Active3D: A Method for Storing 3D Scenes into a RDBMS

Christophe Cruz, Christophe Nicolle, Marc Neveu
Laboratoire Electronique Informatique et Image
Université de Bourgogne
BP 47870, 21078 Dijon Cedex – France
c.cruz@active3d.net, cnicolle@u-bourgogone.fr and neveu@u-bourgogne.fr

Abstract — ***In this paper we present the Active3D method. This method translates the objects of a 3D-scene into tables and attributes of a relational data base. Once the storage is carried out, these 3D-objects can be modified using SQL queries. Moreover, management information stored on other data bases can be associated with the 3D-objects 3D. Thus, we have developed a Web platform allowing the visualization of numerical model of a building like a 3D-scene.***

***Index Terms*—3D Relational Database, X3D, VRML, SQL**

I. Introduction

The 3D graphics computations requires both complex software and high-performance hardware. Moreover, a large number of applications needs to process more than one file at the same time. Sending these files over the Internet and displaying them at a client's computer is not possible without the use of dedicated language such as VRML (Virtual Reality Modeling Language)[4]. VRML allows to create 3D-scenes, and to navigate in these scenes using an Internet browser with an appropriate plug-in. Nevertheless, it is not easy to manipulate these files due to the complexity of the VRML structure. In order to solve this problem, a new language has been developed by the W3C [3] for building 3D-Scenes: X3D (eXtensible 3D). This language is based on XML where 3D-Scenes are represented as XML documents having tags organized in a tree structure. This particular structure allows us to manipulate and decompose the 3D-Scenes into basic elements corresponding to 3D primitives. Nevertheless, this representation doesn't allow computer graphic algorithms to manipulate many scenes at the same time or to associate semantics to 3D primitives.

In this paper we present a method for storing 3D-Scenes into a Relational DBMS by transforming X3D documents into relational attributes and tables. Our work is extended to VRML files by converting them into X3D documents. This translation into X3D overcomes the inflexibility of VRML. We use XSL style-sheets to perform the insertion and the extraction of 3D-scenes into the database. Our methodology, called ACTIVe3D, is presented in Section 2. The transitions between VRML and X3D are particulary complex, and so is the analysis of 3D-scenes for building a 3D database. The section 3 deals with our architecture for insertion and extraction of 3D-scenes into and from the database, respectively. Preliminary results are presented in the last section of this paper.

II. Approach

XML [1] is quickly becoming a universal support for representation of information on Internet. Employed in many domains, it has been developed to transfer multimedia information (text, sound, picture, etc.). The manipulation of pictures by XML is performed at various levels. At the lowest level, it is possible to insert a picture into a XML document by using a tag containing the address of the picture file. At higher levels, there are specific XML languages for modeling and handling 2D [2] and 3D [7,8,9,10] pictures. At such levels, pictures are described by a text file structured by tags. One of the main problems in the 3D modeling domain with XML is the creation and reuse of a data structure for description of a 3D-scene. Currently, there is no normalized support that permits to manipulate 3D-scenes generically. In order to solve this problem, we have developed a new architecture that allows modeling and storage of 3D-scenes by their content in a relational database. Thus, our approach allows manipulation of these scenes using SQL queries. Figure 1 shows our method of storing 3D-scenes in the database.

The main 3D-modeling language used in Internet is VRML [4] (Virtual Reality Modeling Language). Since VRML is extensively used, it is possible that there is a large number of programs and plug'ins using VRML (CosmoPlayer [5], Blaxxun3d [6], etc.). Moreover, many modelers, e.g. 3D Studio Max of Kinetix, can export VRML files. Unfortunately, a VRML file is a monolithic script difficult to manipulate. In order to solve this problem, we translate VRML files into X3D

documents [10] using tools written in Java. Based on the XML standard, X3D is recognized by the W3C Consortium and the Web3D Consortium. They are several reasons for the selection of X3D. XML is a descriptive meta-language that structures information by using tags organized in a tree. The 3D approach combined with XML offers flexibility and extensibility, which VRML cannot offer. It permits to attach information to scenes, and to create news tags without hindering the generation of new scenes. Already, many tools have been developed and become available for manipulation of XML structures.

