ACTIVe3D[1]: Interrogation de scènes 3D en SQL

Christophe Cruz * – Christophe Nicolle * – Marc Neveu **

** Equipe Ingénierie Informatique*
*** Equipe Synthèse d'Image*
Laboratoire Le2i - CNRS FRE 2309
Faculté des Sciences Mirande - Université de Bourgogne
BP 47870 – 21478 Dijon Cedex
{christophe.nicolle, marc.neveu}@u-bourgogne.fr
c.cruz@active3d.net
www.active3d.net

RÉSUMÉ. *A la frontière de deux domaines de recherche, l'interopérabilité des bases de données et la synthèse d'images, cet article présente une méthode de modélisation et de stockage par le contenu d'objets 3D dans une base de données relationnelle. Cette méthode sera utilisée pour le couplage entre une « base 3D » et des bases de données classiques. Notre objectif final est le développement d'un environnement Internet permettant l'accès 3D à des données textuelles et la manipulation de scènes 3D à l'aide de requêtes SQL.*

ABSTRACT. *This paper presents a modeling and storage method of 3D objects in a relational database. This work is placed to the border of two domains of research: the databases interoperability and the computer graphics. An interoperabilty architecture based on this method was built allowing both the manipulation of 3D Scenes with the help of SQL queries and the coupling of these 3D objects with the classic management data bases.*

MOTS-CLÉS : *X3D, Base de données 3D, SGBD Relationnel.*

KEYWORDS: *VRML, X3D, 3D Database, Relational DBMS*

[1] Ces travaux sont co-financés par la société ACTIVe3D-Lab, le Conseil Régional de Bourgogne et l'ANVAR.

1. Introduction

Le développement exponentiel d'Internet tend vers deux domaines qui semblent opposés. D'une part, l'aspect visuel où le texte qui composait initialement les pages des premiers sites WEB a été remplacé par des images et des animations. On constate dans ce domaine la percée de logiciels tels que Flash de Macromédia. D'autre part, l'aspect informatif s'est considérablement développé. L'information donnée par les sites devient intelligente, adaptative, en fonction des comportements des internautes. Les nouvelles avancées en matière d'interconnexion de bases de données avec des pages HTML ont permis la création de nouveaux sites dynamiques. A ce jour, un site Web se doit d'être animé, donc attrayant et intelligent, donc actif et interactif. Néanmoins, il existe de nombreuses limites. Dans le domaine de l'aspect visuel, la représentation 3D est en pleine croissance sur Internet. Néanmoins elle est souvent limitée à de petites animations, car les ressources nécessaires pour utiliser la 3D sur le réseau sont trop importantes. Quant à l'aspect informatif, il est encore trop souvent limité à l'interfaçage d'une base de données avec du code HTML. La construction de systèmes complexes interconnectant plusieurs bases de données n'est encore développée qu'au stade de la recherche.

Pour répondre à ce problème, nous avons développé une nouvelle technologie appelée ACTIVe3D. Son principal objectif est de créer un lien entre le stockage de l'information et sa représentation visuelle. Pour cela nous avons développé la première base de données relationnelle qui stocke des scènes 3D par le contenu. Ensuite, nous avons développé des outils qui permettent l'interconnexion de la base de données 3D avec n'importe quelles autres bases de données dans un environnement coopératif. Ceci nous permet d'ajouter de la sémantique d'un domaine d'application aux objets 3D stockées dans la base de données 3D.

Dans cet article, nous présentons la technique de stockage de scène 3D dans une base de données relationnelle, noyau de la technologie ACTIVe3D. Le présent est organisé en 4 sections. La première section présente un état de l'art des méthodes de stockage de scènes 3D. La seconde section présente la méthodologie de stockage de scènes 3D dans une base de données relationnelle. La troisième section présente l'architecture de la base de données 3D. En particulier, les processus d'insertion, d'extraction et de modification de scènes 3D dans la base de données seront présentés.

