L3DGEWorld 2.3 Hierarchy & Room Reuse Documentation

Lucas Parry
Centre for Advanced Internet Architectures, Technical Report 080222D
Swinburne University of Technology
Melbourne, Australia
lparry@swin.edu.au

***Abstract*—L3DGEWorld 2.3, a product of the L3DGE project, is a data visualisation tool based on the OpenArena derivative of the Quake III Arena game engine, being used in the monitoring and control of networks. This technical report briefly describes some previously undocumented functionality of L3DGEWorld 2.3. It will describe how we have implemented a multi-level hierarchy, which could be used to represent networks, hosts and flows, and the room re-use technique used to achieve this within the limitations of the engine.**

***Index Terms*—L3DGEWorld, hierarchy, room re-use.**

I. Introduction

One of the concepts that we wanted to explore using the L3DGEWorld engine as a part of the L3DGE project [1], was a multiple level hierarchy as previously mentioned in [2] and [3].

"To aid simplicity of view, we propose representing network hierarchy through aggregation. At the highest level, avatars represent entire subnets (cubes), to further investigate the operation of a subnet, the user moves the virtual world camera into the subnet cube object. Inside the subnet cube all avatars are host objects (pyramids). Further moving into a host object reveals the objects representing the particular host's connections (cones). With this three tier system an overall view can be gained of the network, but it also allows for further detail to be gleaned when required. Where possible, similar metaphor mappings are used at each level of the hierarchy. (For example, the size of an avatar representing a subnet might be proportional to the number of connections entering or leaving the subnet. This would be consistent with the use of size to represent the number of connections in and out of an avatar representing a host." [3]

Multi-level hierarchy functionality was first implemented in L3DGEWorld 2.2 [4], but due to time constraints, lack of flexibility and a lack of suitable documentation, it was not an announced feature in the release.

Understanding of this document assumes the reader has read and understood the L3DGEWorld 2.3 Input & Output specifications [5].

II. Why it's needed?

A big concern for us throughout development has been the total amount of data being stored within the "gamestate". Gamestate is where everything clients need to know about the "world" is stored, which includes the configstrings used by us to get string data to clients. When a client connects it is sent the entire gamestate, and from then on is sent incremental updates informing them what has changed.

During development we found that if the gamestate grew too large, the inital update that was sent to clients became too big to fit in a single packet and became fragmented, and clients did not handle this causing them to be unable to connect to the server. This would often occur when using a large number of entities with moderate amounts of data associated with each of them.

In a model as described in the introduction, the number of entities required to represent all of the levels increases exponentially as we add hosts to the various levels. For example, a map with 3 subnets, each containing 3 hosts, each containing 3 connections, we would need 36 hosts and 9 bottom level rooms, but for a map with 6 subnets, each containing 6 hosts, each containing 6 connections, we would need 258 hosts and 36 bottom level rooms.

Instead, we can designate a number of identical physical rooms in a map (which we will refer to as "reusable rooms") to be re-used, in order to "virtually" represent all of the bottom level rooms (which we will refer to as "virtual rooms"), none of which physically exist in the map (as seen in Figure 1).

The number of designated reusable rooms also becomes the limit for the maximum number of simultaneous administrators that can be supported without breaking the illusion that all the virtual level rooms really exist. By re-using 4 rooms to represent all of the virtual rooms in the last example, we reduce the number of entities required to just 66, and are able to emulate the 36 bottom level rooms with just 4 reusable rooms.

One of the positive side effects of re-using rooms is that the gamestate only contains configstrings for the rooms in use, with different configstrings being inserted into the gamestate as administrators move into different rooms. This helps keep the size of the gamestate to a minimum and greatly reduces the likelihood that clients will receive fragmented gamestate packets.

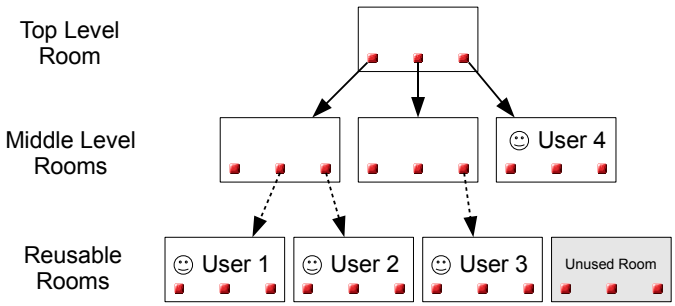

Fig. 1. An diagram of hierarchy with room re-use enabled

III. How Does It Work?

The first customisation we needed to make to the engine to get a hierarchy up and working was the ability to touch a host and be teleported inside. This was achieved by setting a "target" key for each entity, then creating a function to teleport any player who touches the entity to the `misc_teleporter_dest` with that "targetname". In our implementation, top-level entities have targets set to "Middle1", "Middle2", etc, corresponding to the middle level rooms they contain.

