3DFC: a new container model for 3D file formats compositing

Rozenn Bouville Berthelot* Orange Labs and IRISA, Rennes, France
Jérôme Royan† Orange Labs France
Thierry Duval‡ IRISA, Rennes, France
Bruno Arnaldi§ IRISA, Rennes, France

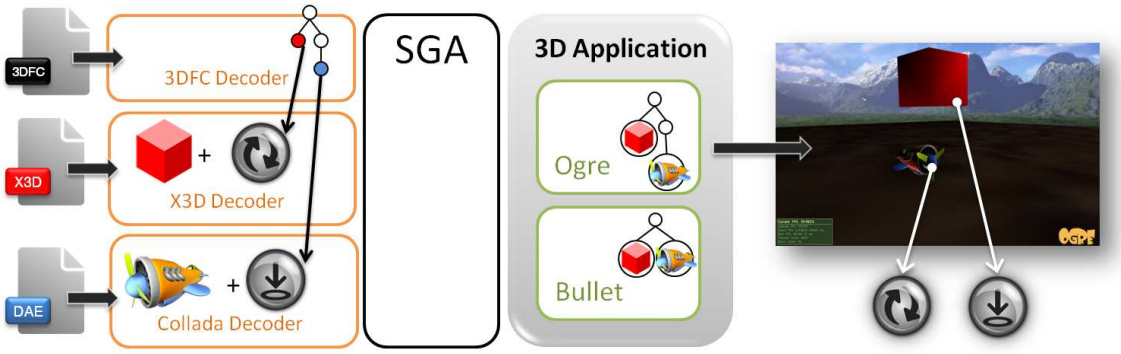

Figure 1: *The 3DFC container model combined with the Scene Graph Adapter (SGA) framework allows to load an X3D file and a Collada file simultaneously in every engine that composes a given 3D application, to mix their respective properties (a rotation interpolator or physics properties here) and to make them interact in the rendering window.*

Abstract

We present a 3D container model that enables the compositing of 3D file formats. It allows not only to compose 3D scenes made of several 3D files of different types but also to combine their functionalities and to make them interact together in the rendering window. This model, called 3DFC for 3D File Container, relies on the Scene Graph Adapter (SGA) architecture that makes it possible to load any scene-graph-based 3D file format in a 3D application whatever the involved engines (rendering engine, physics engine, etc.). This paper describes the 3DFC model, its integration with the SGA architecture and gives an implementation example. In this example we define a scene composed of an X3D file and a Collada file and we make their content interact in a 3D application that relies on Ogre 3D and Bullet.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality; D.2.12 [Software Engineering]: Interoperability—Data mapping; D.2.13 [Software Engineering]: Reusable Software—Reusable libraries

Keywords: interoperability, 3D format, software architecture, rendering, virtual worlds, web3D, 3D container

1 Introduction and motivation

The vision of the Web 3D is a promising trend of the future Web. It will radically change how people navigate the Internet. To make this dream comes true there remains some research challenges to resolve. Havemann and Fellner [Havemann and Fellner 2007] pointed seven of these issues and the one that is interesting us here is "the lack of a single, commonly accepted and comprehensive 3D file format". A common 3D file format would offer many advantages such as a single 3D viewer to access all available 3D applications (i.e. virtual worlds, serious games, simulation, medical, manufacturing or cultural heritage applications for example). Exchanges of 3D data between 3D content creation tools would be a trivial task without unexpected results (loss of information on meshes such as vertices, normals, materials, loss of rigid body properties or loss of animation descriptions). Furthermore we could be able to reuse 3D models in several applications thus saving money and time to implementers. Unfortunately, instead, in 2008 McHenry and Bajcsy counted more than 140 3D file formats in a rather complete overview [McHenry and Bajcsy 2008] and new ones are released each year. Actually almost every 3D software uses its own proprietary format and despite the successive efforts from consortiums to define comprehensive 3D standards [Cellary and Walczak 2012], the situation remains unchanged. Due to this multiplicity of formats, essential tasks as exchanging, importing and reusing 3D data are made arduous. The only available solution is to resort to conversion, leading to errors and loss of information [McHenry et al. 2011].

A recognized solution [Polys et al. 2008; Silva et al. 2008; Spagnuolo and Falcidieno 2009] to answer this issue consists in improving interoperability among 3D file formats. A comprehensive interoperability implies no alteration of the 3D files involved. Moreover it means:

- interoperability between 3D formats and 3D applications i.e. allowing each 3D format to be loaded in any available 3D application,
- interoperability between 3D formats i.e. being able to define interaction between 3D models described in different 3D file formats.

Three categories of interoperability solutions have been discussed in [Berthelot et al. 2011]: the model agreement, the metamodel agreement and the model reconciliation. The reconciliation of dif-

*e-mail: rozenn.bouville@orange.com
†e-mail:jerome.royan@orange.com
‡e-mail:thierry.duval@irisa.fr
§e-mail:bruno.arnaldi@irisa.fr

ferent formats best fits our requirements. We indeed think that we should not get rid of this variety of 3D formats since each one brings specific features that are useful in a particular domain. In fact some 3D file formats are standards in a particular field of application of computer graphics but a standard format is usually not shared by several domains. Moreover a few of them are proprietary formats of very popular 3D softwares, making it difficult to be replaced.

This paper is organized as following: next section presents interoperability solutions and focuses on solutions that achieve interoperability between file formats. Then we present in section 3 the Scene Graph Adapter (SGA) architecture, an API that allows us to load any 3D file format in any 3D application whatever the engines involved. This architecture enables the definition of a 3D container model detailed in section 4, which makes interaction possible between different 3D file formats. Next, in section 5, we show how to instantiate our system using the example of a compositing of X3D and Collada contents and how to make them interact together. Eventually we discuss the usability of our model and conclude in section 6.