2. Etat de l'art

Aujourd'hui, l'utilisation du stockage d'informations 3D dans une base de données se limite à très peu d'applications. Les plus répandues sont les systèmes d'information géographique [SIG 02]. Bien que la majeure partie de ces informations dans ces systèmes soient 2D, ils évoluent vers des versions 3D. Les modèles numériques de terrains (MNT) sont à eux seules les objets 3D principalement manipulés dans les SIG. Un MNT est un réseau maillé régulier, dont chaque maille est repérée par les coordonnées de sont centre. Un MNT peu recevoir

plusieurs couches d'informations dont la première est la couche Altimétrique. Les nouvelles couches sont pour la plupart du temps les contours de bassins versants ou des réseaux hydrographiques (couche vecteur). Ceci leur confère une structure pauvre en terme géométrique.

Le projet STEP [STE 02] est une norme internationale pour la représentation et l'échange entre systèmes électroniques des données de produit dont l'objectif est de fournir un mécanisme neutre capable de décrire des données de produit dans tout son cycle de vie, indépendamment de n'importe quel système particulier. La nature de cette description le rend appropriée non seulement à l'échange de fichier, mais également comme base pour mettre en application et partage des bases de données et l'archivage de produit. Cette Base de données possède une interface SDAI [SDA 02] définie suivant la partie 22 de STEP. La partie 42 quant à elle décrit un modèle géométrique et topologique complet. Bien qu'interfaçable avec une base de données très peu d'applications industrielles ont été réalisées car la visualisation simple d'une base de données dans une application est contraignante. L'INRIA dans le projet iMAGIS [IMA 97] a réalisé un prototype logiciel permettant la navigation interactive dans une scène urbaine, en réalisant une segmentation de la base de donnée par rapport à la localisation du point de vue. Certaines données proches de l'observateur sont traitées en 3D, pour permettre par exemple de véritables tests de collision, mais les données distantes sont simplifiées et regroupées dans des « imposteurs ». Un « imposteur » est donc une représentation simplifiée d'une partie de la scène 3D. Ils ont proposé une technique de construction des imposteurs à base d'images de synthèse, qui permet de concilier le double impératif de simplicité (faible nombre de primitives graphiques, recours à la texturation) et de fidélité (rendu correct des effets de masquage entre immeubles lorsque l'utilisateur avance dans une rue...)

L'approche active3D se tourne vers le web, raison pour laquelle nous avons un intérêt particulier pour les langages orientées réseaux. Nous sommes au printemps de la 3D que connaît Internet car nous voyons fleurir un grand nombre de langages propriétaire axé 3D. Le langage correspondant le plus à la philosophie web est VRML [VRM 97]. Celui-ci est libre, créé par les utilisateurs d'Internet pour leur usage, il reprend les différents principes d'une page HTML, comme les liens entre documents ou les liens d'objets. L'idée de coupler la synthèse d'image et la base de données n'est pas nouvelle. En 1997 Oracle dépose le brevet qui consiste à créer une base de données relationnelle pour le stockage de scènes 3D. Celui-ci n'a abouti à aucune application commerciale. Leur modèle diffère du notre en trois points ; le schéma de la base de données est généré à partir de la spécification VRML. Ils ne créent qu'une table pour tous les éléments sauf pour les éléments Script, Route et Proto. Et pour finir, ils créent une table par type d'attribut.

3. Méthodologie

XML [XML 02] est devenu rapidement un support universel de représentation de l'information sur Internet. Employé dans de nombreux domaines, il est largement utilisé pour véhiculer du multimédia et en particulier de l'image. La manipulation

d'images avec XML se fait à plusieurs niveaux. Au niveau le plus simple, il est possible d'insérer une image dans un document XML à l'aide de balises contenant l'adresse du fichier image. A un niveau plus complexe, il existe des langages de modélisation d'images 2D [SVG 02] et 3D [IAM 98][XGL 01][3DM 02][X3D 02] dérivés d'XML. A ce niveau l'image est décrite en mode texte, sous forme de balises. L'un des principaux problèmes dans le domaine de la modélisation 3D avec XML est la création et la réutilisation d'une structure de données qui décrit une scène 3D. Actuellement, il n'existe pas de moyen normalisé qui permette de manipuler des scènes 3D de manière générique. Pour résoudre ce problème nous avons développé une architecture de modélisation et de stockage de scènes 3D utilisant une base de données relationnelle. Cette base permet de stocker les scènes 3D par le contenu permettant la manipulation des scènes à l'aide d'une algèbre relationnelle.