The upper two levels of the hierarchy are achieved solely with this modification. Each top-level entity has a corresponding middle-level room to which it teleports users to, and all middle level rooms have an exit teleporter teleporting users back to the top-level room. We were also able to make a very small scale three level hierarchy using only this technique, but the amount of work creating the maps was very high (with respect to the small map size) and we were rapidly approaching the limitations of the gamestate size.

Consequently, we chose a more flexible approach. Middle-level entities all have their target set to "L3DGE", and have an additional key "roomNo" that specifies the virtual room relating to this entity (starting from 1 for the first virtual room). When a player touches a middle-level entity, the function called sees that the "target" key is set to "L3DGE", and drops into special room re-use function.

This function takes the entities "roomNo" key, and looks through the list of the reusable rooms to see if any of them are already being used to virtualise it.

If a matching room is found, the player is teleported into the virtualised room, and a counter used to track the number of users in that room is incremented.

If a matching room is not found, the first unused room is selected, the configstrings and entity positions for that room are updated to correspond to those of the virtual room and the player is teleported in. The counter tracking the number of users in the room is set to 1.

If a player leaves a room, dies in a room or disconnects from the server, the counter tracking number of users is decremented by 1. When the counter hits 0 the room is flagged as being unused and can then be re-used to virtualise another room.

The virtual rooms may be represented by a number of separate reusable rooms, each in a different location. Because of this we need to store the entity positions for virtual rooms as an offset from the entities default positions. This allows us to correctly position entities at their custom positions, even though the absolute positions of the entities are entirely different. To prevent absolute coordinate files being interpreted as relative coordinates, we write relative coordinates into a different directory, 'hosts/<mapname>-r/'.

The exit teleporters in reusable rooms should have their target set to "middle". Each middle level room has a `misc_teleporter_dest` with the targetname "middleX" where X is the middle level room number. When the teleporter function sees target set to "middle", it performs some simple math based on the virtual room the player is marked as being in and the number of entities per middle level room, and determines which middle-level room the player should be returned to.

IV. Enabling Room Re-use

The demonstration map that is supplied with L3DGEWorld 2.3 is named "hierarchy-demo". "hierarchy-demo", and any other room reuse compatible maps will automatically notify the engine that they need room reuse enabled, and specify the number of entities

at each level, number of reusable rooms, etc. Note that while in room re-use mode, the detailed info window can only be accessed using the "host inspector" tool.

V. Making a map

- When an entity has a key/value pair named "target", touching that entity will teleport the player to the `misc_teleport_dest` specified in the key, unless it is set to "L3DGE" as discussed in the next point
- Middle level entities should all have their "target" set to "L3DGE". When the L3DGEWorld sees this it will invoke the code that selects and sets up a room to be re-used and teleported into.
- Middle level entities should have a key/value pair named "roomNo" with it's value set to the virtual room that corresponds to the entity.
- Exit teleporters in reusable rooms should have their target set to "middle"
- Reusable room `misc_teleporter_dest` entry points should have their "targetname" key set to "roomX", where X is the reusable room number.
- Middle level `misc_teleporter_dests` must be named "middleX" where X is their middle level room number.
- Entities in reusable rooms should have a key named "room" which is also set to the reusable room number.
- Entities in reusable rooms should also have a key named "reuseid" set to the entities number within the room. ie. In a room with 4 entities, the reuseid for each will enumerate 1 through 4.
- Entities must be in a specific order in the .map file. Static entities must come before the reusable entities. We recommend that top level entities come first, then the middle level entities, and finally the reusable room entities. This can be done using a plain text editor. As in all L3DGEWorld maps, the order that entities occur in the .map file defines the sequence of host id's that will be assigned to the entities in game.
- L3DGEWorld 2.3 supports up to 12 middle level rooms. Creating a map with more than 12 middle level rooms will require additional modification to the L3DGEWorld engine.
- There are several custom key/value pairs that should be attached to the `worldspawn` entity. To enter these in GtkRadiant, select a brush (for example the floor) then press 'n' to bring up the entities window, then add each of the following keys.
  - "roomReuse" should be set to "1" to let the engine know this is a room reuse map.
  - "numStaticEnts" specifies how many static entities are on the map (ie. All those not in a reusable room).
  - "numEntsPerMiddleLevelRoom" specifies how many entities are in each middle level room.
  - "numEntsPerReusableRoom" specifies how many entities are in each reusable room.
  - "numReusableRooms" specifies how many reusable rooms exist on the map. L3DGEWorld 2.3 theoretically supports up to 16 reusable rooms on a single map.

