

Challenges for the Future of Standards for (Immersive) Virtual Reality

Anthony Steed
University College London & Electronic Arts

IEEE VR Workshop on the Future of Standards for Immersive
Virtual Reality
10th March 2007

Background

What we did

- Built immersive virtual for about 15 years
- Members of our lab have built demos or applications with these systems:
 - dVS/dVISE, AVS, OpenSG, OSG, Performer, GL, OpenGL, Torque, RakNet, CrystalSpace, OGRE, Alice, CAVElib, Panda3D, Chromium, Direct3D, Inventor, COIN, Java3D, JoGL, MASSIVE1/2, EQUIP, SecondLife, Quake1/2/3, PowerVR, Avango, DIVERSE, VRJuggler, Unreal, Bamboo, TreacleWell, LightSpace, VTK, VRML, Oblivion, GoogleEarth, Cal3D, Carmel, Renderware, MetaVR, XVR, Ghost, GKS, VRPN, Superscape, Core, ACE, WorldToolkit, XP, CoVISE, Flow, XGL

Why So Many?

- We were the (co-)developers
- Collaboration with developers
- Often demos of the toolkit are easy to repurpose
- Ease of fit to application domain
- It came with the kit

- Fashion
- Necessity due to obsolescence

One “Standard” Platform*

- *For some definitions of standard
- Got most mileage out of SICS’s DIVE (Distributed Interactive Virtual Environment) system
 - UCL co-developer from 1996-2000
 - Maintainer of (immersive) ports since 2000
- <Demos>
- These demos are 5-10 years old (we do have newer demos!)

Why was DIVE successful for us?

- It’s a platform, not some toolkits
- Scripted architecture, with plugin extension
- Graphics were good for their time
- Scaled well
- Built in audio, video
- Networking was “code-less” (distributed scene-graph)
- The DIVE client is (optionally) its own authoring environment
 - Interactive script editing
 - World editing
 - Scripting access is key
- The world authoring and windowing interface scripting are integrated

What does success mean?

- World authoring was isolated from interaction authoring
- Over the years, DIVE as a platform, has been extended
 - New audio protocols
 - New graphics (OpenGL, Performer, OpenSG)
 - New interfaces (Custom fastrak, CAVElib, VRJuggler)
- But core world & scripts have not needed to change

Why was it successful really?

- Two abstractions and two implementation strategies were key
- Abstractions
 - Vehicles
 - Avatar
- Implementation strategies
 - Open event system
 - Platform APIs are reflected in the scripting

Vehicle Abstraction

- You never need to see tracker events
- A “vehicle” moves the user around the world
- Issues events
 - Grab, select
- Multiple vehicles can be active
 - Mouse, keyboard, “game style”, immersive tracker
- Activation of vehicle is a start and run-time option

- Isolates world author from any knowledge of interface devices (unless they really want to know)

Avatar Abstraction

- Related to the vehicle
- All avatars have head, hand, foot, eyes, ears
- Geometric representations of these is optional
- They are in a skeletal hierarchy
- ALL rendering is dictated by the hierarchy
- Interaction events issue from body parts (selection and grasp from eye or hand)

- Isolates world author from knowledge about form of interface (e.g. immersive or not)

Open Event System

- Architecture uses an open publish/subscribe mechanism internally
- The main APIs, plugins, script can all publish and subscribe
- Order of events in a frame loop, and more importantly a world life-cycle, are known
- Distribution mechanism is an extension of the event system
 - Careful tagging of events (high-level events, de-synchronization points) makes the system scale extremely well
- Events can be serialised to disk for replay

API Reflection

- Don't second guess the world author
- All media mechanisms and scalability are scriptable
 - Scene paging
 - Scalability mechanisms
- Only rendering is automatic

So why are we not using it so much now?

- Fashion
 - TCL Obsolescence
 - Maintenance of old demos too hard
- No clustering
 - No time to do it!
- Not (and never will be) Open Source
 - There used to be a lot of IP around
- Needs maintenance
 - New shading models
 - New content formats
 - Tidying up and new documentation
 - Paging and resource management are weak

Where was the architecture wrong?

- No object-orientation in the APIs and scripting (isolation and introspection of function on types would have been useful, but this doesn't mean just wrap SWIG round a C++ object hierarchy)
- Object system contains too many legacy features
- Network subsystem was not flexible enough to support multiple distribution strategies
- No atomicity in distribution
- Content pipeline was primitive
- Publish/subscribe is not a good mechanism for heavy-data lifting internally
 - Certain things bypassed it, such as spatial queries

Towards a New Platform

- We want many of the same features
- However, the problem is getting harder
 - No more “nice” programming platforms (SGIs were extremely good for us)
 - Now expected to “do more” with limited platforms
 - Art & behaviour expectations are higher
- Game engines are not a panacea
 - Problems are the same but graphics are shinier!
 - With a game engine, you take the whole thing, or nothing, they are complete platforms, which is not commonly what an experienced group is looking for

Comments on Game Engines

- Games industry has many of the same problems we have
- Engines are built for a game and thrown away
- Engines that have iterated are very complex (e.g. Unreal)
- They much more effort to customise the engine for an effect
- Engine design is done as much with an eye on the content production pipeline as it is on shininess
- The bottleneck of content production is NOT art but marking up the world assets to describe behaviour

Future

What would we want to standardise?

- Libraries
 - VRJuggler
 - DIVERSE
 - CAVElib
 - ...
- Content
 - VRML
 - OBJ
 - Collada
 - ...
- Scripting
 - Python
 - TCL
 - Lua
 - ...
- Protocols
 - RakNet
 - HLA/DIS
 - Custom
 - Verse
 - ..

Wish List for Next Platform

- Well defined architecture with core features
 - Real-time display and interface management
 - Robust separation between APIs (immediate mode) and Modules (event publishers and subscribers)
 - Reflection
- Distribution built in from the ground up, but around event system
- Scripting support
 - Probably python for programmers and Lua for non-programmer (there is no reason why this isn't possible)
- Broad testing facilities
- Some authoring and mark-up support within the engine
 - Asset management is key, both online and off-line
- Favourite abstractions

How to Get There?

- Keep building toolkits, but make sure they decompose existing functionality and focus on doing a small number of jobs well
- Developing community awareness of best of breed technologies
- Try other people's systems and try to understand not only what it does, but why it was done that way

To that end ...

- We have worked and are working on more general implementation of the “vehicle” concept
- Avatars are becoming increasingly important
 - Piavca (piavca.sourceforge.net)
 - Some exciting announcements soon hopefully
- Want to propose (and offer to manage) a knowledge base of VR software
- One thing we could standardise
 - Markup of models, for behaviour (doors, paths, cover, triggers)
 - But ... need to demand tools maintain markup or ...
 - EA, for example, “standardises” a backend database tracker...

Some bigger challenges

- Authoring of 3D behaviour is a huge challenge
 - Especially testing/debugging
- There has to be a new paradigm for describing 3D behaviours
 - State machines isolate behaviour, but make general changes hard
 - Data flow is attractive, but deals badly with instantiation by example
 - Dealing with synchronous and asynchronous behaviours is very hard (do you thread or not?)