3.1. *Langages*

Le principal langage de modélisation 3D est VRML sur Internet. Largement répandu, il est possible de citer plus de trente programmes ou plug'ins utilisant VRML, tels que CosmoPlayer [COS 02], Blaxxun3d [BLA 02], etc. De nombreux modeleurs comme 3D Studio Max de Kinetix renferme des fonctions d'exportation pour la création de fichiers VRML. Malheureusement, un fichier VRML est un script monolithique difficilement exploitable. Pour résoudre ce problème, nous avons développé un outil Java de conversion VRML vers le langage X3D [X3D 02]. Basé sur la norme XML, X3D fait l'objet d'un accord entre le W3C [W3C 02] et Web3D Consortium. Le choix de X3D s'impose pour plusieurs raisons. XML est un méta-langage de description qui structure l'information à l'aide de balises organisées dans un arbre. En comparaison avec VRML, la modélisation de l'information est identique mais la forme est plus souple en terme d'utilisation. L'approche 3D avec XML apporte la flexibilité et la capacité d'extension que VRML ne peut pas apporter. Ceci permet l'ajout d'informations aux scènes, la création de nouvelles balises sans gêner la génération des scènes. Malgré sa grande jeunesse de nombreux outils ont été développés et sont disponibles pour manipuler des structures XML.

Une fois les scènes obtenues en X3D, nous devons insérer les éléments qui la composent dans la base de données. Pour cela, nous utilisons des feuilles de styles XSL [XSL 02]. XSL est un langage qui permet la manipulation d'un arbre XML. Le point fort de XSL réside dans sa capacité à projeter une source de données uniques sur de multiples cibles d'affichages et de formater les données sur chaque cible. Il existe deux types de formatage : le formatage des données et le formatage des résultats.

– *Formatage des données* : Le formatage des données est constitué de trois étapes. La première étape consiste à générer une structure de données sous forme d'un arbre source, en associant les éléments (nœuds de l'arbre) à divers motifs stylistiques (appelés patterns). La deuxième étape réalise une analyse de l'arbre pour produire un arbre résultats ou final. Celui-ci est basé sur les actions spécifiées dans les règles dites gabarits, patrons ou modèles. Dans la dernière étape, l'analyseur syntaxique prend le relais en modifiant les

données XML à l'aide des instructions de formatage XSL. Il en résulte un arbre final sous forme de page HTML pour l'affichage sur le client Web. XSL est capable de réordonner les données de sorte que l'arbre final puisse être totalement différent de l'arbre source.

– *Formatage du résultat* : XSL est indépendant du langage d'affichage. Il est donc possible de développer un analyseur syntaxique XSL qui traduise l'arbre source en d'autres formats de sortie comme le RTF, le PDF ou encore créer des scripts comme PHP [PHP 02] ou des scripts Oracle.

```
<schema>
   <element name="EspduTransform">...
   <element name="ReceiverPdu">...
   <element name="SignalPdu">...
   <element name="TransmitterPdu">...
   <element name="X3D">
      <type content="elementOnly">
         <group>
            <element minOccurs="0" ref="Header"/>
            <element ref="Scene"/>
         </group>
   </element>
   <element name="Header">...
   <element name="Scene">...
</schema>
```

Script 1. *Extrait de la grammaire de X3D*

Un document X3D, comme tout document XML possède une grammaire qui définit la structure des balises qui composent le document. Cette grammaire appelée aussi schéma est décrit dans le format de balisages d'XML à l'aide du langage XML-Schéma. Un parseur teste la validité d'un document X3D en comparant sa structure avec cette grammaire. Un exemple de grammaire X3D est présenté dans le script 1.

