part of a book-window is for reading. All booklet elements are replaced by the reading ones (illustrated in the two foreground windows of the screenshot in fig. 2):

- the current page of the chosen volume described by a `IndexedFaceSet` geometry and textured with the picture of the scanned page,
- standard buttons (from top to bottom : first page, previous page, next page, last page) to browse through sequential documents,
- a "link" to come back to the booklet part.

2.3 Design discussion

Using 3D models instead of 2D components might have been the best solution for a better visual effect. A real "book" metaphor could be more practical and comprehensible at first sight. Interactions with this object should be easier to understand and could be much more powerful. For example we could provide a tool for rolling automatically pages like the one given for interacting with Web books [2]. It would be an intuitive way to acquire the structure of a book. As with a real book, this could also be a quick way for finding a page of interest. However, other interactions around the book (for moving it, closing it…) would not be as easy for us to implement and for users to understand. This would require adding widgets near the book (therefore modifying its real appearance) or to use rotating menus like the ones described in [12] or included in [6] (that users may not know how to use).

Interacting with 3D objects could lead to a sophisticated but powerful interface. We could experiment all sorts of interactions. However we were aiming at doing the most usable application. So in this particular context we do believe that using 2D components, instead of some 3D metaphors, is a valid choice. Every interaction mechanism is easily understood thanks to previous experiences acquired in 2D interfaces. Every widget (hence every action associated) in our prototype is also directly accessible (no disturbing items are popping up depending on the user's actions).

3. APPLICATION'S ARCHITECTURE

After presenting the elements of the interface, we now explain in detail how interactions are implemented. Whatever the purpose of an interaction is, the same scheme is always used. First a VRML part describes an object (with maybe different elementary shapes) and the sensor activated by this object. The sensor recovers user's events. The routes to the `Script` nodes, the attributes of which are also given in the VRML file, transmit the events. With some `Sensor` nodes an object is not required. For example the `ProximitySensor` sends events based on the position of the point of view in the scene.

Most of the interactions we have implemented in our prototype just modify the VRML scene graph (adding a new transformation in a `Transform` node or changing the diffuse color of shape). Some of them also have to add and delete VRML content in the scene graph in accordance with the user's actions. In this case they also add and delete routes from and to this content if needed by the application. These particular interactions require the use of scripts since the VRML specification [14] does not include a creation mechanism.

As shown in fig. 3, the prototype's architecture has three different blocks each one managing a different aspect. We will describe each block separately and we will tell how the main operations are implemented. The overall application is mainly constituted with Java scripts as we can see in table 1. We will discuss the problems of such ratio (over 93%) of scripts in the next section.

MainScript	71
MainClass	83
Window3D	444
Window3DScript	141
BookScript	614
PageScript	158
Total for script files	1511