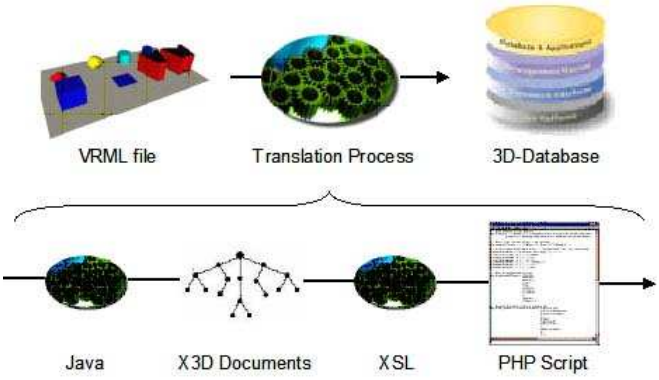

Fig. 1. Storage Method of VRML files in a Relational DBMS

Once VRML scenes are translated into X3D documents, our tool can insert them into the database. We may ask why it uses XSL style sheets [11]. XSL is a language that permits the manipulation of XML trees. The strong aspect of XSL is its capacity to distribute one XML source document to multiple display targets and to format corresponding data for each target. There are two types of formatting: the data formatting and the result formatting.

1) **The data formatting** consists of three steps. The first step consists in generating a data structure with the shape of a source tree, while associating elements (nodes of the tree) with various stylistic pattern. The second step performs an analysis of the source tree to produce a result tree or a final document. This process is based on actions specified by rules called templates, patterns or models. In the last step, the syntactic analyzer modifies the XML data using XSL formatting instructions. It produces a final tree in the form of a HTML page for displaying at a Web client's computer.
2) **The result formatting**: XSL is independent of the display language. It is therefore possible to develop a XSL syntactic analyzer for translation the tree source into other formats such as RTF, PDF; or to create either PHP [12] or Oracle scripts.

A X3D document is described by a grammar. Such a grammar defines structure of tags that are used to compose the document. XML tags formatting, along with the XML-schema language, define the grammar, which is also called a schema. A XML parser tests validity of a X3D document while comparing its structure to the grammar.

The X3D-schema is composed of tag elements organized in a tree. These tag elements, e.g. <X3D> or <Header>, will be instance nodes of the XML schema. Nodes of the tree can have descendants that will represent some of the elements declared in the schema. For example, in Figure 2, the <X3D> element cannot have descendants other than the <Header> and the <Scene> element. Rules associated with the schema are as follows:

1) An element can contain a tree structure.
2) A tree structure is always composed of elements.
3) The combined use of (1) and (2) permits to model all nodes of a tree.
4) Nodes of a tree are ordered.

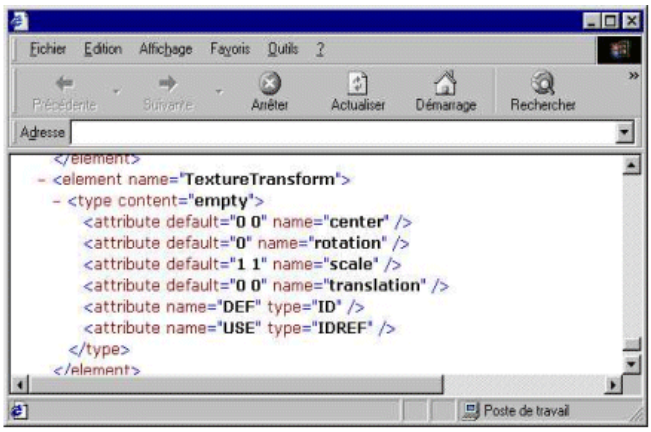

Fig. 2. the <Texture Transform> element

III. Creation of a 3D-Scenes Database

To create a database, we proceed in two steps. First, at the conceptual level, we define rules mapping from an X3D-schema to an Entity-Relationship (ER) schema [13]. Then, we translate the ER schema into a relational schema. Finally, at the physical level, we optimize the database structure.