Le schéma X3D est composé d'un ensemble de balises éléments. Ces balises éléments, par exemple **X3D** ou **Header**, seront des nœuds de l'instance du schéma XML. Les nœuds de l'arbre peuvent avoir des fils qui seront des éléments déclarés dans le schéma. Ainsi dans le script 1, l'élément **X3D** ne pourra avoir comme fils que l'élément **Header** et l'élément **Scene**. Les règles associées au schéma sont les suivantes :

1. Un élément peut contenir une arborescence
2. Une arborescence est toujours composée d'éléments
3. L'utilisation conjointe de 1. et 2. permet de modéliser tous les nœuds de l'arborescence
4. Les nœuds de l'arborescence sont ordonnés

3.2. *Création de la base de données*

Pour créer la base de données, nous avons procédé en deux étapes. Tout d'abord, au niveau conceptuel, nous avons défini un ensemble de règles de passage du schéma X3D dans un schéma entité-association. Ensuite, nous avons traduit ce schéma dans un schéma relationnel. A ce niveau physique, nous avons optimisé la base de données.

2.1.1. *Niveau conceptuel*

Plusieurs règles ont été développées au niveau conceptuel. Par faute de place, nous ne présentons dans cet article que les principales. Pour illustrer cette partie, nous utiliserons l'exemple présenté dans le script 2.

```
<element name= "TextureTransform">
   <type content= "empty">
      <attribute default= "0 0" name= "center"/>
      <attribute default= "0"   name= "rotation"/>
      <attribute default= "1 1" name= "scale"/>
      <attribute default= "0 0" name= "translation"/>
      <attribute name="DEF" type="ID"/>
      <attribute name="USE" type="IDREF"/>
    </type>
  </element>
```

Script 2. *Elément "TextureTransform"*

Règle 1. *Gestion des éléments* "*Un élément peut être composé d'une arborescence*"

Cette règle implique une clé externe d'Arborescence pour chaque table élément. De plus, chaque élément du schéma aura une table composé d'un identifiant (**Nomtable_Id**) et d'un ensemble d'attributs. L'application de cette règle sur notre exemple donne l'entité présentée en figure 1. Cet exemple ne contient pas de clé externe car l'élément **TextureTrasnform** ne peut être composé d'un autre élément. Dans le cas présent, nous créons une table par élément X3D, ce qui a pour effet de générer un nombre non négligeable de tables.

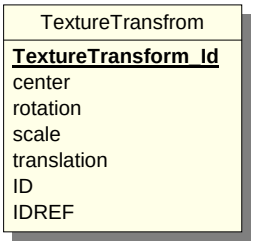

Figure 1. *Entité TextureTransform*

Règle 2. *Gestion des Nœuds* "Une arborescence est toujours composée d'éléments"

Pour traduire cette règle, nous avons créé une entité Arborescence et une entité **Nœud** modélisant la structure de l'arborescence. La modélisation de cette table nécessite l'utilisation de contraintes d'exclusion. Elles sont nécessaires pour modéliser le caractère optionnel de certaines relations entre les éléments. Par exemple l'élément **X3D** est formalisé sous la forme suivante : « *X3D = (Header | λ).Scene* ». Cette formalisation signifie que le nœud **X3D** est composé éventuellement d'un élément **Header** et d'un élément **Scene** (l'élément λ étant l'élément vide). Dans un schéma XML le « et » et le « ou » logique sont spécifiés par l'attribut *order* dans la balise **<group>**. Si l'attribut *order* est égal à 'choice' cela signifie que la balise **<group>** est de type « ou », dans le cas contraire si elle est égale à 'seq' alors la balise **<group>** est de type « et ». La balise **<group>** est par défaut de type 'seq' (correspondant au « et » logique). La balise **<group>** a deux autres attributs. L'attribut **minOccurs** correspond au nombre minimum d'occurrence, par défaut 1 et l'attribut **maxOccurs** correspond au nombre maximum d'occurrence.

```
<element ref="TextureCoordinate"/>
      <group minOccurs="0" order="seq">
            <element ref="Color"/>
            <element ref="ColorNode"/>
            <group order="choice">
                  <element ref="GeoCoordinate"/>
                  <element ref="Coordinate"/>
            </group>
            <group order="choice">
                  <element ref="USE"/>
                  <element ref="ProtoInstance"/>
            </group>
      </group>
</element>
```