Main.wrl	37
Book.wrl	73
Total for VRML files	110

Table 1: number of lines in script and VRML files

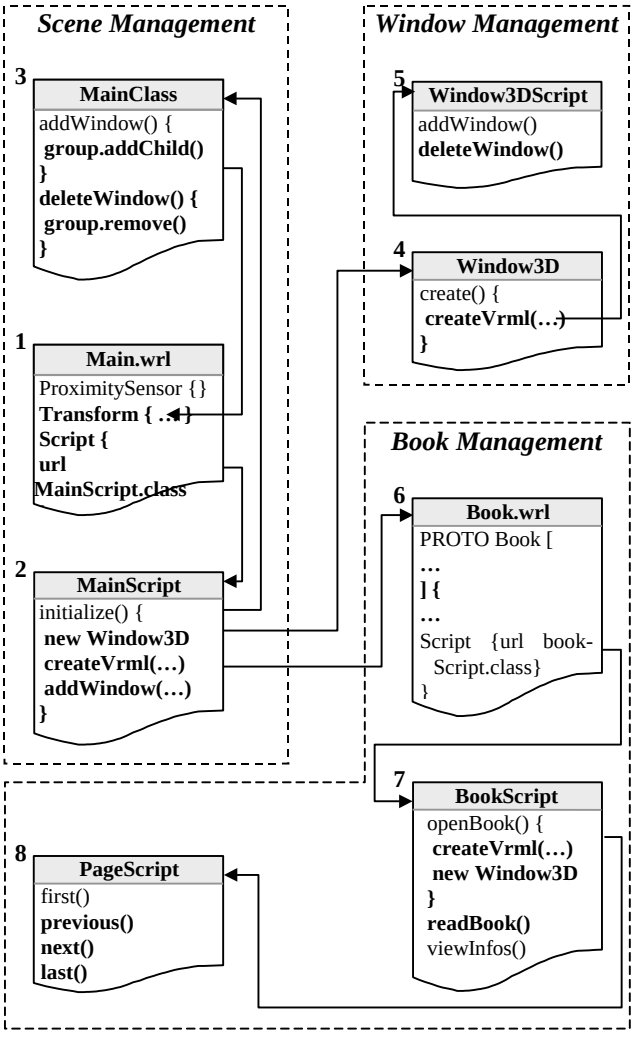

Figure 3 : Architecture of the VRML prototype

3.1 The Scene Management Block

The main VRML file (given as the URL to access this scene) is only a skeleton for our application. Usually a main file contains shapes to be rendered and/or external files (`Inline` node or `EXTERNPROTO`'s) to be loaded. In our case, it does not include geometry nodes in a direct or indirect way. It is only composed of six nodes with no hierarchy (they are all first level nodes). The first three ones are bindable nodes that are not even required for the application to work:

- a `WorldInfo` node to set some information,
- a `Background` to set the background color,
- a `NavigationInformation` node to specify that "fly" as the default navigation metaphors and to tell that a headlight must be used.

The three next nodes are more important since they are necessary for adding shapes to this initially empty scene and for dealing with the navigation:

- a `Transform` node in which new objects will be added and in which objects to be destroyed will be searched,
- a `ProximitySensor` node with a large size (so it is always active) to retrieve the movements made by the user,
- a `Script` node to declare the `MainScript` script.

Two groups of two routes are also declared for managing the viewpoint's position and the orientation changes captured by the `ProximitySensor`:

- one group to route to the `Transform` node in order to inhibit all user's moves,
- one group to route to the `MainScript` script that uses these movement changes to compute and apply the opposite ones to every mobile object (*i.e.* not pinned).

A pinned object (that must remain immobile) will only be translated and rotated when the viewpoint is changed. The common manner to make still objects, as shown in fig. 4, is to route the `ProximitySensor`'s events to the main `Transform` node of the scene that contains all objects. With this only mechanism, objects stay still while the user navigates. However we only want the pinned objects to stand immobile so the `MainScript` script computes the opposite movements and applies them to the main `Transform` node of each unpinned window.

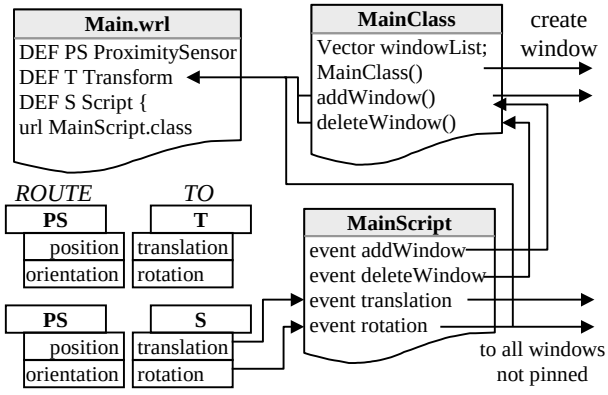

Figure 4 : Details of the Scene Management Block

The `MainScript` script also processes the events used for adding and destroying windows. To add a window it calls on a method of the static `MainClass` class. For destroying one it uses the return value of another method of `MainClass` as an index for deleting the right children in the main Transform node. Last but not least `MainScript` initializes `MainClass` in order to

transmit the browser class and its own instance, both needed in every other Java class to create or destroy some VRML content.

Since `MainClass` is a static class every public attribute and method can be accessed directly without instantiation. To keep a track on all opened windows a `Vector` utility class is used. This vector is used whenever we need to loop on each window (to know which one is out of the view frustum for example or which one is pinned). The first 3D window (containing the entire collection) is created in the `MainClass` constructor. There, is also set the window counter indispensable for giving a window a unique identifier. This counter is incremented for every new window created. Since windows could have been deleted a window identifier is not necessarily equal to the index at which the window is in the children field. This is why we need a method returning the real index of a particular window number so we can remove it in the children field of the main `Transform` node.

