Improving Reusability of Assets for Virtual Worlds while Preserving 3D Formats Features

Rozenn Bouville Berthelot, Thierry Duval, Jérôme Royan, Bruno Arnaldi

Improving Reusability of Assets for Virtual Worlds.

Improving Reusability of Assets for Virtual Worlds while Preserving 3D Formats Features

Rozenn Bouville Berthelot
Orange / IRISA Rennes, France

Thierry Duval
IRISA Rennes France

Jérôme Royan
Orange Rennes France

Bruno Arnaldi
IRISA Rennes France

Abstract

We propose a generic architecture that allows mixing several 3D formats in a single viewer whatever the rendering engine used by the virtual world. Our goal is to solve the issue raised by the multiplicity of 3D formats and rendering engines through an interoperability solution inspired by the web model. Our architecture relies on the Scene Graph Adapter, a component which aims at interfacing communication between virtual world inputs (e.g. 3D files) and outputs (e.g. the interactive visualization window). For this purpose, the Scene Graph Adapter is made up of two APIs that leverages similarities between 3D formats and 3D rendering engines, the Format Adapter API and the Renderer Adapter API.

Keywords: 3D format, interoperability, rendering engine, interface, wrapper

Scene Graph Adapter: a flexible architecture to enable interoperability between 3D formats and virtual worlds

3D formats interoperability: a first step toward virtual world interoperability

The lack of interoperability between 3D formats strongly impedes the deployment of virtual worlds. Creating a new virtual world is a complex task; it involves several modules, each one carrying out with a specific task like a rendering module, an animation module, a physics module, etc. Whereas most of these modules do not need to be built from scratch for each new virtual world thanks to third-party tools and libraries, the creation of 3D models still does. It is indeed very hard to reuse 3D models from a 3D application to another. In most cases there is no other solution than to resort to transcoding. But this has major drawbacks like functionality loss and invalid models (missing faces, geometry not correctly translated or physics data loss for example). Besides the choice of an input format is often determined by the choice of the rendering engine. With more than 140 3D files formats (McHenry & Bajcsy, 2008) and new ones created each year like XML3D (Sons et al., 2010), reaching a consensus on one standard format seems impossible. It is all the more difficult that a consensus format would have to provide every specific functionality of every format to have a chance to be broadly recognized and used. As noticed in (Thompson, 2011) and in (Jovanova & Preda, 2010), it is a critical issue for the future of virtual worlds. The only way to improve the current situation is to allow interoperability between 3D formats. Actually providing interoperability between 3D formats has several benefits. First, it allows easily importing and sharing assets from a virtual world to another. Then it enables mixing formats functionalities and leveraging the most of each 3D format (compression facilities, physics properties description, animation and interaction data, etc.). Finally it makes it possible to create a single virtual world client for all applications. In much the same way as web browsers allow access to every available web page, a single client could access every existing and future virtual

worlds. This could ease the creation of new virtual worlds and significantly increase user adoption of virtual environment. Several works present some solutions.

For example, Hu and Jiang in (Hu & Jiang, 2008) present a client-server architecture called “Plug” to find and access networked virtual worlds. The client uses streaming techniques to retrieve a virtual world that is running on a dedicated server.

The MPEG group proposes MPEG-V (Gelissen, 2008). One goal of this work is the definition of an intermediate format to enable interoperability between virtual worlds. An introduction to MPEG-V can be found in (Gelissen, 2009).

VWRAP (Bell, Dinova & Levine, 2010) is a comparable initiative but much more restrictive since it applies only to virtual worlds similar to Second Life (i.e. spatially partitioned into regions, hosted by simulation servers, with agents representing users and user-controlled via a client application).

These are efficient solutions but none of them allows reusing content without modification. Indeed all of them require an alteration of the original data like transcoding or annotation of content. Besides they do not guarantee to keep all features of the original data. Our goal is to propose a solution that avoids modification of input data while ensuring no functionality loss. By preserving functionalities we want to easily mix 3D formats and their features in a single viewer.

A flexible architecture that allows 3D format mixing

The Scene Graph Adapter architecture

Our architecture relies on a component called the Scene Graph Adapter whose role consists in managing the communication between virtual world inputs (e.g. 3D files) and outputs (e.g. the interactive visualization window). This API leverages similarities between 3D formats and 3D rendering engines as presented in (Döllner & Hinrichs, 2002) which generally use a scene-graph-based structure. We call the scene graph representation of a 3D format the *format scene graph* and the scene graph representation of a rendering engine the *renderer scene graph*. The Scene Graph Adapter is made up of four components as illustrated in figure 1:

- the Format Adapter API,
- the Renderer Adapter API,
- the Node Indexer,
- the kernel.