Script 3. *Exemple d'élément complexe et multivalué*

Les contraintes d'exclusion totale vont donc permettre de modéliser ces règles de grammaire. A partir des règles énoncées précédemment dans la section Nœud et Elément nous avons réalisé le schéma entité-association de notre base de données. Ce schéma se compose de 103 entités dérivées de la règle 1, de 2 entités dérivées de la règle 2 (Entité Nœud et Entité Arborescence), de 104 relations et de 104 contraintes d'exclusions. L'ensemble des contraintes XML liées à la grammaire X3D est pris en compte grâce aux tables de contraintes d'exclusions. Ex : l'élément TextureCoordinate présenté dans le script 3 se formalise sous la forme.

« TextureCoordinate = (Color.ColorNode.(GeoCoordinate | Coordinate). (USE|ProtoInstance) | λ) »*

2.1.2. *Niveau Physique*

A ce niveau, le schéma entité-association fait place au schéma physique de la base de données. Ce passage a fait apparaître deux problèmes principaux dans notre phase de conception. Le premier problème réside dans les contraintes d'exclusion portées par les relations Nœud vers chacun des éléments. Leur traduction directe en relationnel surcharge de manière importante la taille de la base (Facteur 2). Le second problème réside dans la table arborescence qui ne contient qu'un seul attribut clé primaire. Pour résoudre ces problèmes, nous avons considéré que tout document X3D manipulé était valide. Cette validation sera effectuée par un parseur, en amont de la base de données. Ainsi, le schéma relationnel peut être optimisé en fonction de son utilisation et non plus en fonction de la grammaire du langage X3D. En résultat, nous avons supprimé la table arborescence, les contraintes d'exclusion et nous avons modifié la table Nœud. *Nœud(Noeud_Id, Elt_Id, Ordre, Type_Elt)*

L'apport de l'attribut *Type_Elt* va permettre d'identifier la table élément liée. La clé primaire a aussi été modifiée, elle est composée de l'identifiant de Nœud et de l'Ordre d'apparition du nœud dans l'arbre. Les autres tables restent identiques. Le schéma final contient 104 tables. Nous avons un nombre conséquent de tables par rapport à l'approche d'Oracle mais notre méthode apporte deux avantages. Premièrement, La génération du script de création de base de données X3D est crée à partir du schéma XML, ce qui permet de suivre automatiquement les évolutions du langage X3D. La seconde, dans le cas de l'approche Oracle, il faut autant d'interrogation de la base de données que d'attribut constituant un élément. Le fait de faire une table par élément supprime cette contrainte en faisant une simple sélection d'un tuple. En ce qui concerne la structure de l'arborescence, le nombre de requête pour l'extraction ou l'insertion est identique.

4. Architecture

Après avoir décrit la création du schéma de la base, nous allons maintenant présenter les processus d'insertion et d'extraction des scènes 3D.

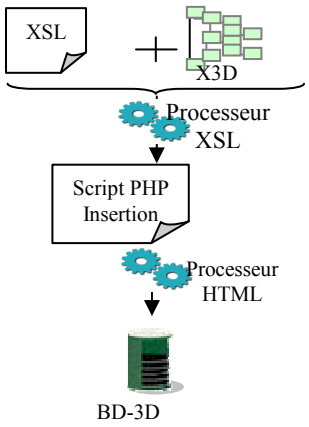

Figure 2. *Architecture d'insertion*

4.1. *Processus d'insertion*

Ce processus est basé sur l'utilisation de feuilles de style XSL. Les feuilles de style XSL ne fonctionnent pas avec la même logique que les autres langages de programmation. Pour cela nous avons fait deux modèles qui s'appellent récursivement. Ce procédé nous permet de parcourir n'importe quel type d'arborescence, quelle que soit sa taille. Le script 4 présente un extrait de la feuille de style XSL correspondante.

