AAA

toni ...

Contents

Abstract

We present an inherently networked entity system for creating multiuser applications, including networked games. The entities aggregate components with attributes which are automatically synchronized in a client-server setup. So called entity-actions can be called both locally and remotely, they are a simple form of RPC. This system is implemented in the open source realXtend Tundra SDK which is a complete platform for multiuser 3d applications.

The goal is to make creating efficient multiplayer games easy, with for example just JavaScript logic code, without the need to invent own network messages for simple functionality. Custom messages are supported for efficiency in more complex cases.

Besides the generic attribute sync, the Tundra SDK includes custom messages and logic for moving rigid bodies, utilizing the physics module also on the client side and by implementing (XXX basic sensible things / nice clever tricks / best practices from the literature / something). Measurements from these optimizations are presented -- with the optimizations the maximum number of moving objects in a scene went up by N.

Introduction

Writing a networked multiplayer game is easily(urghXXX?) more complex than a local game, part for the logic handling for multiplayer in general but typically largely for the networking part. And it involves dealing with the connections etc.

The realXtend Tundra SDK provides a decent API and a solid platform for networked 3d applications. It originates from the realXtend virtual worlds project, but has always been developed to be used for games as well. **and has been largely developed by a local game company, Ludocraft Ltd., also after the initial project -- their recent Circus demo is also the best simple Tundra game demo now.

Tundra applications are written against the Tundra Core API and utilizing the Entity-Component scene model. The platform takes cares of the networking basics, so that an application developer does not necessarily need to even know about connections, not to mention dealing with implementing own server and client applications somehow. When the application is run on a server, all clients due to the nature of the shared environment participate in the same session and see everything identically (and when they don't its' a bug and we must file an issue :p) <-- scrap that stupidity, it's just like scripting in any scriptable MMO .. or modding a FPS, using engine like Unreal or Quake. so can just put briefly and ref to something perhaps too, for clarity hopefully).

This paper presents the Tundra application plaftorm, and analyzes / reviews (/smtXXX) it for networked game development. A main research focus here is on the API and scene mode in the design level for an application developer point of view, but also the quality of the current Tundra open source runtime is examined. (XXX? robustness and efficiency of the).

...

background / related

- use of other multiplayer engines: FPS, but also the unity plugins and bigworld etc.
    - the APIs of those, the app dev model: are e.g. connections dealt with at all typically etc? how is data synched (or is it even needed in those, server logic?, scripts?). how do messaging things work (room for improvement in Tundra perhaps?)

The Tundra SDK

- API
- Module System
- Core functionality: Ogre3d, Qt, kNet

Case at hand: is it good to dev an app, a multiplayer networked game, with the entity model and the api overall? -- research question how to answer? illustrate a set of examples -- two were already in the IEEE paper, and are kind of nice? XXXwhat here -- short treatment of those, and then some new example(s)? analysis of the Circus code? -- make this 'the circus .. and lvm, paper' ?

Pong - the Hello World of networked multiplayer?

pong - oh, 'the pong paper' ? :o -- there's a ref for it! deals with latency etc, whether&when to do what and where in a networked setup.

Notes / References

-- about that work -- in a diff paper from the group: "From the result we prove that the decorator feedback only had the positive effect on the lower delay condition but not in the high delay condition." """

Greger Wikstrand, Lennart Schedin and Fredrik Elg [9] gave three hypotheses before they did their Pong game experiment in a simulated mobile phone: ”Delay effort”, ”De- lay action” and ”Delay performance”. The experiment put eyes on significant effects on four independent variables: enjoyment, mental effort, net distance and paddle move- ---

Avango is a framework for building distributed virtual reality applications. It provides a field/fieldcontainer based application layer similar to VRML. Within this layer a scene graph, based on OpenGL Performer, input sensors, and output actuators are implemented as runtime loadable modules (or plugins). A network layer provides automatic replication/distribution of the application graph using a reliable multi-cast system. Applications in Avango are written in Scheme and run in the scripting layer. The scripting layer provides complete access to fieldcontainers and their fields; this way distributed collaborative scenarios as well as render-distributed applications (or even both at the same time) are supported. Avango was originally developed at the VR group at GMD, now Virtual Environments Group at Fraunhofer IAIS and was open-sourced in 2004. An in-depth description can be found in here.