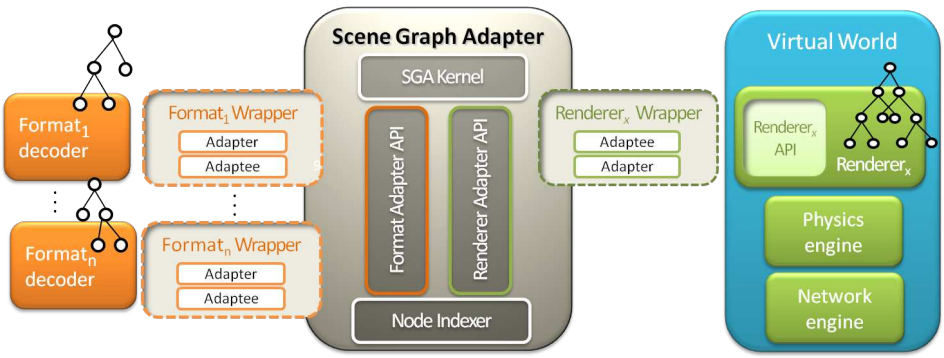

Figure 1 The Scene Graph Adapter allows loading simultaneously in a single rendering window several 3D files whatever their formats.

The Format Adapter API provides methods for accessing and modifying a *format scene graph* whereas the Renderer Adapter API provides methods for accessing and modifying a

renderer scene graph. At application run-time, the Scene Graph Adapter also maintains a Node Indexer that keeps track of the relation between a format scene graph node and its matching node in the renderer scene graph. Finally, the kernel provides methods used by the virtual world to load a file of any format and enable data exchange between a format decoder and a rendering engine. In order to use the Scene Graph Adapter, we must create wrappers. There are two types of wrappers: format wrappers and engine wrappers. Format wrappers use the Scene Graph Adapter to exchange data between a format decoder (e.g. a parser or a more advanced existing tool that interprets input content files) and the renderer wrapper. Its task is similar to the role of a plug-in for a web browser. As for web browsers plug-in, there is one format wrapper for each format. On the other hand, renderer wrappers use the Scene Graph Adapter to exchange data between the renderer API and format wrappers.

Through wrappers, a virtual world based on the Scene Graph Adapter can load any type of input files providing that there exists a format wrapper for that type. It has also no rendering constraint, it can use any scene-graph-based rendering engine providing that there exists an appropriate renderer wrapper. A more detailed description of the Scene Graph Adapter architecture can be found in (Bouville Berthelot et al., 2011) with technical information.

3D formats mixing mechanism

Thanks to our architecture we are able to easily mix 3D formats. We propose two solutions to achieve this. First we can load files separately. In that case, each format scene graph will be adapted in the renderer scene graph as a child of its root node. Second, we can reference 3D files of different types in a single container file using its own reference functionality. For example, we can reference Collada files in an X3D files through X3D's inline node. Figure 2 explains how this case is processed by the Scene Graph Adapter.

First when a node that references an external file is reached during the file parsing step (1), the format wrapper of the container file calls the kernel (2).Then the kernel loads the appropriate format wrapper according to the file extension (3,4).The format wrapper calls the format decoder (5) which proceeds to the loading of the file (6, 7) and to its adaptation to the renderer scene graph (8, 9, 10) adding these new nodes to the previous node created. Figure 3 gives the detailed sequence for the loading of a referenced file.

