VE software support systems and standards (VESSSS!)  

Doug A. Bowman
A VE software system should manage display devices, input devices, trackers, and the environment.
Some VE software system goals

- Device-independence
- Portability
- Extensibility
- Configurability
- Flexibility
- Support for novice developers

*Device-independence – application code should compile and run without modification on different displays (CAVE, HMD, Desktop… cluster, multi-pipe, etc), and with different input devices and trackers.*

*Portability – application code should compile and run without modification on a variety of platforms (Linux, Windows, Mac, SGI, etc)*

*Extensibility – it should be easy to add new capabilities to the system (displays, devices, toolkits, etc)*

*Configurability – it should be possible to modify system behavior (tracking coordinate system, target display, input devices) without modification of application code*

*Flexibility – system should not be over-restrictive, should support a variety of middleware, toolkits, and libraries*

*Support for novice developers – system should be easy for novices to learn and use (default behavior, documentation/tutorials, easy to install/configure)*

*Other considerations – VE specific issues (?), support for collaboration*
A VE system is made of several components:
* Hardware like displays, input devices, and tracking systems
* Software libraries built on top of hardware, like OpenGL and DirectX for graphics hardware, SDK’s like those provided by Intersense for their trackers
* Middleware and toolkits build on top of software libraries, like Performer and OpenSG which implement scene graphs. Usually this is the software that handles loading model files generated by modeling software (obj, x3d, md2, etc) into the environment.
* Finally, the application program is built on top of the toolkits and libraries
Two strategies for system design

- **Monolithic**
  - One toolkit does everything
  - All parts are required
  - Tightly controlled
  - e.g. SVE

- **Component-based**
  - Multiple parts / modules
  - Use what’s needed
  - Extensible
  - e.g. VR juggler, DIVERSE

Two approaches to VE system design:

1) Monolithic (e.g. SVE) – one toolkit provides everything, *must* use the parts provided by the toolkit

2) Component-based (e.g. DIVERSE and VR Juggler) – multiple modules, use only what is needed, easy to extend, natural fit for VE systems?
Simple Virtual Environment library (SVE)

- “One size fits all” VE toolkit
- Goals:
  - Simple VE apps are simple to program, but complex apps are also possible
  - Rapid prototyping, desktop development
- C/C++, OpenGL
- Scenegraph-based
- Programming based on callbacks

SVE is to VE programming as GLUT is to OpenGL programming. Developed by Drew Kessler at Ga Tech. Goals were to

(1) make simple apps easy to build, but also leave room for advanced use, and
(2) support rapid prototyping and desktop development

Set your configuration, load your models, set your callbacks, and run the main loop.
SVE features

- Supports a range of input, tracking, and display devices
- Provides default navigation and rendering, but allows these to be changed
- Run-time configuration of devices, options
- Hardware configuration details not critical to the application programmer

supports a range of input, tracking, and display devices (like?)
default navigation and configuration

extensive configuration system (displays, trackers, world, materials)
A common Application Programming Interface (API) to interactive graphics and Virtual Environment (VE) programs. By using DIVERSE a program can be run on everything from a CAVE™ to a desktop workstation without modification.

A common interface to VE oriented hardware such as trackers, wands, joysticks and stereo glasses.

A “remote shared memory” facility that allows data from hardware or computations to be asynchronously shared between both local and remote processes.

open-source software, available for download free of charge.

runs on the SGI and Linux.
DIVERSE’s main components

- Graphics:
  - DIVERSE graphics interface for Performer™ (DPF)
  - DIVERSE graphics interface for OpenGL (DGL)
- Devices and data:
  - DIVERSE ToolKit (DTK)
- Cluster support:
  - DIVERSE Adaptable Display System (DADS)

The DIVERSE graphics interface for Performer™ (DPF) provides a framework to implement 3D Virtual Environment (VE) applications, DGL for OpenGL and presumably any OpenGL based toolkits like OpenSceneGraph.

The DIVERSE ToolKit (DTK) provides access to local and networked VE interaction devices and data.

DADS uses DTK shared memory to communicate between nodes, automatically replicates processes started on console on every node.