3.2 The Window Management Block

The Window management block is composed of two Java classes. The first one is the `Window3D` class that is responsible for the creation of the window objects. `Window3Dscript`, the second class that inherits from the Script class takes care of all possible interactions with a window.

The `Window3D` constructor only assigns to the new instance of a window its unique identifier held by `MainClass`. The constructor does nor create (*i.e.* show the window). The creation is effective after calling the `create` method, which takes 5 parameters: the three coordinates giving the window's position, its width and height. To create a new window different operations must be performed before it is fully operational in the browser:

- the geometry, appearance, sensors and interpolators of its different elements are given in a VRML expression, which is parsed with the `createVrmlFromString` method of the `Browser` class. The new nodes are then inserted as a new children of the main `Transform` node (in `main.wrl`) accessible by the MainClass class,
- the "normal" event routing from sensors to interpolators for automatic animations are added by telling the browser that new paths are added (with calls to the `addRoute` method of the Browser class),
- the behaviors are defined by routing some events from the sensors to a script added as a `Script` node in the generated VRML expression.

The `create` method creates an empty decorated window. To add objects in a window the `addObject` method is provided. The other methods in this class are used by `MainClass` to get the current state of a window (iconified, pinned on or off).

A Window3D has two different appearances (defined in a `Switch` node): one is used when the window is in the volume frustum and the other one when it is out. When the window is visible different states can change its appearance: maximized or minimized, translated with its title bar or not, pinned or not. For each state we have defined different `Transform` node to specify each transformation independently. `MainClass` uses the first one to translate and rotate the window if it is not pinned when the user changes the point of view. The second one holds the transformations for the minimized or maximized state. And the third `Transform` node is reserved for the translation after dragging and dropping the window's title bar.

`Window3Dscript` is the script declared in the generated VRML expression. Its purpose is to take care of all interactions with the window, not its contents. Events raised by the sensors are transmitted to the script by the routes added by the `create` method. Two different ways were used to implement interactions depending on the type of the interaction:

- a direct modification in the scene graph,
- or an `eventOut` sent to be caught by an element of the window.

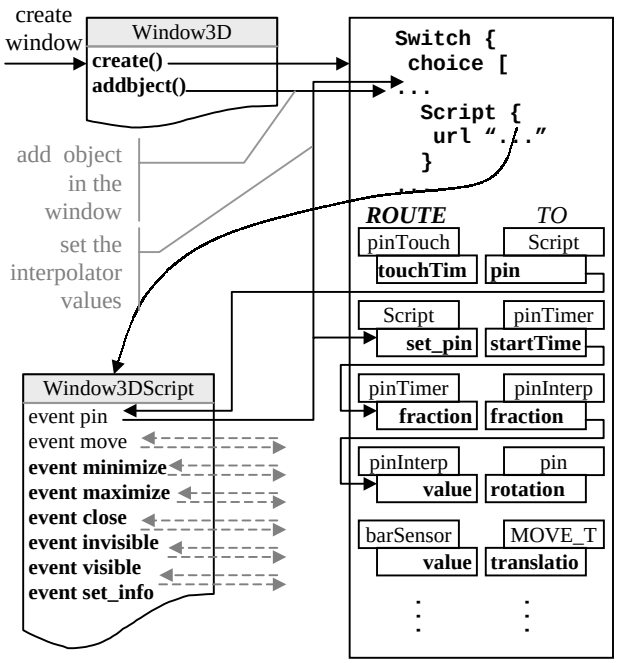

Figure 5 : Details of the Window Management Block

In fig. 5 only the pin interaction is detailed since the overall details would require too much space. However eight events are implemented for a window and works basically the same way:

