Title
The blue-c Distributed Scene Graph

Authors
Näf, M.
Lamboray, Edouard
Staadt, Oliver G.
et al.

Publication Date
2003

Peer reviewed

The blue-c Distributed Scene Graph

Martin Naef[1], Edouard Lamboray[1], Oliver Staadt[2], Markus Gross[1]

[1]*Computer Graphics Laboratory*
Swiss Federal Institute of Technology, Zurich
{naef, lamboray, grossm}@inf.ethz.ch

[2]*Computer Science Department*
University of California, Davis
staadt@cs.ucdavis.edu

Abstract

In this paper we present a distributed scene graph architecture for use in the blue-c, a novel collaborative immersive virtual environment. We extend the widely used OpenGL Performer toolkit to provide a distributed scene graph maintaining full synchronization down to vertex and texel level. We propose a synchronization scheme including customizable, relaxed locking mechanisms. We demonstrate the functionality of our toolkit with two prototype applications in our high-performance virtual reality and visual simulation environment.

Keywords

Distributed graphics, scene graph, collaborative virtual environments, networked virtual reality

1. Introduction

Most immersive VR applications are based on scene graph toolkits which provide a hierarchical object-oriented scene representation. Toolkits used in stand-alone VR systems are usually not immediately suited for distributed applications due to the lack of built-in mechanisms for sharing application data in a consistent fashion across multiple sites. Thus, distributed scene graphs have been developed to solve this problem.

We have developed the *blue-c Distributed Scene Graph* (bcDSG) in the context of the blue-c project [5] – a collaborative tele-presence environment with simultaneous acquisition of 3D video and immersive projection. We employ OpenGL Performer [3], which is one of the most widely-used toolkits for high-performance immersive VR applications, as the underlying scene graph.

The bcDSG shared nodes are replicated and fully synchronized down to vertex and texel level. Furthermore, we do not rely on specific notification mechanisms and do not change the API paradigm of the underlying scene graph.

2. Related work

Various methods have been proposed to build networked virtual environments, such as NPSNET, RING, DIVE and DIS/HLA. A detailed overview of these and similar systems can be found in [4]. These systems focus on large-scale virtual environments with synchronization happening at the application level as opposed to the geometric representation. Recently, shared scene graph architectures have been proposed for cluster-based rendering [2]. They are not immediately suitable for distributed applications, though. Architectures more closely related to the approach taken in bcDSG include Avango [6], Distributed Open Inventor [1], and Repo-3D. Our approach differs from these in that we do not change or even create a new programming interface to the scene graph, completely synchronize down to vertex and texel level, and base on a toolkit that provides high rendering performance, multi-processing, and multi-pipe support.

3. System overview

The blue-c API provides an application development environment which offers flexible access to all blue-c features, including graphics and sound rendering, device input, 3D video, and scene distribution. These subsystems are provided as services and managed by the blue-c core (Fig. 1).

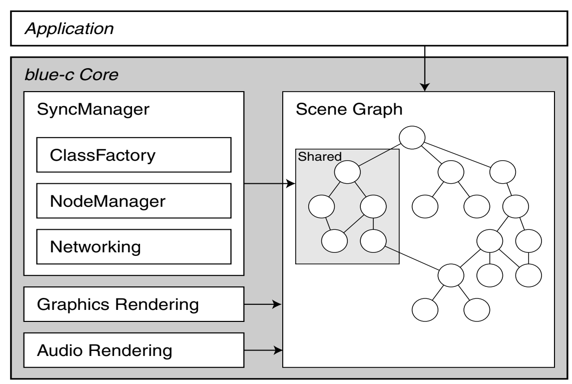

Figure 1: *System overview.*

The blue-c distributed scene consists of the following base components:

- The shared partition of the **scene graph** is built using special node classes. These shared nodes are derived from Performer objects and inherit an additional synchronization interface. This allows us to add the necessary functionality without needing access to the Performer source code. Derived nodes include both the traditional scene graph nodes (such as groups, transformations, and geometry containers) as well as attribute classes (such as materials, textures, and highlighting).
- The **synchronization service** traverses the shared portion of the scene once per frame. It generates, sends, and handles scene operation messages. The synchronization service also includes the class factory, and the node and ID management.

- The **consistency, locking, and ownership management** mechanisms are implemented as part of the synchronization service.
- The **network** interface sends and receives scene operation messages. It also provides session and ID management.