2 Related work

As explained in previous section, interoperability among 3D files formats must meet two main requirements. Indeed, an interoperable 3D format must be compatible with:

- most 3D applications without constraints,
- other 3D formats without alteration of files.

In this section we will study solutions that allow the second aspect, namely interaction between 3D files. The way we choose to enable interactions between file formats consists in aggregating them in a container file. We could not find many containers for 3D files except 3DMLW [3D Technologies R&D 2009] which allows to include models encoded in a different format inside a 3DMLW file (i.e. .3ds, Collada, AC3D, OBJ and .blend). Nevertheless container files is a long-standing model in other domains of computer science. One of the most prolific area is the digital media field where more than 60 file formats are commonly used. They are of two kinds: 1) storage container file formats and 2) transport container file formats. Their goal is to provide to users a reduced set of formats in a field where a huge variety of formats exists from natives formats (from digital production tools for audio, video and photo or recording tools, etc.) to compression formats. This simplifies tasks on multimedia files such as acquire, record, search, interrogate, edit, convert, transfer and distribute multimedia data. Digital media container file formats indeed allow to include metadata in order to annotate their contents. For example, metadata in video container file formats such as AVI [John McGowan 2000] or DIVX [DIVX 2012] allow to synchronize reading of video and audio streams but also to include other data such as multiple audio streams or subtitles. Other application of containers include software architecture management such as in the Salomé Project [Bergeaud and Tajchman 2007].

For 3D contents, the definition of a container format would be slightly different. In fact we do not need to synchronize 3D models but rather to organize them within a 3D scene and to make them interact with each other. The requirements for a 3D container file format are:

- encapsulate most existing or coming 3D file formats,
- spatially organize models within a virtual scene,
- enable communication between models (i.e. link nodes from the different scene graphs),
- and enable to annotate contents through metadata.

We can find some of these requirements in many 3D graphics API such as Java3D [1], OpenSceneGraph [2] and also for 3D engines such as Ogre 3D [3] in its import functionality. They indeed propose loading functions for the most commonly used 3D formats and make it possible to integrate models in their scene graph representation. The main limitation of these loading functionalities is that they convert the original scene graph into a targeted scene graph. This conversion usually results in features loss from the original format and sometimes leads to conversion errors. We then have no real interaction between 3D formats since interactions occur on the converted scene graph not on the original scene graph. This is not a case of interoperability as we have defined it in the previous section.

Another type of format, interchange formats, are a common type of 3D file format that covers most of the requirements we have defined for a container file except the encapsulation of 3D file formats. In [Mendling 2004], we can find a list of general design criteria for interchange formats:

- *simplicity*: a compact metamodel easy to understand,
- *completeness*: refers to the ability to include any available 3D file format,
- *generality*: in order to keep the features of each file format,
- *unambiguous*: a well defined format,
- *extensibility*: this prevents the interchange formats to be useless because of unanticipated further developments.

As regards the 3D graphics domain, several works published in the 90's have focused on the specification of an interchange file format such as the OFF file format [Rost 1989], P3D [Welling et al. 1990] or Koegel's project [Koegel 1992]. In the later, the author already argued that

> "the lack of a common format is a serious impediment to the development of the market for multimedia applications."

These early initiatives did not anticipate that new formats would be released each year since then. Today several interchange formats are commonly used but each one is generally dedicated to specific purpose. For example, FBX[4] is used for data exchange between Autodesk's tools, Collada is very popular among the game industry and OGC[5] is used for geospatial data. In fact there is at least one interchange format for each application field of 3D graphics. Even if most interchange formats enable the possibility to include external files, none of them meet the requirements mentioned before for a 3D container format. In the remainder of this paper, we present a 3D file container model that could be adapted to most interchange format thus turning any of them easily into a container file format in order to load complex 3D scenes using our Scene Graph Adapter architecture. First we present the Scene Graph Adapter, an architecture that makes it possible to load any 3D file format into a 3D application. Then we introduce a 3D file container model which allows a full interoperability between 3D file formats.

3 A flexible architecture to mix 3D files

3.1 Overview

A 3D application is usually based on several components such as a rendering engine, a physics engine, sometimes a network engine or

[1] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html
[2] http://www.openscenegraph.org/
[3] http://www.ogre3d.org/
[4] http://usa.autodesk.com/fbx/
[5] http://www.opengeospatial.org/

a behavior engine for example. These components are provided by third-party libraries which can only load a subset of file formats. As a result these components set the file formats compatibility of the application. Unless 3D models are encoded in one of these formats, a transcoding step will be needed to be able to load them in an application. The aim of our solution is to carry out the loading of any 3D file format in a 3D application that relies on any available engines, this without alteration of the loaded files. Our framework is based on an API called the Scene Graph Adapter (SGA). Its main task aims at adapting a format scene graph into engines scene graphs. We call *format scene graph* the scene graph defined in a file and *engine scene graph* the scene graph model used in a specific engine. Our API relies on the fact that almost every 3D graphics formats or engines are based on a scene graph structure, organizing nodes in the 3D space as well as their appearance properties. Several approaches are based on the same review but they focus on provided a flexible rendering like [Repplinger et al. 2010] or [Döllner and Hinrichs 2002]. We use our API to instantiate wrappers for each component of the 3D application in order to make a given format decoder communicate with a given engine. Figure 2 illustrates the SGA architecture and relations between wrappers, decoders, engines and the SGA.