- mouse clicks on the pin are routed by a `TouchSensor` to the script which computes :
  - the two values of the `RotationInterpolator` for the animation (depending on the state of the pin the rotation is not the same),
  - enables or disables the `PlaneSensor` associated with the title bar,
  - fires the animation with a TimeSensor,
- drag and drop on the title bar (translations) are sent from the `PlaneSensor` to affect the translation defined in the `MOVE_Transform` node,
- mouse clicks on the minimize, maximize and close buttons (three different events) are also driven from `TouchSensor`'s to the script which computes animations and performs the right actions,
- visibility and invisibility events of the script are respectively activated by the `enterTime` and the `exitTime` events of

a `VisibilitySensor` associated with the window. To render either one appearance or the other the script sends an event to assign a new value to the `choice` field of the `Switch` node,

- a `set_info` event is activated by a `TouchSensor` to print the tooltip underneath the iconified book when the mouse is over it.

3.3 The Book Management Block

The books are created when the collection-window is created. The books in this window are described using an external VRML PROTO named `Book`. The prototype (in `book.wrl`) builds a book from basic information:

- its position and size,
- its texture for the appearance,
- its information for the booklet,
- the URL's to access the pages of each of its volumes.

An empty `Transform` node is included in the PROTO in order to add the tooltip when needed. A `TouchSensor` node and the `BookScript` script are also included and events are routed from the first to the second. The rollover behavior is achieved with the `isOver` `eventOut` of the `TouchSensor`. The book is opened when a `touchTime` eventOut is caught and routed to the script.

When the `isOver` `eventOut` is `true` the script creates the tooltip if it is not already present (if the `Transform` node is empty). When the value changes to `false` the tooltip is deleted. This effect is achieved by removing the children in the `Transform` node if present. These changes in the VRML scene graph are immediately reflected on the screen by the VRML plugin.

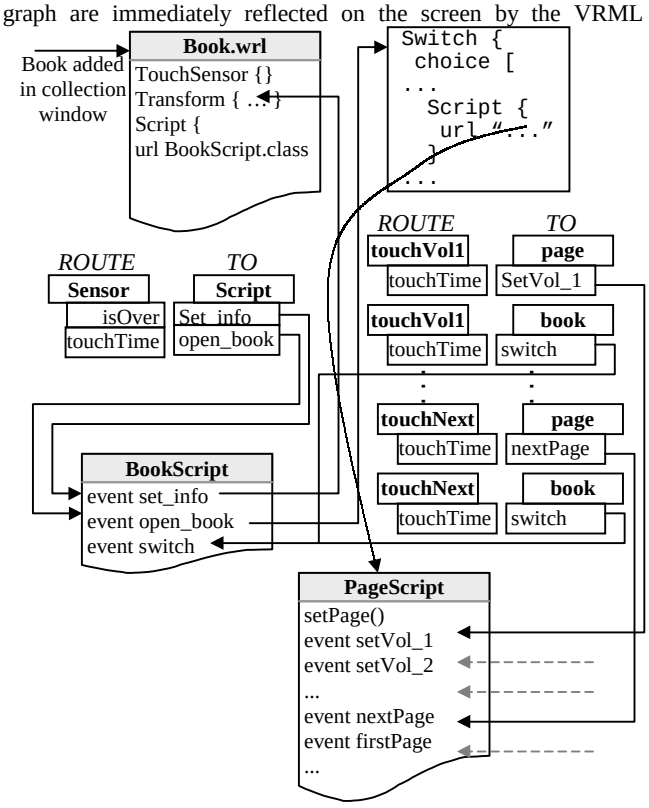

Figure 6 : Details of the Book Management Block

When a click occurs on a book the script is aware of it. The actions performed are: creating a new window, .adding the VRML expression for this particular book (including `PageScript` in a `Script` node) and creating routes for interactions (from the content of the book-window to `PageScript` events). As we said before a book-window has two different parts but the interactions for changing from one to the other are pretty simple. The `BookScript` script changes the value of the `whichChoice` `exposedField` belonging to the `Switch` node.

In the booklet part when a mouse click occurs on one of the volume labels, the value of the `whichChoice` field is set to 1 in order to display the reading part. The current page (the first one by default) of the corresponding volume is loaded by the `getPage` method of the `PageScript` script called by the `setVolume` `eventIn`.

