Internal and External Scene Graphs: A New Approach for Flexible Distributed Render Engines

Jan Ohlenburg* Thorsten Fröhlich† Wolfgang Broll‡

Collaborative Virtual and Augmented Environments Department
Fraunhofer Institute for Applied Information Technology FIT
Sankt Augustin, Germany

ABSTRACT

Render engines or render APIs are a core part of each Virtual Reality (VR) or Augmented Reality (AR) environment as well as 3D games. Most existing approaches either focus on rendering speed for high frame rates or on the presentation of advanced visual features. However, no approach exists to integrate scene descriptions based on multiple file formats without converting and thereby partly destroying the native scene format.

In this paper we will present our approach of internal and external scene graphs for realizing render engines. We will show how this approach overcomes existing limitations, while still providing decent frame rates and rendering features. By using internal scene graphs for rendering, we use external scene graphs to store format or application specific information, preserving its native structure and content.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types

Keywords: Augmented reality, virtual reality, framework, render engine, distributed system design

1 INTRODUCTION

Today, a large number of 3D toolkits and AR/VR scene graphs exists. Some of them are widely used in a large number of application scenarios. Additionally, several dedicated render engines exist, rather focusing on the optimization of the graphics pipeline throughput for best rendering results and best performance. But there are only a few, which have some native support for multiple file formats. Due to its large availability the ISO standard VRML'97 is widely used as a basis for scene graph representation, resulting in node and scene graph structures very similar to the ones defined by VRML'97. Others are tightly related to the underlying frame buffer, i.e. OpenGL or Direct3D, to be most efficient in pumping the scene graph data into the graphics pipeline, an example for this approach is OpenSceneGraph. Common to all scene graphs using these approaches is the lack of supporting more than one, sometimes even proprietary, file format or scene graph structure. This is usually left to the application developer.

In this paper we will present our approach to support different 3D file formats by using internal and external scene graphs. While the data held in the internal scene graph (RSG) is purely limited to information relevant for rendering, all other scene graph related information is stored in the external scene graphs (XSG). For each file format a different XSG has to be used, which defines how the scene graph information is mapped onto the internal representation. The RSG is designed to ensure that the information mapped onto the scene graph can be retrieved later on without losing semantic information.

*e-mail: jan.ohlenburg@fit.fraunhofer.de
†e-mail: thorsten.froehlich@fit.fraunhofer.de
‡e-mail: wolfgang.broll@fit.fraunhofer.de

2 RELATED WORK

In this section we want to give a short overview on existing render engines and rendering APIs.

OpenInventor™ [4] is an object–oriented toolkit for interactive 3D graphics application and its file format served as the basis for the ISO standard VRML 1.0. When first published in 1992 there was a lack of 3D toolkits supporting interactive 3D application. This motivated SGI to provide a system with a simple event model and a 3D scene database that dramatically simplified graphics programming. Since *OpenInventor* served as a basis for VRML it has a file format on its own, but it is very similar to VRML and so is the internal node structure, resulting in nodes and fields, which are named exactly the same as in VRML, and with the same fields.

Performer or previously *IRIS Performer* [3] is a toolkit originally designed for high performance real–time 3D graphics applications on SGI workstations. It provides a scene graph and a rendering library designed to be most efficient for rendering using multiple processors. As such, its goal is to allow application developers to easily produce high performance rendering. *Performer* belongs to the category of scene graphs with internal representation is tightly coupled to the underlying frame buffer, namely OpenGL, i.e. the information stored inside the nodes can be passed on to OpenGL without conversion of any kind and also OpenGL states can be set for each node.

OpenSceneGraph[1] – closely related and inspired by *Performer* – currently is one of the leading open source scene graphs and a widely used toolkit for VR projects. This is due to its portability and its high performance. It provides an object–oriented framework on top of OpenGL. Although there are a lot of plugins for *OpenSceneGraph* already existing to import and export file formats, but due to the nature of plugins these are external developments. The scene graph does not ensure that a lossless mapping from a specific file format does exist, additionally it does not ensure that all information of the scene graph can be mapped onto a specific file format. Like *Performer*, the scene graph is tightly coupled to OpenGL.

3 INTERNAL AND EXTERNAL SCENE GRAPHS

The major part of our VR/AR framework MORGAN [2] is its render engine. It has been especially designed for the requirements of distributed virtual environments, providing native support for different file formats and scene graph structures. This is achieved by

[1] OpenSceneGraph, www.openscenegraph.org

our approach of using internal and external scene graphs. The underlying concept separates application or file format specific information from pure rendering information. In this context an application may be any program defining its own 3D scene graph structure, e.g. a CAD program. The internal scene graph (RSG) only stores information necessary for efficient rendering while the external scene graphs (XSG) are responsible for all other scene graph related information. Each application or file format has its own XSG, which defines how the scene can be mapped onto the RSG. This is an essential feature to allow users to process files created by other applications and retain a view of the scene native to those applications. E.g. VRML'97 uses `Interpolator` nodes to provide simple animation facilities. X3D provides advanced support for geospatial applications, distributed interactive simulation and humanoid animation. So far, we have implemented two such XSGs for VRML'97 and the CAD system MicroStation by Bentley Systems. In the following paragraphs we will introduce the main mechanisms used in our approach using examples from the VRML'97 XSG.