Pour compléter cette approche, nous avons géré dynamiquement la construction de scripts PHP qui seront créés en fonction du document X3D manipulé. L'architecture d'insertion mise en place est modélisé par le schéma présenté en figure 2. Chacune des flèches représente une action conjointement utilisée par les objets correspondant à l'origine de chacune des flèches. La feuille de style XSL qui va générer le script PHP d'insertion contient les éléments qui permettront d'établir la connexion avec la base de données. Elle va tester chaque nœud et déterminer son type. Une fois le type déterminé, elle crée le script PHP d'insertion correspondant. Ce script PHP réalise l'insertion de l'élément courant dans la bonne table en exécutant des requêtes SQL. Il crée une variable globale portant le nom du numéro de nœud courant dans l'arbre. Cette variable a pour valeur l'identifiant du nœud père. A la fin, cette feuille crée un script de fermeture de la base de données. Une fois les scènes stockées dans la base de données, il est très simple de les manipuler à l'aide du SQL. Les valeurs de tous les attributs peuvent être modifiés. Des insertions, modifications, suppressions d'objets 3D dans plusieurs scènes peuvent être effectuées simultanément. Il est possible de faire évoluer dynamiquement des

attributs de la scène pour avoir un rendu plus réaliste. Par exemple, positionner une source lumineuse dans une scène représentant le soleil et faire changer sa position au cours du temps.

```
...
<!-- Variable globale pour manipuler une valeur -->
<xsl:variable name="Noeud_courant" />

<!-- Modele de depart -->
//La feuille de style se place à la racine
<xsl:template name="root" match="/">
   //Initialisation de la variable globale
   <xsl:variablename="Noeud_courant">0</xsl:variable>
   // Appel de la boucle
   <xsl:call-template name="brother">
      <xsl:with-param name="PERE_ID"select="$Noeud_courant"/>
   </xsl:call-template>
</xsl:template>

<!-- Modele brother pour passer en revue tous les freres-->
<xsl:template name="brother" match="/*">
//Pour chaque elt Passage en parametre id du pere
   <xsl:param name="PERE_ID"/>
   <xsl:for-each select="*">
      <xsl:value-of select="$PERE_ID"/>
      <xsl:value-of select="name()"/>=
      <xsl:number level="any" count="*"  format="1.0 "/>
      //Calcul du numero du noeud courant
      <xsl:variable name="Noeud_courant">
         <xsl:number level="any" count="*"  format="1.0 "/>
      </xsl:variable>
      <xsl:if test="count(*)>0">
      //Si le nœud courant a des fils alors on continue
      //Alors appel du modele pour le traitement recurcif
       <xsl:call-template name="child">
        <xsl:with-paramname="PERE_ID" elect="$Noeud_courant"/>
       </xsl:call-template>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
...
```

Script 4. *Exemple de feuille de style XSL récursive*

4.2. *Processus d'extraction*

Le processus d'extraction des scènes 3D est basé sur le même procédé que le processus d'insertion. Un script PHP extrait les données de la base à l'aide d'une requête SQL qui va reconstituer les données de l'arborescence X3D. Ce script formate ensuite ces données sous forme de texte et de balises pour la restitution de la scène dans un format X3D. Une feuille de style XSL est appliquée pour traduire directement le document X3D en script VRML. Il est possible alors de visualiser à l'aide d'un navigateur la scène 3D correspondante. Ce processus d'extraction n'est pas limité uniquement à l'affichage des scènes 3D. La requête SQL d'extraction peut

aussi porter sur des objets qui composent les scènes. A l'intérieur de la base de données la scène n'existe plus comme entité individuelle comme elle peut l'être dans un modeleur. C'est le contenu de la scène qui est stocké. Un des avantages de cette organisation est la possibilité d'extraire un objet particulier de plusieurs scènes à la fois. Au lieu d'interroger les différentes scènes une par une, il suffit de requêter la table contenant les instances de l'objet pour toutes les scènes.