In the reading part two different interactions are provided:

- when a mouse click occurs on a button provided for the navigation through pages the desired action is performed by routing a `TouchSensor` (associated with the label) event to `PageScript`,
- when the mouse click occurs on the booklet label the display is switched back to the booklet part by assigning 0 to the `whichChoice` `exposedField`.

The `getPage` method retrieves the URL of the picture of the desired page by calling a CGI script on the CNUM server. The URL returned is used in the `url` field in the `ImageTexture` node to load the new page. The filename of the picture in the URL includes its width and height so the rectangle on which is mapped can be resized.

4. DISCUSSION

The overall structure of the VRML scene graph is basically entirely generated by scripts and will look like this:

```
DEF PS ProximitySensor [...]
DEF T Transform {
 children [
  Switch {                  # first window
   choice [
    DEF MAIN_Transform Transform {
     children [
      DEF MINMAX_Transform Transform {
       children [
        DEF MOVE_Transform Transform {
         children [         # visible app
          Switch {
           choice [
            Group { [...] } # booklet part
            Group { [...] } # reading part
           ]
          }
         ]
        }
       ]
      }
     ]
    }
    Transform {
     Children [...]         # invisible app
    }
   ]
  }
  Switch { [...] }          # second window
```

```
  [...]
 ]
}
DEF S Script {
 url MainScript.class
}
```

The lines in bold type are the ones declared in the `main.wrl` file. With the ones used from the book prototype, also in VRML syntax, they only represent 10% of the whole scene graph. The several scripts generate the other 90%. The Java source code for those scripts also represents 93% of the coded lines. Those numbers show that the scripts are the most important part of the application. Hence, with this kind of interactive scenes the contribution of VRML as a description language is poor.

This VRML scene can be seen as a Java program using the 3D engine enclosed in the VRML browser. As if we had used Java3D to specify a scene graph, our application lies on the VRML scene graph description to manage 3D contents, based on the user's interaction. A powerful aspect that remains a VRML property is the modularity. We can transform the `main.wrl` file to declare the whole scene as a reusable prototype within other scenes.

4.1 Problems Raised

The interactions we have added in our prototypal application are quite simple. However, the great amount of scripts required to implement these behaviors is really surprising. Telling that VRML+Java is a useless couple to create interactive scenes is not the point. We do believe that every imaginable interaction can be coded this way, including a collaborative environment like [6]. Thanks to the numerous Java classes, the Java scripts also authorize programs to deal with operations that are not directly related to the VRML scene. For example we have implemented the retrieval of the pages' URL by sending an HTTP request to the CNum server. However, considering the number of script lines required for these simple interactions, one can think that a more complete and rich system could be unmanageable.

Even if the scripts are used primarily to modify the VRML scene graph (by inserting or deleting a parsed VRML expression with the `createVrmlFromString` method) they can not be considered as being part of the VRML scene since they can not be edited. Nevertheless a Java source code can be obtained from a compiled Java class. For a highly interactive commercial scene this reverse engineering process is a main problem. The "intelligence" of such a VRML scene requires a hard and long work. Authors of the scene would not appreciate that other programmers use their own code and knowledge. Industries would undoubtedly use another language to settle this spying problem. Thereby, only a few highly interactive worlds are downloadable on the Internet. The one we present in this paper is one of the most complicated we have found.

To create Java scripts, authors have to understand the object oriented programming paradigm. Hence creating new behaviors for VRML objects is only accessible for programmers. Because of this, graphic designers and artists who create the more interesting scenes can not use VRML.

4.2 Solution

Because of the many scripts needed to manage complex interactions, VRML is not the right language for creating online 3D applications. Several propositions to enhance VRML were proposed but none of them resolve our particular problems. J.-F. Balaguer recommends to make VRML evolve towards a flexible run-time environment based on indirect constraints [1]. Basically he proposes to add scripts directly in the VRML file and to manage events and routes with constraints. S. Diehl has also proposed a constraint based enhancement of VRML that deals with animations and behaviors [5]. With these solutions, the scripts would be easier to maintain. However this does not solve the main problems stated above. An interactive scene will still be hard to implement and the scripts will remain the biggest part of the application. The scripts will also remain the "intelligent" part of the scene and even more public (not even compiled).

