The blue-c Distributed Scene Graph

Martin Naef, Edouard Lamboray, Markus Gross
Computer Graphics Laboratory, ETH Zurich

Oliver Staadt
Computer Science Department, UC Davis
Context: The blue-c

- Collaborative Immersive Virtual Reality Environment
- Provide remote collaboration features
  - Shared, synchronized virtual world
  - Render partners using 3D video streams
  - Concurrent rendering and acquisition
Overview

- Motivation and Related Work
- Design Goals
- Scene Distribution
  - System integration
  - Object model
  - Messages
- Locking and Consistency
- Networking
- Example applications
- Conclusions
Motivation

• Support collaborative VR applications
• Distributed manipulation of shared geometry
  – Generalized concept for a large class of applications
  – No application-specific synchronization protocols
• Support blue-c features
  – 3D video rendering
  – Multimedia integration (2D video, audio)
  – Animation
Related Work

- Large-scale virtual environments
  - DIS/HLA, RING, DIVE, MASSIVE
- Cluster rendering
  - OpenSG, Syzygy, WireGL/Chromium
- Distributed scene graphs
  - Distributed Inventor, Avango, Repo3D
**bcDSG: Design Goals**

- Distributed, synchronized scene graph
  - Including all attributes
- Avoid reinventing the wheel
  - Use existing technology
  - Support legacy code
- Based on familiar tools
  - C++ programming language
  - Proven scene graph (e.g. Performer)
bcDSG: Overview

• Part of the blue-c API
• Partitioned scene
  – Shared and local partition
• Synchronization service
  – Traverses the shared partition once per frame
  – Generates and handles update messages
• Networking
  – Built upon blue-c communication layer (bcl)
Object Model (Nodes)

• Shared nodes
  – Derived from Performer Nodes
    • Add interface and state using multiple inheritance
    • Keep Performer interface in place
  – Streaming interface
    • Read / write state
    • Read / write connectivity
Object Model II

- State data per node
  - Unique node ID
  - Ownership information
  - State information
    - State dirty flag and serial no.
    - Connectivity dirty flag and serial no.
    - Ownership requested
Components

• Sync Manager
  – Scene traversal once per frame
    • Checks flags, generates and handles messages
  – Initiates and handles network connections
    • Using the blue-c communication layer

• Node Manager
  – Maps IDs to nodes

• Class Factory
  – Create class instances by name or type
Synchronization Messages

• Create Node
  – New ID and class name is transmitted
  – Handled by the class factory
• Update State
  – Transmits internal node state / attributes
  – Updates ownership information
  – Serial number ensures consistency
• Update Connectivity
  – Similar to Update State
  – Includes ID list of child nodes
• Delete Node
Locking Scheme

- Concept: Node ownership
- Messages
  - Request ownership
  - Pass ownership
  - Decline ownership request
    - Application intervention
Locking Scheme II

- Default behavior: Lazy locking
  - Local site may modify node regardless of ownership status
  - Updates are only sent by the owner
  - Local changes may be overridden by node owner
Per-Frame Evaluation

Visit node

Dirty? no

Owner? no

req. pending? yes

send update

clear dirty flag

Done

req. pending? no

request ownership

yes
Locking Scheme III

• Customized locking scheme
  – Application may decide not to modify a node without having ownership (strict locking)
  – Ownership may be requested at any time
  – Application may grant or decline ownership requests by providing a hook
Consistency

- Network assumptions
  - Reliable transmission
  - Guaranteed ordering between two machines
  - No global ordering
- Conflicting updates
  - Only the owner sends updates
  - Serial number scheme
    - Outdated update messages are discarded
- Delayed processing of updates
  - Avoids global ordering requirement
Networking

- Connection management
- Reliable transport and multicast
- ID management
- Time service

- Based on CORBA: ACE/TAO
Example Application

• Collaborative Painter
  – Place quads anywhere in 3D space
  – Paint on textures
  – Geometry and textures are synchronized
Example Application

- Distributed Chess
  - Static background geometry
  - Shared chessboard and pieces
  - Strict locking for chess pieces
  - No chess application logic
Video: Chess
Conclusions

• Distributed scene graph
  – No changes in the Performer API
  – Complete synchronization including vertex and texture data
  – Relaxed locking scheme, customizable
Future Work

• Enable late joining sites
• Finer granularity for updates
• Completely transparent operation
  – Automatic setting of flags
• Network congestion control

• Applications
Questions?