There are two types of wrappers: format wrappers and engine wrappers. For each type of wrappers we can find in the SGA a specific API, namely the Format Adapter API and the Engine Adapter API. Format wrappers depend on a format decoder that parses the input file, loads the scene graph, handles updates of scene graphs or manages events. A format wrapper task is to make the decoder communicate with the 3D application components by implementing methods from the Format Adapter API and calling methods of the SGA Kernel. Engine wrappers allow engines to communicate with format decoders. They use the engine own API to implement methods from the engine adapter API and call methods of the SGA Kernel to achieve this task. The engine adapter API is made up of several APIs that provide specific methods for each type of engine (a renderer adapter API for rendering engine, a physics adapter API for physics engine, etc.). One format wrapper (respectively one engine wrapper) has to be implemented for each 3D format (respectively each engine). A more detailed description of our architecture can be found in [Berthelot et al. 2011].

3.2 The SGA Kernel

The SGA is actually made up of three main APIs: the Format Adapter API, the Engine Adapter API and the SGA Kernel API. In section 3.1 we have presented the purpose of the Format Adapter API and the Engine Adapter API as well as how they are used by wrappers. The Kernel is the entry point for every component of the SGA architecture, namely the 3D application, the format wrappers and the engine wrappers. The kernel manages the synchronization of format scene graphs and engine scene graphs involved in the 3D application in a transparent manner for wrappers. Each time a wrapper asks for the creation or the update of a node, the SGA Kernel ensures that this action will be performed by all concerned components. Figure 3 illustrates how the SGA kernel interacts with the other components of the architecture. When a format wrapper calls a method to create a new node then the SGA Kernel calls the corresponding methods for each engine that is part of the 3D application. If an engine wrapper notifies that a node has been modified then the SGA Kernel finds the format wrapper instance of the matching node in a format scene graph. In order to achieve these operations, the SGA uses the Node Indexer. The Node Indexer is a table that keeps track of each node loaded in the application. For each node of a format scene graph, it gives the matching node in each engine scene graph if there is any. Likewise, for each node of an engine scene graph, it gives the matching node in a format scene graph.

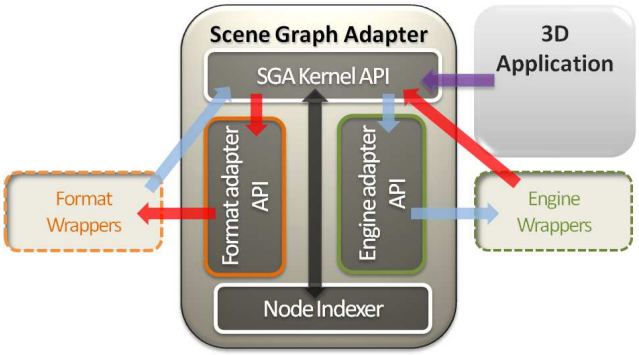

Figure 3: *The SGA Kernel is the entry point of our architecture. The coloured arrows shows the various communication flows between the different components.*

The SGA Kernel API is the core component of the SGA architecture, it can reach each loaded format wrapper, each engine attached to the application and each loaded node thanks to the Node Indexer. Some methods are only called by the 3D application such as methods to load a new file, to initialize engines used by the application or to update the elapsed time. The following methods can be called by format wrappers, engine wrappers or both:

- build a scene graph: build the scene (*attachNode, createTransform, createGround, etc.*), build geometries (*createBox, createMesh, etc.*), their appearance (*createMaterial, createTexture, etc.*) and their physics properties (*setNodePhysicsProperties*),
- update the scene graph: update the scene *(deleteNode, deleteNodeChildren, hideNode, updateNodePosition etc.*), update geometries appearance (*setMaterial, setTexture, etc.*) and their physics properties (*setNodePhysicsProperties*),
- handle interaction (*onPress, onRelease, onEnter* and *onExit*).

Through these methods the synchronization of scene graphs can be achieved whatever the involved engines, and scene graphs can cooperate with each other. For example, when setting the physics properties of a given node, a physics engine needs the absolute position of the node. This information is provided to the SGA Kernel by the rendering engine before updating the physics properties of the node in the engine scene graph of the physics engine.

3.3 A quick user guide

We have designed the SGA to make it an easy to use, versatile and extensible solution. To use it with a given 3D application, we first need to find a wrapper for each rendering component. If one or more does not exist, then it is necessary to develop one. Develop a new engine wrapper simply means to instantiate all the methods of the adapter API using the API provided by the chosen engine (Ogre API for an Ogre rendering for example) to initialize and update the engine scene graph. If the appropriate adapter API does not exist (for now there is a renderer adapter API and a physics adapter API), it is possible to create a new one that inherits from the Engine Adapter API.

In order to load a 3D file of a given format in your application, an existing format wrapper with its associated decoder can be used or a new one can be developed. In case a new format wrapper is needed, the first step consists in finding a suitable decoder on which the wrapper will rely. Then it is necessary to instantiate the Format Adapter API based on the chosen decoder. Depending on the functionalities included in the format, not all the Format Adapter API methods are required. We indeed try to make the Format Adapter

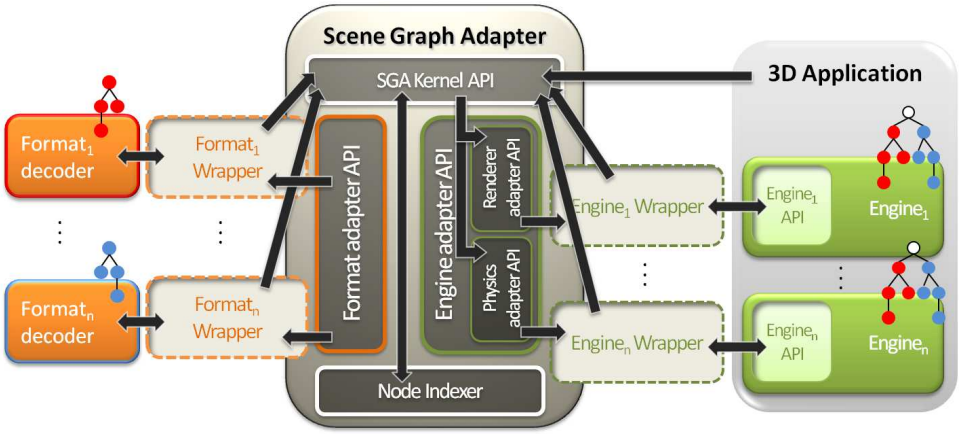

Figure 2: *The SGA architecture: it allows to mix several 3D contents in a single application without compatibility constraints.*