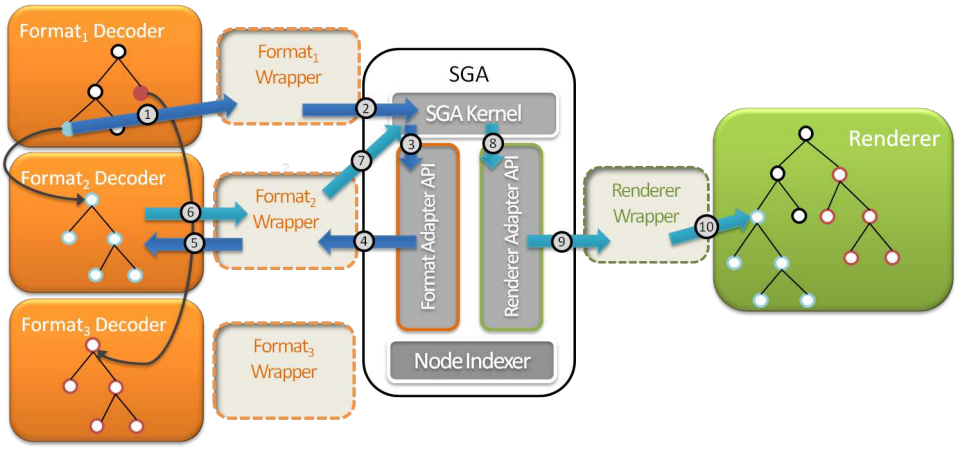

Figure 2 The Scene Graph Adapter allows the loading of any type of 3D files through references in a container file.

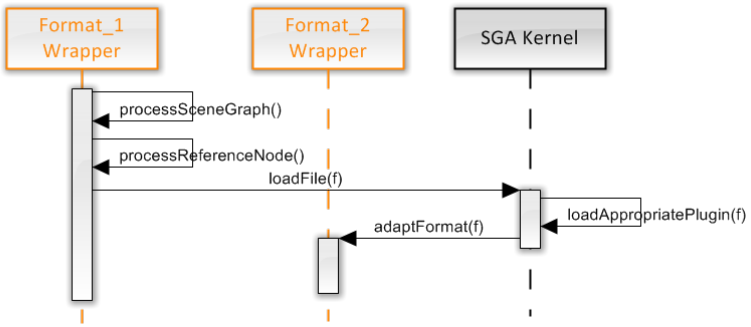

Figure 3 Sequence diagram for the loading of a 3D file of any type through another 3D file.

This has several benefits. First we can load several 3D files of any type through a single 3D file. Second, this allows including files in a structured manner, not simply as children of the sole root node of the renderer scene graph. Reference nodes can indeed be encapsulated in transform

nodes. The next benefit is that it enables interactions declared in the container file to be applied to 3D models that are not of the same type allowing animation of these models. This is similar to techniques used in 3D modelers but additionally the loading of behaviors, interaction and animation defined in the original 3D model is allowed.

Implementation example

We have realized an implementation of our architecture with a Collada wrapper and an X3D wrapper on the format side and an Ogre wrapper on the renderer side. This implementation enables the loading of X3D and Collada files and renders their contents in an Ogre rendering window. As we do not transcode input files, we encountered none of transcoding drawbacks. The Scene Graph Adapter adapts each format scene graph to the rendering engine scene graph. We are able to keep each specific functionality of X3D and Collada. For example, we have managed user interactions (sensor nodes) and event mechanisms (interpolators and route) provided by X3D in our X3D wrapper. A user input from the Ogre rendering window can thus be transmitted to the X3D wrapper and resulting changes in the X3D scene graph are sent back to Ogre in order to update the rendered scene.

We also process the inline node of X3D and enable inline nodes to include a file that is not necessarily an X3D file. This way we can include a Collada file into an X3D file. Through the Scene Graph Adapter, we can use X3D as a container for 3D formats and use sensors, interpolators and routes to animate Collada models.

As an example, suppose we have an X3D file like the following:

```
<X3D>
        …
        <Transform DEF='TRANS'  translation='-15 -5 1.0' rotation='0.0 0.707 0.707 0.9'>
        < Inline url='..\collada\Duck\duck.dae' />
                < TouchSensor DEF='Clicker'/>
        </Transform>
    <TimeSensor DEF='Timer' cycleInterval='4.0' />
    <OrientationInterpolator DEF='Animation'
                key='0.0 0.5 1.0'
                keyValue='0.0 0.0 1.0 0.0 0.0 0.0 1.0 2.1 0.0 0.0 1.0 0.0'/>
    <ROUTE fromField='touchTime' fromNode='Clicker' toField='startTime' toNode='Timer' />
    <ROUTE fromField='fraction_changed' fromNode='Timer' toField='set_fraction' toNode='Animation'/>
    <ROUTE fromField='value_changed' fromNode='Animation' toField='rotation' toNode='TRANS'/>
    …
</X3D>
```

Figure 4 illustrates how the interaction described in this sample file is managed by the Scene Graph Adapter.