4.3. *Modifications des données*

Une scène est modélisée sous la forme d'un arbre. Il en est de même pour les objets. Ainsi un objet regroupe un ensemble de composantes. Par exemple, une table est constituée d'un plateau et de quatre pieds. Il est possible de définir des attributs d'apparences pour chacun des éléments ou bien d'une façon générale pour tous les éléments. La plupart du temps les objets sont définis par une balise **Shape** contenant la description physique et visuelle de l'objet comme ses dimensions et sa couleur. Il est possible de définir un nom pour chaque balise Shape donc lorsque l'on connaît le nom de l'objet il est possible d'avoir accès aux éléments de l'objet et de leurs caractéristiques. La modification des attributs s'opère à l'aide de requêtes SQL. Ex : *select SHAPE_ID, DEF from SHAPE where DEF is not NUL.* Cette requête permet de sélectionner l'ensemble des balises Shape portant un nom et de récupérer l'identifiant correspondant. En suite à l'aide de requêtes récursives on peut extraire les balises Material qui correspond aux caractéristiques de la couleur. Ex : *update MATERIAL set TRANSPARENCY = '0.3' where MATERIAL_ID ='10'.* Cette requête permet de modifier la valeur de la transparence de la balise **Material** dont l'identifiant est 10. L'avantage majeur de cette méthode réside dans le fait que cette modification s'opère pour l'ensemble des scènes contenant l'objet modifié. L'exécution automatique de la modification génère un gain non négligeable en terme de temps.

5. Conclusion

Dans cet article, nous avons présenté une méthode de modélisation et de stockage de scènes 3D dans une base de données relationnelle. Nous avons utilisé comme pivot le langage X3D. Ce langage, basé sur la norme XML est compatible avec VRML. Nous avons construit une architecture fonctionnelle d'insertion et d'extraction de scènes 3D dans une base de données. Ceci nous permet de manipuler les scènes 3D à l'aide du langage SQL. Dans le cadre d'un projet industriel, nous avons une architecture d'interopérabilité associant notre base de données 3D à des bases de données de gestion et des bases de données spécialisées (IFC). Notre prochain objectif est d'étudié les possibilités de manipulations des objets 3D au travers de la sémantique fournie par les bases de données IFC pour définir quel est le rôle de chaque objet 3D. Ceci pour extraire les objets en fonction de leur sémantique, ce qui à terme autorisera la génération de scènes en fonction d'un contexte [BAR 97][LIP 99].

Bibliographie

[3DM 02] 3DML, www.global-dev.com/ress_dev/3dml

[BAR 97] Alberto Barbora Raposo, Léo Pini Malgalhães, Ivan Luiz Marques Ricarte, Sate University of Campinas UNICAMP, School of Electrical and Computer Engineering FEEC, Dept. of Computer Engineering and Industrial Automation DCA, University of Waterloo - Computer Science Dept., Working with Remote VRML Scenes Throught Low-Bandwith Connections, 1997

[BLA 02] Bluxxun, www.bluxxun3d.com

[COS 02] Cosmo Player, www.cai.com/cosmo

[IAM 98] Information Architecture Markup Language, IAML, www.vizbang.com

[IMA 97] Rapport d'activité, Projet iMAGIS,1997, INRIA

[LIP 99] Daniel Lipkin, Belmont, Calif., Oracle Corporation, United States Patent, Method and Apparatus for Implementating Dynamic VRML, 1999

[PHP 02] PHP, HTML embedded scripting language, www.php.net

[SDA 02] Standard Data Access Interface, www.perdis.esprit.ec.org/apf/sdai/

[SIG 02] Enjeux de la gestion de base de données par un SIG, perso.respublica.fr/gerssat/sig/gereravecunsig.htm

[STE 02] Standard for the Exchange of Product model data, www.nist.gov/sc4/www/stepdocs.htm

[SVG 02] W3C Scalable Vector Graphics, www.w3.org/Graphics/SVG/Overview.htm8

[VAK 98] Alex Vakaloudis, Babis Theodoulidis, Timelab, Dept. of Computation UMIST, The storage and querying of 3D objects for the Dynamic Composotion of VRML Worlds, 1998

[VRM 97] Web3D Specifications VRML97 International Standard, www.vrml.org/fs_workinggroups.htm

[W3C 02] The World Wide Web Consortium (W3C) develops interoperable technologies, www.w3c.org

[X3D 02] Extensible 3D, X3D, www.web3d.org/x3d.html

[XGL 01] XGL File Format Specification, www.xglspec.org

[XML 02] Extensible Markup Language, www.w3.org/XML/

[XSL 02] Extensible Stylesheet Language, www.w3.org/Style/XSL/