API comprehensive so that it covers many features that are not always specified by every formats. For example, when writing a Collada Wrapper, methods concerning interaction do not need to be instantiated.

Integrating a SGA in an application presents several benefits. First, as previously mentioned, our solution is highly extensible. It can be extended by creating new format wrappers, new engine wrappers or by adding new methods to the API thus adding new features. Furthermore wrappers are reusable, they are not dependent of a 3D application so that they can be shared by developers. Lastly, the SGA allows mixing simultaneously several 3D files whatever their format but also allows referencing a 3D model in a 3D file without being hampered by format compatibility issues. Indeed, using the possibility offered by a 3D format to reference an external file (the inline in X3D or URIs in Collada for example), the SGA makes it possible to call a file of a different format from any 3D file. Thus an external model can be easily included into the scene graph of a file. Thanks to that feature we can use a file format as a container format for 3D files. This feature of the SGA framework has been broadly discussed in [Bouville Berthelot et al. 2011].

In order to improve this interoperability concept, we propose in the next section a container model that aims at enabling interactions between 3D models encoded in different format.

4 A container model for 3D file formats

4.1 Design choices

The SGA architecture introduced above allows us to mix simultaneously heterogeneous 3D files in a single scene. Furthermore, using the file including feature of 3D file formats, we are able to use them as containers for other 3D files of different types, and spatially organize the resulting 3D scene. This feature answers the first requirement of interoperability as defined in section 1 but does not respond to the second requirement i.e. allow interaction between files. We indeed have no interaction between the 3D involved files.

The proposed approach consists in defining these interactions inside a container file and describing the 3D files associated and their connections. Our intention is not to propose a new 3D file format but rather to outline a model of container that could possibly be inserted into existing standard formats. We opt for an XML-based model because it is a widespread open-standard, supported by a large number of tools. Moreover its extensibility as well as its readability make it appropriate for 3D graphics. Besides it is the base language format of many common 3D file formats. Our model is called 3DFC for 3D File Container and has been designed as compact as possible, only defining the main nodes required for compositing. Therefore it contains a root node (`SGA-3DContainer`), a grouping node (`Group`), a positioning node (`Transform`) and a content node (`Content`).

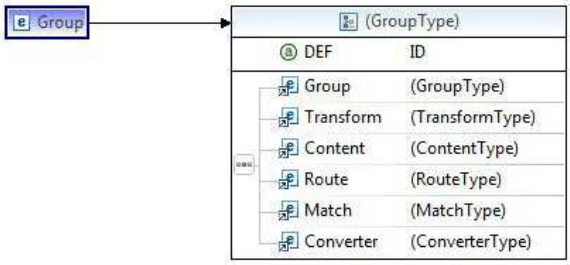

Figure 4: *The group node of 3DFC.*

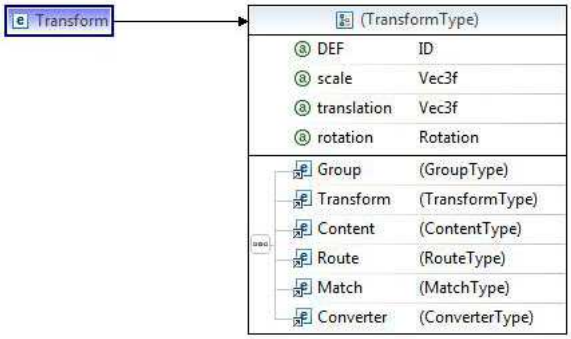

Figure 5: *The transform node of 3DFC.*

The (`Group`) (figure 4) and `Transform` (figure 5) nodes are typical nodes for organizing the scene. The `Content` (figure 6) node allows us to reference an external 3D file whatever its format. It has several attributes to describe the referenced file: its url, its type, its wrapper url, its decoder url and an attribute `DEF` to identify itself inside the container file. In figures 6, 8, 9 and 10, some attributes are types "ID" if the field has to be unique inside the 3DFC file, otherwise they are typed "string".

Table 1 shows an example of a 3DFC file that references two 3D

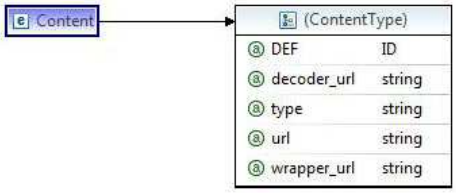

Figure 6: *The content node of 3DFC.*

files and the figure 7 illustrates the scene graph resulting from its adaptation by the SGA.

```
<?xml version="1.0" encoding="UTF-8"?>

<SGA_3DContainer>
  <Group DEF="MyGroupNode">
        <Content DEF="File_1" type="t1" decoder_url=""
            wrapper_url="" url="file1.t1" />
        <Transform DEF='TRANS' translation='-80 60 50'
            scale="2 2 2" >
           <Content DEF="File_2" type="t2" decoder_url=""
               wrapper_url="" url="file2.t2" />
        </Transform>
  </Group>
</SGA_3DContainer>
```