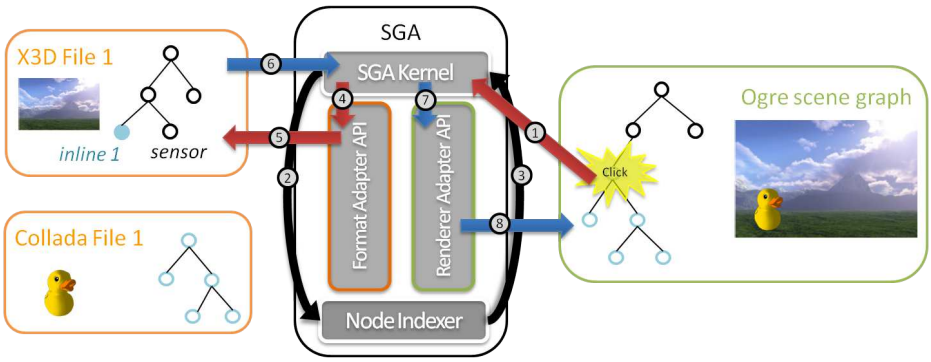

Figure 4 Through the Scene Graph Adapter it is possible to apply an animation described in an X3D file to a Collada model referenced by an inline node.

To start, the mouse pressed event is caught by the rendering engine which transmits it to the Scene Graph Adapter kernel along with event information (1). The kernel then interrogates the node indexer (2,3) to know which format scene graphs and more precisely which node within them are concerned by the event. Once a format wrapper is identified the pressed event is transmitted to it (4,5). The result of the interaction is computed by the format decoder and then the renderer scene graph is adapted according to it (6,7,8). In our example, the orientation of the node will be updated.

It is also possible to use Collada as a container format and references X3D files and nodes inside it using URIs.

Conclusion

We propose a generic architecture that allows mixing 3D formats in a single 3D application whatever the rendering engine used by the application. This architecture enables 3D applications like virtual worlds to share assets without transcoding, which ensures no functionality loss among shared data. Our architecture is suitable for standardization. A format provider can indeed promote its format by creating a wrapper and even several wrapper versions depending on which feature he wants to expose. Similarly a rendering engine promoter can provide a wrapper to enable the loading of any scene-graph-based format: there is no need to provide a plug-in for every format. Besides our solution makes it possible using 3D files as container for 3D files of different types thanks to the flexibility of our architecture. The proposed architecture meets the requirements defined by the MPEG-V standardization group, and is fully compliant with many other standards such as X3D or Collada.

To improve our architecture we plan to enable more specific targets for events than allowed by the inline node. Our goal is to be able to route events defined in an X3D container file to a specific node in a 3D model. We also consider adding other engines to our architecture like a physics engine or a network engine.

References

Bell, J., Dinova, M., & Levine, D. (2010). VWRAP for Virtual Worlds Interoperability. *IEEE Internet Computing*, *14*(1), (p. 73-77).

Bouville Berthelot, R., Royan, J., Duval, T., & Arnaldi, B. (2011). Scene graph adapter: an efficient architecture to improve interoperability between 3D formats and 3D applications engines. *Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11* (p. 21-29) . New York, NY, USA: ACM. doi:10.1145/2010425.2010429

Döllner, J., & Hinrichs, K. (2002). A generic rendering system. *IEEE Transactions on Visualization and Computer Graphics*, (p. 99–118).

Gelissen, J. H. A. (2008, mai). ISO/IEC JTC 1/SC 29/WG 11/N9901.

Gelissen, J. H. A. (2009). Introduction to MPEG-V. *Journal of Virtual Worlds Research, 2*(3).

Hu, S.-Y., & Jiang, J.-R. (2008). Plug: Virtual Worlds for Millions of People. *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems* (p. 787-792). IEEE Computer Society.

McHenry, K., & Bajcsy, P. (2008). *An overview of 3D data content, file formats and viewers.* Technical Report ISDA08-002.

Jovanova, B., & Preda, M. (2010). Avatars interoperability in Virtual Worlds. *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop on* (p. 263-268).

Sons, K., Klein, F., Rubinstein, D., Byelozyorov, S., & Slusallek, P. (2010). XML3D: interactive 3D graphics for the web. *Proceedings of the 15th International Conference on Web 3D Technology,* Web3D '10 (p. 175–184). New York, NY, USA: ACM. doi:10.1145/1836049.1836076

Thompson, C. W. (2011). Next-Generation Virtual Worlds: Architecture, Status, and Directions. *Internet Computing, IEEE*, *15*(1), (p. 60-65). doi:10.1109/MIC.2011.15