VI. Limitations and Disadvantages

The engine needs to know exactly how many real and re-used entities are on the map, how many entities are in each reusable room, and how many entity are in each middle level room, in order to be able to work out (a) which set of hosts to put in the virtual rooms and (b) which real, middle level room, to return the player too when they exit a virtual room. This all has to be coded into the map by the map maker and is completely manual.

Middle level rooms must all have the same number of entities in order to allow users to be returned to the correct middle level room when leaving a reusable room. Reusable rooms must all be mapped identically to be able to convincingly pull of the illusion of many rooms. These factors make it very difficult to dynamically populate the rooms based on a detected network.

Room re-use also inherently limits the number of simultaneous administrators that can collaborate together to the number of "reusable" rooms. This is because if there are more administrators than there are reusable rooms, and they all try to inspect a different virtual room, there will be no reusable room available to virtualise the fifth virtual room and the player will be blocked from entering.

VII. A better solution

Room re-use as we have implemented was needed to get around certain limitations of the Quake III Arena engine. Maps must be completely defined at runtime, and the entire map is loaded before play begins.

Using an engine in which we could dynamically create new rooms and entities would alleviate a lot of the rigidness of the design, and allow rooms to more accurately represent the things they are supposed to (eg.

a dynamic number of entities in rooms depending on what they represent).

Also, our use of configstrings to get string data to clients has lead to problems. Because configstring data is not directly associated with any particular entity or area of a map, every configstring is sent to every client rather that just the ones pertaining to the visible entities, which causes the gamestate to grow and lead to possible fragmentation of gamestate packets.

Devising a way more efficient way to distribute string data to clients would also greatly help with our gamestate size problems. If we were able to have to strings stored within the `entitystate_t` structures, then the engine's networking would theoretically only tell clients about entities within their line of sight, allowing us to make a map equivalent to 'hierarchy-demo', without needing to resort to room re-use.

It should be noted that we did try storing strings in the `entitystate_t` structure in early prototypes, but ran into endian problems when messages went from a PowerPC based system to an Intel based one. Several sources had recommended the use of configstrings over trying to put strings in to `entitystate_t`, but we've since seen that they aren't really suitable such large amounts of data.

VIII. Conclusion

This technical report has described the hierarchy and room re-use features that exist in L3DGEWorld 2.3.

We have documented the method we have used to achieve hierarchical maps, explained why room re-use was required and given a description of how it was implemented.

We have covered how one might go about making a map that can work in a similar way to our 'hierarchy-demo' example map.

We have outlined the failings and disadvantages of our implementation and suggested a possibly better avenue for future hierarchical work.

Our example map that demonstrates these features, 'hierarchy-demo' is available as part of L3DGEWorld 2.3 [6].

IX. Acknowledgements

L3DGEWorld is being developed is under the direction of Grenville Armitage and Warren Harrop. This project has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley

References

[1] "The L3DGE Project," February 2008, http://caia.swin.edu.au/urp/l3dge.

[2] W.Harrop and G.Armitage, "Intuitive Real-Time Network Monitoring Using Visually Orthogonal 3D Metaphors," in *Australian Telecommunications Networks & Applications Conference 2004 (ATNAC2004)*, Sydney, Australia, December 2004.

[3] ——, "Real-Time Collaborative Network Monitoring and Control Using 3D Game Engines for Representation and Interaction," in *VizSEC'06 Workshop on Visualization for Computer Security*, Virginia, USA, October-November 2006.

[4] "L3DGEWorld 2.2 website," December 2007, http://caia.swin.edu.au/urp/l3dge/tools/l3dgeworld_2.2.

[5] L. Parry, "L3DGEWorld 2.3 Input & Output Specifications," CAIA Technical Report 080222C, February 2008, http://caia.swin.edu.au/reports/080222C/CAIA-TR-080222C.pdf.

[6] "L3DGEWorld 2.3 website," February 2008, http://caia.swin.edu.au/urp/l3dge/tools/l3dgeworld_2.3.