Table 1: *Example of a 3DFC file, a container model for compositing of 3D files.*

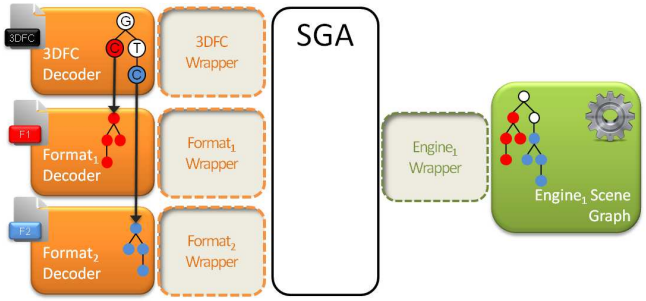

Figure 7: *This figure shows the engine scene graph that results from the 3DFC file in table 1. The nodes marked G, T and C stand for `group` node, `transform` node and `content` node respectively.*

4.2 Interaction nodes to mix 3D files functionalities

Now that we have introduced our 3D container model, we have a format that allows us to identify and annotate the 3D files that are combined in a 3D scene. We define interaction between these files through three new nodes: a `Route` node, a `Match` node and a `Converter` node.

The `route` node (see figure 8) creates a path between two equivalent properties of two nodes from two different format scene graphs. By equivalent properties we mean properties that have the same semantic. For example the field `translation` of an X3D `transform` node and the `translate` attribute of a `node` from a `visual scene` in Collada. The `route` node contains 7 attributes: `fromFile`, `toFile`, `fromNode`, `toNode`, `fromField`, `toField` and `converter`. `FromFile` and `toFile` values are identifiers of the two format scene graphs connected together. These identifiers are defined in the `DEF` attribute of the `content` node referencing each file. The attributes `fromNode`, `fromField`, `toNode` and `toField` are names of the nodes and fields in their respective format scene graph that are connected trough the `route` node. The last attribute, `converter`, is optional and is used when the properties linked with the `route` node, even if they have the same semantic, are not expressed in the same unit. Its value is the identifier of a `converter` node.

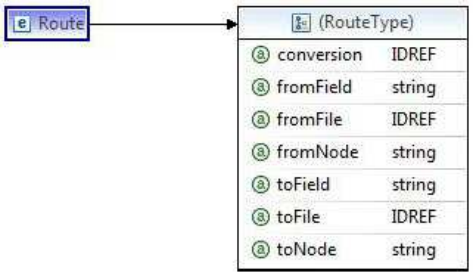

Figure 8: *The `route` schema structure.*

The `Converter` node (see figure 9) has two attributes. A `DEF` attribute to identify it within the 3DFC file and a `type` attribute that gives the type of conversion to apply. We can express many conversions like converting a vector in a Z-up referential to an Y-up referential or converting a rotation given in an Euler angle vector into a quaternion. These conversions are identified by a string that is interpreted by the 3DFC decoder which will perform the conversion.

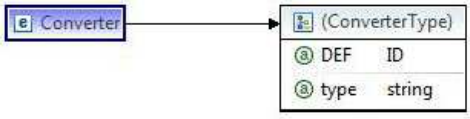

Figure 9: *The `converter` schema structure.*

The `match` node allows to add a property extracted from a given node to another node. We can use it to add a mass to a node encoded with a format that does not take into account physics properties. The match node has five attributes (see figure 10): `fromFile`, `toFile`, `fromNode`, `toNode` and `field`. The `fromFile`,`toFile`, `fromNode` and `toNode` attributes are identical to those of `Route` node. Eventually, the `field` attribute is the name of the property to match as it appears in the original format.

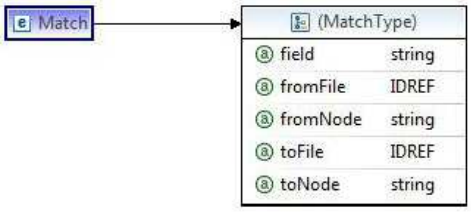

Figure 10: *The `match` schema structure.*

The sample file in table 2 declares a route and a match between nodes of two different format scene graphs.