4. Scene operations

The scene graph is kept consistent among the participating sites by sending messages with node modification operations. For identification across sites, each shared node has its own NodeID structure which consists of a main identification number (ID), a generation number, ownership, and serial numbers for both connectivity and attribute state.

4.1 Node status and update messages

Each shared node keeps a set of flags, encoded in a single integer. These flags are:

- *New:* The node has just been created.
- *State Dirty:* Attributes of the node have changed.
- *Connectivity Dirty*: The connectivity of the node has changed.
- *Request Ownership:* The local system wants to become owner of the node. No request has been sent yet.
- *Ownership Request Pending:* An ownership request for the current node has been sent.

The application programmer is responsible for setting the state flags whenever Performer attributes or node connectivity has changed.

During the traversal, scene update messages are created according to the state of the visited node. The following messages are used to keep the scene synchronized: *Create node*, *Update state*, *Update connectivity*, and *Delete node*. Additional messages for requesting and passing ownership are used to guarantee a consistent state in the presence of concurrent modifications of the scene. State serial numbers are used to handle inconsistent message ordering between different sites.

The state of a single node is always transferred as one atomic operation. This granularity proved to be well suited for most applications.

5. Consistency and locking

For the blue-c distributed scene graph, a relaxed locking scheme based on object ownership was implemented. By default, it provides immediate response to user interface actions by allowing for local modifications of nodes before ownership is acquired. For scenarios where a strict locking scheme is appropriate, the application developer may easily change the semantics as required.

6. Networking

The consistency of the distributed scene graph is guaranteed by a reliable transmission of the scene graph operations from each participating site. However, no total ordering of the scene graph operations is required. In order to fulfill real-time requirements, we implemented an appropriate scheme for reliable data transmission based on the connectionless and unreliable UDP protocol and on explicit positive and negative acknowledgements. Our system is based on the TAO/ACE framework (http://www.cs.wustl.edu/~schmidt/TAO.html) and on the CORBA A/V Streaming Service.

7. Example applications

We implemented two collaborative example applications based on the blue-c distributed scene graph: A distributed chess, and a collaborative painting tool[1]. The applications take advantage of the full scene graph synchronization, including texture updates, and use different locking schemes.

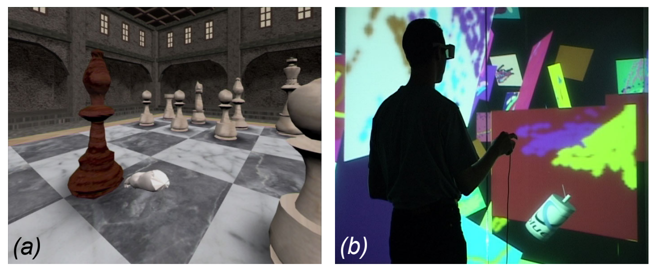

Figure 2: *(a) Distributed chess. (b) Collaborative painter.*

Acknowledgements

We would like to thank all members of the blue-c team for many inspiring discussions. This work has been funded by ETH Zurich as a “Polyprojekt” (grant no. 0-23803-00).

References

[1] G. Hesina, D. Schmalstieg, A. Fuhrmann, and W. Purgathofer. “Distributed open inventor: A practical approach to distributed 3D graphics.” In D. Brutzman, H. Ko, and M. Slater, editors, *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 74–81. ACM Press, 1999.

[2] D. Reiners, G. Voss, and J. Behr. “OpenSG - Basic concepts.” 1. OpenSG Symposium, 2002.

[3] J. Rohlf and J. Helman. “IRIS Performer: A high performance multiprocessing toolkit for real-time 3d graphics.” In *Proceedings of SIGGRAPH 94*, ACM SIGGRAPH Annual Conference Series, pages 381–395, 1994.

[4] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. ACM Press - SIGGRAPH Series. Addison-Wesley, 1999.

[5] O. G. Staadt, A. Kunz, M. Meier, and M. H. Gross. “The blue-c: Integrating real humans into a networked immersive environment.” In *Proceedings of ACM Collaborative Virtual Environments 2000*, pages 201–202, San Francisco, Sept. 2000. ACM Press.

[6] H. Tramberend. “Avocado: A distributed virtual reality framework.” In *Proceedings of IEEE Virtual Reality 99*, pages 14–21, 1999.

1. The applications are illustrated in the video proceedings.