We agree that object's behaviors must be included in the scene graph as well as their appearance and geometry. However we would prefer a descriptive manner to do that instead of a computational one. All interaction mechanisms would be hard coded in the browser. This would be the best solution for all problems given above. Generic animation and interaction nodes could be inserted in the scene graph to specify the different behaviors. Some interactions in our prototype create some VRML content. A mechanism for updating a scene automatically or on user's interaction is also needed. A Post method like the one used in HTML to send a form could be used to ask the server to deliver a new part of the scene.

5. FUTURE WORKS

Like with the CNum, we are waiting for users feedback before enhancing the current prototypal application. In the HCI domain this kind of spiral approach is often used. Users needs are collected to build a prototype and the users feedback are used to enhanced it. For this second phase we should work with designers to obtain a better look and feel for the interface. Meanwhile we should be able to determine the important interactions to include in the application for a better manipulation. We could then assemble these behaviors in a set of Java classes that could be used as an API in other VRML-based applications.

More generally, we should work on determining a set of generic 3D behaviors in order to include interaction nodes in the scene graph. These 3D behaviors should operate on 3D objects but also on 2D elements. Today's situation is to render 3D scenes in 2D windows. We believe that low-cost 3D graphic cards will lead to full 3D interfaces [13]. However a 3D workbench will necessarily coexist with the huge number of 2D applications. A similar problem occurred for the handling of keyboard-based applications (such as UNIX shells) when 2D interfaces appeared. We think that the same set of generic 3D behaviors could be used as a toolkit for specifying 3D interactive interfaces.

Note for readers: the prototypal application described in this paper is accessible on our web site at the following URL: http://cnum.cnam.fr/vrml/

6. REFERENCES

[1] Balaguer, J.-F. Less Is More : The Power of Simplicity. In *proc. of VRML'99*, Paderborn, Germany, February 23-26, 1999.

[2] Card, S., Robertson W., York W. The WebBook and the Web Forager : An Information Workspace for the World-Wide-Web. In *proc. of ACM CHI'96*. Vancouver, April 13-18, 1996.

[3] Cubaud, P, Girard, D. ABU : une bibliothèque numérique et son public. *Document numérique*, Vol. 2, Hermès, 1998.

[4] Cubaud, P., Thiria C., Topol A. Experimenting a 3D Interface for the Access to a Digital Library. *In proc. of ACM DL'98*, Pittsburg, July 1998.

[5] Diehl, S, Keller, J. VRML with Constraints. *In Proceedings of Web3D/VRML 2000*, Monterey, California, ACM Press, 2000.

[6] Dumas, C., Degrange, S., Saugis, G., Chaillou, C., Viaud, M.-L., Plénacoste, P., Spin : a 3-D Interface for Cooperative Work. *Virtual Reality Society Journal*, Springer-Verlag, 1999.

[7] Fox, E., Eaton, J., McMillan, G., Kipp, N., Mather, P., McGonigle, T., Schweiker, W. and DeVane, B. Networked Digital Library of Theses and Dissertations : An International Effort Unlocking University Resources. *D-lib magazine*, September 1997. http://www.dlib.org/dlib/september97/theses/09fox.html

[8] Lesk, M., *Practical digital libraries – Books, Bytes and Bucks*, Morgan-Kaufmann, 1996.

[9] Manguel, A. *A History of Reading*. Viking Penguin Group, 1996.

[10] Robert, L., Lecolinet, E. Browsing hyperdocuments with multiple Focus+Context Views. *In Hypertext'98*. 1998: ACM Press.

[11] Robertson, G., Card, S., Mackinlay, J. Information visualization using 3D interactive visualization. *Communications of the ACM*, 36(4):56--71, April 1993.

[12] Shaw, C., Green, M. THRED : A Two-Handed Design System. *Multimedia Systems Journal*, 5 (2), ACM/Springer Verlag, 1997.

[13] Topol, A. Immersion of XWindow Applications into a 3D Workbench. *In proc. of ACM CHI'2000*, The Hague, Netherlands, April 2000.

[14] Virtual Reality Modeling Language - International Standard ISO/IEC 14772-1:1997. http://www.vrml.org/Specifications/VRML97/