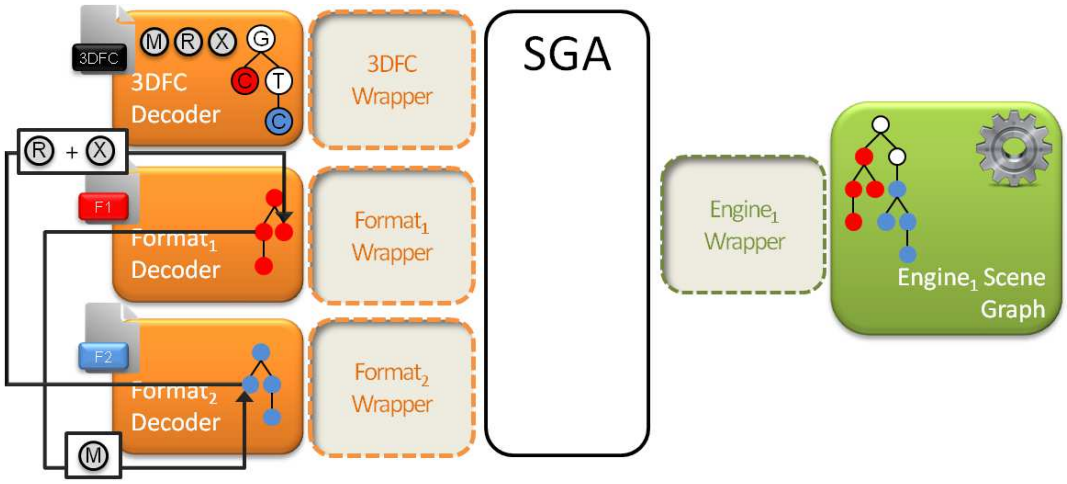

Figure 11: *This figure shows the engine scene graph that results from the 3DFC file in table 2. The nodes marked G, T and C stand respectively for* `group` *node,* `transform` *node and* `content` *node. The node M, R and Co stands respectively for* `Match` *node,* `Route` *node and* `Converter` *node.*

```
<?xml version="1.0" encoding="UTF-8"?>

<SGA_3DContainer>
  <Group DEF="MyGroupNode">
        <Content DEF="File_1" type="t1" decoder_url=""
             wrapper_url="" url="file1.t1" />
        <Transform DEF='TRANS' translation='-80 60 50'
             scale="2 2 2" >
           <Content DEF="File_2" type="t2" decoder_url=""
                wrapper_url="" url="file2.t2" />
        </Transform>
  </Group>
  <Match fromFile="File_1" fromNode="Sphere_1" toFile="
       File_2" toNode="Sphere_2" field="mass"/>
  <Route fromFile="File_2" fromNode="Sphere_2" fromField=
       "rotation"  toFile="File_1" toNode="Cube_1"
       toField="rotate" conversion="converter_1" />
  <Converter DEF="converter_1" type="VectRotToQuaternion"
        />
</SGA_3DContainer>
```

Table 2: *Example of a 3DFC file with interaction declarations between nodes of different 3D files.*

4.3 Possible uses and benefits of a container

Using a container makes 3D files compositing possible without format compatibility constraints and also allows their contents to communicate. Along with the use of the SGA framework, we have an interoperability model that answers the issue raised by the multiplicity of 3D formats.

The `route` node is used to copy a property of a given node to another node from a different format scene graph. The properties are shared among different files possibly encoded in different format. With the `Route` node, it is possible to organize and break down in our scene among several files in order to improve the reusability of the different involved contents. Moreover, the routing functionality allows us to bring animation in 3D file formats that do not provide it. Indeed we have two options to animate an ordinary node: 1) by creating a route between an external animated node and this node, 2) by creating a route between an external animation node and this node. Thanks to the `Converter` node, a route can be created between nodes without being hampered by an incompatibility of units between two semantically identical properties.

The `Match` node is used to match a property of a given node to another node declared in a format that does not offer this property. Through the `Match` node we are able to add properties to a model of any 3D format without altering the original file. The new property is defined by the file format where the match declaration comes from and it is interpreted by the SGA as if the modified node was a node of that format.

At this state of development, the creation of a 3DFC file is a manual operation (namely the organization of the composing contents, the definition of routes and matches as well as the use of converter node if required). We do not provide error detection; if a route has been declared between two incompatible properties then it will have no effect on the resulting scene. Nonetheless, it is possible to perform this task automatically by annotating the involved contents with an interaction model such as the MIM model described in [Chmielewski 2012].

5 Mixing X3D and Collada: our instantiation of the SGA architecture

5.1 Wrappers instantiation

We have implemented an instantiation of our architecture with three format wrappers and two engine wrappers. The three format wrappers are: an X3D wrapper, a Collada Wrapper and a 3DFC wrapper. The X3D wrapper uses CyberX3D [6] for the decoder part and instantiates most of the features provided by X3D specifications (cf table 3). The Collada wrapper relies on FCollada[7] and uses ColladaDOM[8] for URI management. Table 3 gives details on features implemented in the current version of the Collada wrapper. In order to have an in-depth description of wrappers instantiation, you can refer to [Berthelot et al. 2011].

The last format wrapper, the 3DFC wrapper, relies on a decoder that uses TinyXML[9] to parse 3DFC files. The 3DFC wrapper works in the same way as other format wrappers. During the parsing step,

[6] `http://www.cybergarage.org/twiki/bin/view/Main/CyberX3DForCC`
[7] `http://collada.org/mediawiki/index.php/FCollada`
[8] `http://sourceforge.net/projects/collada-dom/`
[9] `http://www.grinninglizard.com/tinyxml/`

each time a `Content` node is reached, the 3DFC wrapper calls the loading method of the SGA Kernel to load the referenced file. Then the SGA Kernel, according to the format of the loaded file, calls the appropriate format wrapper to adapt the format scene graph encoded by the file. On the other hand, the parsing of a `Route` or a `Match` node implies to refer to each involved format scene graph. This is the reason why the SGA Kernel provides developers with a programming interface to retrieve and set the fields values of a node in a given format scene graph. To retrieve or update a field value, in order to reach the concerned node, the SGA Kernel needs only two informations: the node identifier (i.e. its file url and its identifier within its format scene graph, usually its name) and the field name.

The following example will clarify the handling of external routes. Consider a route R between the files A and B. It creates a path between the field F_a of the node N_a and the field F_b of the nodeN_b (see figure 12). First the 3DFC wrapper retrieves the value of F_a in the format scene graph of N_a (**1**). It calls the retrieve method of the SGA kernel, then the kernel reaches the matching format wrapper to pass the value (**2**) helped by the Node Indexer. If a conversion is needed, when the value is known by the 3DFC wrapper (**3**,**4**) it is computed by the 3DFC wrapper depending on the type of conversion defined inside the `Converter` node referenced within the `Route` node (**5**). Finally, the value of F_b is updated inside the format scene graph of N_b (**6**,**7**) then the update is pushed to every engine attached to the current application (**8**).