*Interesting configuration system: C++ is used as a configuration language. Configuration files are compiled as plugins.
The structure of the Crane Ship Simulator is a cluster of cooperating programs. They are coupled by data structures that retain current physical state information in inter-process shared memory. This design facilitates a modular structure for the system. All the programs in the simulator run asynchronously with respect to all other programs. Some programs simulate physical dynamical models whose cyclic solver is synchronized to real-time, others use the current shared physical state information from the shared memory. Consequently, the need for process synchronization is replaced indirectly by the sampling of physical state values that are being modeled. If the coupling between two processes is large, then the shared data is queued and the reading process interpolates values for the given time (or another state variable) that it is reading, so that continuity in the coupling may be better preserved.
VR Juggler is DIVERSE’s main competition. It is component-oriented and cross-platform (windows, mac os x, linux, SGI)

* VPR implements cross-platform networking and threading (as well as some other magic with design patterns)
* Tweek is a distributed model view controller implementation built on CORBA (good for collaborative VEs)
* JCCL is VR Juggler’s XML-based configuration system (a little overwhelming)
* Gadgeteer is the device interface (trackers, wands, etc), transparent clustering
* Sonix is an audio framework

*** VR Juggler “proper” is a small set of application classes built on top of all these components

Started using VR Juggler on Gigapixel display as a higher-performance alternative to Chromium. Ported the Stratagus game engine and TerraServer Blaster to VR Juggler.
Pros of VR Juggler:
- Easy to install
- Simulator system lets you prototype VE applications on the desktop (maps keyboard and mouse to head position)
- Transparent cluster support

Cons:
- Lack of default behavior
- Configuration system is a little intimidating
Standardization

- Typically standardized components:
  - file formats / specification languages
  - libraries
- For multi-user VEs a network-related standardization is also needed
- Portability and performance issues
Virtual Reality Modeling Language (VRML)

ISO/IEC 14772, the Virtual Reality Modeling Language (VRML), defines a file format that integrates 3D graphics and multimedia. Conceptually, each VRML file is a 3D time-based space that contains graphic and aural objects that can be dynamically modified through a variety of mechanisms.
VRML characteristics

- Based on a scene graph (object tree)
- Implicitly establishes a world coordinate space for all objects
- Explicitly defines and composes a set of 3D and multimedia objects
- Object behaviors
- In-lining of other VRML files
- Hyperlinking to other files & applications
- Established Internet and ISO standards for other file formats
- Compact syntax
- Declarative specification language, NOT a toolkit or library!
Extensible 3D (X3D)

- A software standard for web and broadcast multimedia content
- Universal interchange format - based on XML
- Successor to VRML
  - New features
  - Additional data encodings
  - Improved APIs
  - Stricter conformance
  - Modular architecture
  - Extensibility
- The new ISO standard
X3D design goals

- Support new graphical, behavioral and interactive types
- Modularize the architecture: Lightweight core + add-on components
- Define subsets of the specification ("profiles") that meet different market needs
- Allow for the specification to be implemented at varying levels of service
- Allow for the specification to be extended more easily
- Eliminate, where possible, unspecified or underspecified behaviors
X3D features

- 2D and 3D graphics
- Animation
- Spatialized audio and video
- User interaction
- Navigation
- Simulation
- User-defined objects
- Scripting
- Networking
VRML- and X3D-based VEs

VRML and X3D are just specification languages, so how are they used in the real world?

Development:
- Write code directly, and/or
- Use a software package that directly outputs the correct code (e.g. VizX3D)

Usage:
- Code must be “run” by a “browser” or “player” (e.g. Cortona, Xj3D)
- Historically, most browsers assume web-based content, but recently some tools have started to support immersive VEs (e.g. JINX)
Supporting novice developers

- Simple model loaders with navigation
- Default interaction techniques
- Prototyping on the desktop
- Visual programming?
- Immersive programming?
Conclusion/Discussion

- Comment on your experience using VE software systems.
- In your opinion, what are the most important features of a VE system?
- What other segments of VE systems should be standardized?
Acknowledgments

- Denis Gracanin, lecture Spring 2000
- John Kelso/Lance Arsenault, DIVERSE presentations
- Drew Kessler, lecture from GT VE class
- Nicholas Polys, X3D information