A. Conceptual level

Several rules have been developed at the conceptual level. Due to the lack of space, we only present the main rules. For illustration, we will use the example presented in the Fig. 2.

Rule 1: Management of elements: "*An element can be composed of a tree structure.*"

This rule requires a key of an external tree for every table element. In addition, every element of the schema becomes a table that is composed of an identifying attribute (TableName_Id) and a set of attributes.

Rule 2: Management of Nodes: "*A tree structure always contains elements*"

Using this rule, a "Tree" entity and a "Node" entity are created. These two entities model the tree (Figure 3). The modeling of the table necessitates the application of exclusion

constraints. These are necessary for modeling optional characters of certain relations between elements. For example, the element <X3D> can be formalized by the following rule:

$$X3D = (Header \mid \lambda).\ Scene$$

This formalization means that all <X3D> nodes are composed of an optional <Header> element and an <Scene> element (the element λ represents the empty element). In a XML schema, the "AND" and the "OR" logical operators are specified by the attribute "order" in the tag <group>. If the attribute "order" is equal to "choice", it means that the tag <group> is of type "OR"; in the opposite case, if the attribute "order" is equal to "seq", the tag <group> is of type "AND". Initially, the tag <group> has value of "seq". The tag <group> also has two other attributes. The attribute "minOccurs" corresponds to the minimum number of occurrences (the default is one), and the attribute "maxOccurs" corresponds to the maximum number of occurrences.

Total exclusion constraints allow us to model the above grammar rules. By using the pre-defined management rules of Node and Element, we have developed the Entity-Relationship schema for our database. Our schema is composed of 103 entities derived by the Rule 1, of 2 entities derived by the Rule 2 (Entity Node and Entity Tree), and of 104 relations and 104 exclusion constraints.

B. *Physical level*

At this level, the Entity-Relationship schema is mapped onto the physical view of the database. Two main problems arise in this mapping. The first problem concerns the exclusion constraints supported by the "Node" relations (Rule 2). Their direct translation into a relational schema increases the database size (by a factor of 2). The second problem concerns the "Tree" table. It contains only one attribute (the primary key), which contains no significant information. To solve these problems, we assume that all manipulated X3D documents were validated by a program which verifies the correct syntax. This validation will be carried out by the XML parser before loading the document into the database. Thus, the relational schema can be optimized based on its utilization rather than based on the grammar of the X3D language. As a result, the "Tree" table can be dropped. Exclusion constraints are deleted at the physical level and the "Node" table is modified as follows:

Node(Node_Id, Elt_Id, Order, Type_Elt, Father_Node)

The Type_Elt attribute allows to identify linked table elements and to substituate the conceptual exclusion constraints. The primary key is "Node_Id" which represents the serial number of a node in the tree. The "Order" attribute represents the location of a Node in a list of descendans of the given element. The "Father_Node" attribute represents the ancestor Node, with which the given node is directly linked. The other tables remain unmodified. The final schema contains 104 tables.

IV. ARCHITECTURE PROCESS

After having described the creation of the database schema, we are now going to present the insertion and extraction processes of 3D-scenes.

A. *Insertion of scenes into a database*

This process is based on the use of XSL style sheets. The style sheets don't operate in the same way as the other programming languages do. They are specially defined to analyze and manipulate tree structures. In our project, we have defined two style sheets that call each other recursively. Using this method, we can browse all existing XML tree structures regardless of their size. A style sheet is defined using tags. Thus, an XSL style sheet is considered as an XML document and can be manipulated by tools for XML documents.