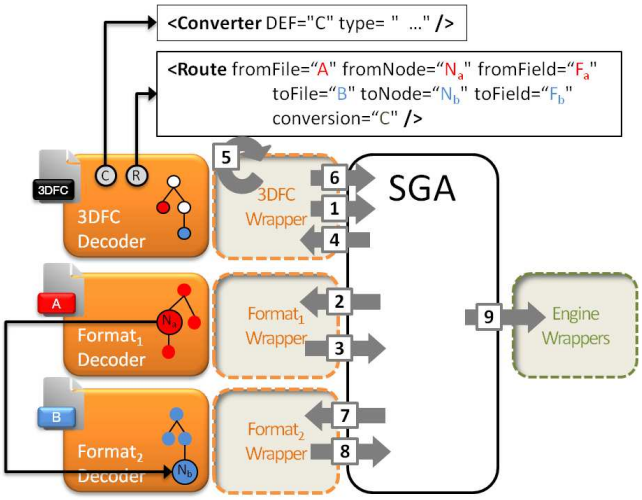

Figure 12: *This figure shows how the SGA process a* `Route` *inside a 3DFC file.*

Likewise, on the rendering side of our application, we use Ogre3D[10] as rendering engine and Bullet[11] as physics engine. We have created the two appropriates engine wrappers for these engines. Thanks to these four wrappers we are able to load any X3D file and any Collada file, to render them simultaneously in an Ogre window and to visualize the effect of physics simulation synchronized by Bullet on objects having physics properties.

5.2 Sample file and demonstration

In order to demonstrate the efficiency of our container model, we have written a 3DFC file that creates a composite scene with an X3D file and a Collada file. The X3D file contains a red cube, a rotation interpolator, a touch sensor and a time sensor that makes the cube rotate along the Z axis when the cube is clicked. The Collada file contains a plane, more precisely the Seymour's plane model, its physics properties and the physics properties of the world. They are placed side by side by a `Transform` node declared in the 3DCF file that encapsulates a `Content` node. This is the basic scene of the two sample files we present next (see table 4).

[10] `http://www.ogre3d.org/`
[11] `http://bulletphysics.org/wordpress/`

		X3D wrapper	Collada wrapper
Description	Navigation	X	-
	Viewpoints	O	X
	Models	O	O
	Materials	O	O
	Metadata	X	X
	Audio	X	-
	Textures	O	O
	Lighting	X	X
	Environment	O	-
	Performance	O	-
Interaction	Animation	O	-
	Sensors	O	-
	Scripting	X	-
	Route	O	-
	Physic	-	O

Table 3: *This table shows for each feature provided by X3D and Collada if it has been instantiated in the matching format wrapper. Features marked* ***O*** *are available, features marked* ***X*** *are not available and features marked* **-** *do not exist in the file format.*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <SGA_3DContainer>
4   <Group DEF="Mygroupnode">
5
6          <Content DEF="X3DFile" type="x3d" decoder_url=""
               wrapper_url="" url="../x3d/cube.x3d" />
7          <Transform DEF='TRANS' translation='-80 60 50'
               scale="2 2 2" >
8             <Content DEF="DAEFile" type="dae" decoder_url=
                  "" wrapper_url="" url="../collada/
                  Seymour_Plane/seymourplane.dae" />
9          </Transform>
10
11  </Group>
12 </SGA_3DContainer>
```

Table 4: *This sample 3DFC file creates a scene composed of an X3D file and a Collada file.*

In a first example, we add to our basic 3DFC file a `Route` node and a `Converter` node. The `Route` node creates a path between the rotation interpolator of the X3D file and a rotate node that is child of the propeller node. The table 5 shows an extract of the content of our 3DFC file.

The `Converter` node is referenced inside the `Route` node and it allows the conversion of a rotation expressed in axis-angle into a quaternion. The result of this route is that the propeller of the plane rotates in the Ogre rendering window when the cube is clicked. We indeed use the interpolator functionnality of X3D to make the Collada propeller rotates.

In a second example, we add a `Match` node to our basic file. The `Match` node allocates the mass of the plane to the cube as shown in

```
1        <Route fromFile="X3DFile" fromNode="AnimationOI"
             fromField="value_changed" toFile="DAEFile"
             toNode="prop" toField="rotate" conversion="
             convertor_1" />
2        <Converter DEF="convertor_1" type="
             VectRotToQuaternion" />
```

Table 5: *This code creates a path between an X3D node and a Collada node.*

table 6. Now that the X3D cube has a mass, it is a dynamic object for the physics engine. It falls in the Ogre window, its trajectory computed by the Bullet engine. We have used the possibility given in Collada to describe physics properties of a node in order to match it on an X3D node.