The VRML'97 XSG builds a complete VRML'97 scene graph and each nodes is mapped onto the RSG using one of the following strategies, *directing mapping* maps the node directly onto one RSG node, e.g. the `Sphere` node. *Complex mapping* is utilized for nodes such as the `Transform` node, which may either be mapped onto a grouping node or onto a geometric node. This depends on whether it is used for grouping several nodes or sub-hierarchies, or for applying a transformation only (since all geometric nodes in the RSG store the local transformation matrix anyway). Additionally, the different fields of the `Transform` node, which define the transformation have to be converted into a transformation matrix before they can be mapped. Finally, some nodes such as the `WorldInfo` node are not mapped at all.

The scene graph structure of the RSG is quite compact, especially when comparing it to VRML'97, resulting in a much smaller number of nodes. E.g. see Figure 1, where a VRML'97 scene consisting of a `Transformation` – `Shape` – `Sphere` combination is mapped onto the RSG. Nevertheless, the XSGs will maintain the original hierarchy. It is very important that the original hierarchy and information structure of a scene is preserved in order to allow for concurrent or subsequent access by an external applications relying on this structure and data. Importing and exporting such scene graph structures usually does not preserve this information.

The complexity of a VRML'97 scene graph, in particular facilities such as prototypes, demands that nodes can have an arbitrary number of fields whose names and composition may even change at runtime. As C++ does not provide a facility to extend classes at runtime, the VRML'97 XSG makes heavy use of templates [1] to provide a convenient interface to fields while retaining important language features such as strict type checking and providing full type safety when accessing fields.

The nodes of the RSG have been designed to allow a mapping between all supported file formats and the internal representation. Thereby, a lot of functionality is provided in the nodes which cannot be mapped onto each file format, to ensure lossless storing of information. The XSGs can take ownership of the internal nodes and define access rights for each field of each node. This way an XSG is able to control consistency of the internal to the external representation. E.g. the VRML'97 `Cone` node is defined by the field *bottomRadius* and *height*, this node is mapped onto the internal node `Frustum`, which allows specifying the number of sides and the top radius as well. In case of a `Cone` the number of sides should be large enough to smooth up the sides, additionally the top radius has to be zero. To ensure, that a `Frustum` can be mapped back onto a `Cone`, it must set the access rights of the *sides* and the *topRadius* fields to read–only. Thus updates to these fields cannot be made directly. Only the owning VRML'97 XSG has write access to these fields.

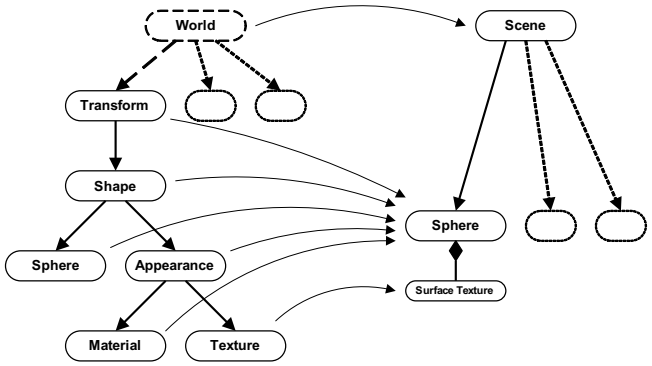

Figure 1: Mapping of a VRML'97 XSG onto the RSG

The same approach has to be used for the transformation fields of a VRML'97 `Transform` node, since not all information can be retrieved from the transformation matrix, e.g. extracting the field *scaleOrientation* is not possible. Therefore, the VRML'97 XSG restricts the write access to the respective transformation field and only permits to change it by setting the fields of the VRML'97 representation.

Another useful feature of this design aspect is the ability of nesting XSGs, e.g. it is possible to nest a CAD model within a VRML'97 XSG. Although, the VRML'97 XSG may not be used to access this sub-hierarchy by scene-internal mechanisms such as events or scripts (i.e. a VRML event `ROUTE` cannot point to a field inside the other XSG), it is possible to manipulate nested scene graphs on the sub-tree level. Thus, a nested XSGs e.g. may be moved similar to any other sub-tree within the VRML'97 scene graph hierarchy.

4 Conclusions and Future Work

In this paper we presented our approach for supporting different scene graph structures and file formats within a single render engine. We achieved this by separating the pure render information within the render scene graph from specific information and structures in external scene graphs. Since the XSGs are responsible for storing application specific data without the need for efficient rendering, they can be fully adapted to the needs of the individual application. On the other hand the RSG can provide the needed flexibility to support a multitude of different file formats by providing an independent and efficient rendering structure. The mapping mechanism, which ensures that data is only stored and manipulated where needed, makes the overall approach efficient and powerful.

As part of our future research efforts we will make more extensive performance tests and compare these results with other frameworks and render engines. Additionally, we will implement an XSG for X3D.

References

[1] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison–Wesley, 2001.

[2] J. Ohlenburg, I. Herbst, I. Lindt, T. Fröhlich, and W. Broll. The MORGAN Framework: Enabling Dynamic Multi–User AR and VR Projects. In *Proceedings of VRST 2004*, pages 166–169, 2004.

[3] J. Rohlfs and J. Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3d graphics. In *Proceedings of SIGGRAPH 1994*, pages 381–394, 1994.

[4] P.S. Strauss and R. Carey. An object-oriented 3D graphics toolkit. In *Proceedings of SIGGRAPH 1992*, pages 341–349, 1992.