The above combination of X3D and XSL files generates a new PHP script. This script contains commands to establish and close the database connection. Moreover, it contains all the SQL insert commands for storing the X3D document in the database. At each step of this process (described in Figure 3), parsers are applied to check the validity of the documents (style sheets, X3D, HTML). Once scenes are stored in the database, it is very simple to manipulate them with SQL queries. Values of all scene attributes can be modified. Insertions, modifications, and deletions of 3D objects in several scenes can be performed simultaneously. It is possible to make all the attributes of the scenes evolve dynamically. For example, it is possible to dynamically modify the coordinates of a light source in a scene by simple SQL queries.

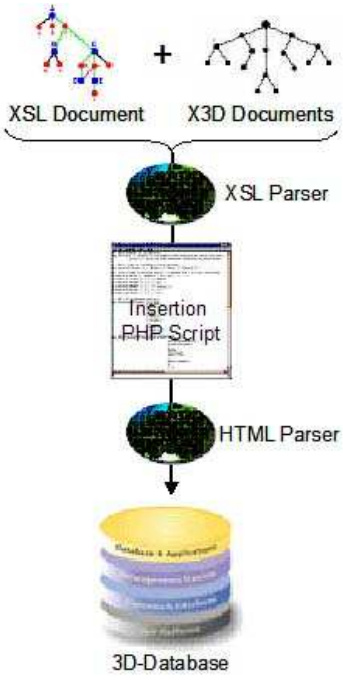

Fig. 3. Insertion Process

B. Extraction of scenes from a database

The extraction process is based on the same procedures as the insertion process. It is presented in Figure 4. there, we have defined a PHP script, which establishes and closes the database connection. Moreover, it recursively extracts the scene specified by SQL queries. The PHP script creates an X3D document that contains the result of SQL queries formatted as a tree containing tags. The resulting document is always valid. Then, a style sheet is applied to this X3D document. This style sheet maps the X3D document into a VRML file. Finally, it is possible to visualize the VRML script by a browser.

The extraction process is not limited to the display of 3D-scenes. The SQL extraction queries can also manipulate objects appearing in scenes. Inside the database, the scenes are not stored as a single file but they are decomposed into many tables corresponding to all components composing the scenes. One of the advantages of this organization is the possibility of extraction of objects shared by several scenes. Thus, many scenes can be modified at the same time. Instead of interrogating different stages one by one, it is sufficient to use one SQL query to modify objects composing the scenes stored in the database.

A scene is modeled as a tree. In addition, objects themselves are represented as trees. An object contains a number of components. For example, a table consists of a tray and four flegs. It is possible to define appearance attributes for each of these elements or for broadly speaking of all elements. In fact, a tag called «Shape» which contains the physical and visual descriptions of an object defines 3D objects: its dimensions, its color, etc. (see Figure 5). Therefore, it is possible to define a name for every "Shape" tag. With this name, it is possible to access the elements and features of an object. The modification of attributes is performed by SQL queries. In the below, we give several examples of SQL queries that allow to manipulate scenes.

```
Select Shape_id, DEF from SHAPE
where DEF is not NULL;
```

This request permits to select all Shape tags that compose all scenes stored in the database and to display certain features when DEF attributes of Shape is not Null. In the following query, we update a specific instance of "Material" tag by changing the value of its transparency attribute.

```
update Material set TRANSPARENCY = '0.3'
where material_id ='10' ;
```

The major advantage of this method lies in the fact that such a modification operates either on a selected scene or on all scenes in the databases. The automatic execution of the modification reduces the cost of manipulation of scenes.

Fig. 4. Extraction Process

C. Updating 3D-scenes using context

This section will focus on combination of several databases, witch affect scene's properties in databases. This is done thought the SQL language. Previous paragraph show how to gets information from 3D database and makes updates into the same database. In this section, we present the update of the 3D database according to the information extracted from others databases like a management database. For example, the 3D database contains a scene representing cylinders in the 3D representation of the building of the company. The cylinders represent the employees of the company. The color of the cylinder varies according to the hierarchical position of the employee in the company (a red cylinder represents the boss and a blue cylinder represent the secretary). Our goal is to combine the 3D elements of this scene with information of a management database witch specified that the features (Name, department name, phone office…) of the employees and the localization of their office in the building.