```
1        <Match fromFile="DAEFile" fromNode="plane" toFile
             ="X3DFile" toNode="cube_RED" field="mass"/>
2        <Match fromFile="DAEFile" fromNode="plane" toFile
             ="X3DFile" toNode="cube_RED" field="dynamic"
             />
```

Table 6: *These lines allocate two physics properties of a Collada node to an X3D node.*

We can of course combine the two examples in a unique scene as shown in figure 1. These two samples illustrate the possibilities offered by our container model and how simple it is to define interactions trough it.

5.3 Discussion and future work

Through this example, we demonstrate that it is possible to make 3D models of different types interact together in a single scene with the help of a container file. To do so, we propose a model that uses concepts that are simple and already used in the 3D graphics domain. Therefore the 3DFC model can easily be integrated into any existing 3D file format, it indeed requires to define only three new nodes to be able to create complex scenes composed of several types of 3D files. Of course, to make it work, the application should rely on the SGA architecture but, as previously noticed, our solution has been designed to provide interoperability among 3D formats and 3D engines with a model reconciliation approach. The proposed framework enables to develop a wrapper for each needed format or engine, it is a modular solution that works on reusable components.

The instantiation we have developed has to be improved in order to gain in efficiency. It indeed works well with simple files but we have not yet evaluated the impact of our architecture on complex scenes. In fact, we have identified three points in our architecture that could impact efficiency. First, the search in the Node Indexer is a task that involve calculation but it is not much demanding. Its complexity is in $O(\log n)$, n being the number of nodes in format scene graphs that have a representation in at least one engine scene graph. The second point is that for each update of a scene graph, an additional call is needed for the SGA kernel, compared to a standard decoder or engine. Nonetheless this should not have a huge impact on complex scene. The last point is that our architecture maintains several representations of the involved scene graphs at runtime. For each loaded file, we indeed have a representation in the format decoder and one in each engine used by the application. This fact should not impact performance since today it is the case in many 3D engines implementation without making them ponderous. We also consider several ways to gain in efficiency such as more efficient decoders, use of optimized geometries for physics engine, or compressed data.

As for future work, in order to elaborate our architecture, we want to assess the usability of the SGA for online virtual platforms. Thus we want to add a network engine to our current architecture. In addition we investigate a way to add specific attributes to nodes in our container model such as rights and user interaction properties to enable online collaborative works.

6 Conclusion

We present a solution that solves the interoperability issue among 3D file formats. Our Scene Graph Adapter (SGA) architecture allows several 3D contents of different formats to be rendered simultaneously in a 3D application that relies on any available rendering components (rendering engine, physics engine, etc.). Moreover the SGA enables communication between 3D assets encoded with different 3D formats. Indeed through a container file, we are able to describe a 3D scene composed of several 3D contents no matter of their original format and also to define interactions between them. A container format allows us to capitalize on all functionalities of a 3D format, mix them with other 3D format features and obtain a rich 3D scene without any preliminary alteration of the involved 3D files. Facing the proliferation of 3D file formats and the difficulties to establish a unique standard in the 3D graphic domain, we demonstrate that a solution trying to reconcile existing and coming models is a wise alternative. In order to use a new 3D format through the SGA architecture, the only thing to do consists in finding a decoder and implementing a format wrapper according to an API given by the SGA. This allows anyone to promote its own file format, whether a small or a big one, and makes it interact with other 3D file formats and 3D application.

References

3D TECHNOLOGIES R&D. 2009. 3DMLW. http://www.3dmlw.com.

BERGEAUD, V., AND TAJCHMAN, M. 2007. *Application of the SALOME software architecture to nuclear reactor research*. SpringSim '07. Society for Computer Simulation International, 383–387.

BERTHELOT, R. B., ROYAN, J., DUVAL, T., AND ARNALDI, B. 2011. Scene graph adapter: an efficient architecture to improve interoperability between 3d formats and 3d applications engines. In *Proceedings of the 16th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '11, 21–29.

BOUVILLE BERTHELOT, R., DUVAL, T., ROYAN, J., AND ARNALDI, B. 2011. Improving reusability of assets for virtual worlds while preserving 3d formats features. *Journal of Virtual Worlds Research 4*, 3.

CELLARY, W., AND WALCZAK, K. 2012. Interactive 3d content standards. In *Interactive 3D Multimedia Content*, W. Cellary and K. Walczak, Eds. Springer London, 13–35.

CHMIELEWSKI, J. 2012. Describing interactivity of 3d content. In *Interactive 3D Multimedia Content*, W. Cellary and K. Walczak, Eds. Springer London, 195–221.

DIVX. 2012. DivX Official Site . http://www.divx.com/.

DÖLLNER, J., AND HINRICHS, K. 2002. A generic rendering system. *IEEE Transactions on Visualization and Computer Graphics*, 99–118.

HAVEMANN, S., AND FELLNER, D. W. 2007. Seven research challenges of generalized 3d documents. *Computer Graphics and Applications, IEEE 27*, 3, 70–76.

JOHN MCGOWAN. 2000. John McGowan's AVI Overview: Audio and Video Codecs . http://www.jmcgowan.com/avicodecs.html.

KOEGEL, J. 1992. On the design of multimedia interchange formats. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, 12–13.

MCHENRY, K., AND BAJCSY, P. 2008. An overview of 3d data content, file formats and viewers. *Technical Report ISDA08002*.

MCHENRY, K., ONDREJCEK, M., MARINI, L., KOOPER, R., AND BAJCSY, P. 2011. Towards a universal viewer for digital content. *Procedia Computer Science 4*, 732–739.

MENDLING, J. 2004. A survey on design criteria for interchange formats. *Technical Report JM-2004-06-02. Vienna University of Economics and Business Administration-Department of Information Systems*.

POLYS, N. F., BRUTZMAN, D., STEED, A., AND BEHR, J. 2008. Future standards for immersive vr: Report on the ieee virtual reality 2007 workshop. *IEEE Comput. Graph. Appl. 28*, 2, 94–99.

REPPLINGER, M., LÖFFLER, A., SCHUG, B., AND SLUSALLEK, P. 2010. Sora: a service-oriented rendering architecture. In *Proceedings of Software Engineering and Architectures for Realtime Interactive Systems*, IEEE.

ROST, R. 1989. Off-a 3d object file format. *Digital Equipment Corporation Technical Report*.

SILVA, J., MAHFUJUR RAHMAN, A., AND EL SADDIK, A. 2008. Web 3.0: a vision for bridging the gap between real and virtual. In *Proceeding of the 1st ACM international workshop on Communicability design and evaluation in cultural and ecological multimedia system*, ACM, 9–14.

SPAGNUOLO, M., AND FALCIDIENO, B. 2009. 3d media and the semantic web. *Intelligent Systems, IEEE 24*, 2, 90–96.

WELLING, J., NUUJA, C., AND ANDREWS, P. 1990. *P3D: a Lisp-based format for representing general 3D models*. Supercomputing '90. IEEE Computer Society Press, 766–774.