To reach this goal, we add an association-table to link an employee tuple to a cylinder tuple. The evolution of the database by using new association-tables for each 3D database properties we want to update allows a dynamical update of the scene. For example if an employee becomes a secretary the scene is update by the following request:

```
Update Material set DIFFUSECOLOUR= blueColour
where material_id=(
      select Marerial_id from SHAPEMATERIAL
      where shape_id=(
            select shape_id from SHAPEEMPLOYEE
            where Employee_id =  updateEmployee_id
            ;))
```

This example shows how to change a shape's color form a management database update. More complexes modifications can be done by updates of 3D modeling of a shape. We are working to develop a complete system that directly generates the 3D scenes from a view of management databases. This view is defined by associating some specific information of the management database with a 3D representation will allow the database administrator to dynamically handles company information through a 3D graphical interface.

D. 3D Database Evaluation

In this section, we present our performance evaluation of the X3D database (insertions and extractions). The source files are in the VRML format. These tests are made without database or network optimization. The following tests concern 1) the time to convert a VRML file into a X3D document, to create the corresponding insertion script, and to execute this scrip by the database, 2) the time to extract data and build a X3D document from the database.

These results show that processing times depend on two criteria: The size of the file and the structure of the scene. We note that, Scene 4 contains fewer tag than Scene 3. Nevertheless, Scene 4 size is larger. The graphic elements are described by a set of points rather than by a set of primitives (cube, sphere, etc.). Some elements of the Scene 4 possess more than 15000 characters. We note that the extraction times are very significant in comparison with the insertion times. It is due to the Oracle optimizer that detects the best path in the database schema and then rewrites the query before executing it. We are going to proceed with three types of optimizations: file optimization by detection of primitives, network optimization by the addition of a cache zone on the server to reduce the client response time, and database optimization (switching off the Oracle optimizer and using Oracle's cache system for extraction of data).

Scene	1	2	3	4
File Size (in Kb)	315	1962	4039	8767
Creation of the Insertion Script	1s	4s	5s	40s
Insertion Script Runtime	6s	25s	30s	25s
Extraction of the X3D scene	15s	1m20s	2m20s	2m05s

V. CONCLUSION AND FUTURE WORK

In this article, we have presented a method of modeling and storing of 3D-scenes in a relational database. We use the X3D language as an underlying model. This language, based on the XML standard, is compatible with VRML. We have constructed a functional architecture of insertion and extraction of 3D-scenes into and from a database. This architecture permits us to manipulate 3D-scenes by SQL queries.

In the setting of an industrial project, we currently strive to set up an interoperability architecture that associates our 3D-database with standard databases. The final objective is to build an Internet environment to provide a three dimensional-visualization of management information.

BIBLIOGRAPHY

[1] Expandable Markup Language (XML): ttp://www.w3.org/XML /
[2] W3C Scalable Vector Graphics (SVG): http://www.w3.org/Graphics/SVG/Overview.htm8
[3] The World Wide Web Consortium (W3C) develops interoperable technologies: http://www.w3c.org
[4] International Web3D specifications VRML97 Standard: http://www.vrml.org/fs_workinggroups.htm
[5] Cosmo Player: http://www.cai.com/cosmo
[6] Blaxxun: http://www.bluxxun3d.com
[7] Information Architecture Markup Language (IAML): http://vizbang.com
[8] XGL File Format Specification: http://www.xglspec.org
[9] 3DML: http://www.global-dev.com/ress_dev/3dml
[10] Expandable 3D (X3D): http://www.web3d.org/x3d.html
[11] Expandable Style sheet Language (XSL): http://www.w3.org/Style/XSL
[12] PHP is has server-side, cross-country-platform, HTML embedded scripting language: http://www.php.net /
[13] P.P. Chen, The Entity-Relationship Model Toward has Unified View of Dated, ACM Transaction one Database Systems, Vol.1, N°1, March 1976,