Level-of-Detail in Behaviour of Virtual Humans[*]

Ondřej Šerý, Tomáš Poch, Pavel Šafrata, and Cyril Brom

Charles University, Faculty of Mathematics and Physics,
Malostranské nám. 2/25, Prague, Czech Republic
{ondrej.sery, pavel.safrata}@seznam.cz, tom.poch@post.cz,
brom@ksvi.mff.cuni.cz

Abstract. An application featuring virtual humans is a program that simulates an artificial world inhabited by virtual people. Recently, only either small artificial worlds inhabited by a few complex virtual humans, or larger worlds with tens of humans, but performing only walking and crowding, are simulated. This is not surprising: a large world inhabited by complex virtual humans requires unreasonable amount of computational and memory resources. In this paper, we report on the project IVE, a common simulation framework for huge artificial worlds, pointing out the level-of-detail technique used at the behavioural level. The technique addresses the issue on reducing simulation demands by gradually decreasing simulation quality on unimportant places, while keeping the simulation plausible, with minimum scenic inconsistencies.

1 Introduction

Virtual humans are becoming increasingly popular both in the academic and industrial domains. Applications featuring virtual humans include computer games, virtual storytelling, movie industry, entertainment, military simulations, and behavioural modelling. An overview of domains of virtual humans is given for example in [11].

From the technical point of view, typically, each virtual human is viewed as an autonomous intelligent agent in the sense of Wooldridge [12]; such an agent that carries out a diverse set of goals in a highly dynamic, unpredictable environment with the objective to simulate behaviour of a human.

The research on virtual humans (v-humans in the following) is mostly focused around graphical embodiment and action selection mechanism. Generally, the former means a graphical visualization of a v-human's body, and the latter means deciding of what action to perform next in the virtual world. The main problem with the visualization is that v-humans must look believably. For example, it has been shown (*e.g.*, [9]) that emotional modelling plays a significant role in a posture and face visualization. The main problem with the action selection is that the environment is dynamic and unpredictable. A v-human must respond in a timely fashion to environmental changes that are beyond the v-human's control.

In this paper, we address a different issue. During our previous work on a toolkit ENTs for prototyping v-humans [1], we discovered that it was not a problem to

[*] This work was partially supported by the Czech Academy of Sciences project 1ET400300504.

develop a single v-human with meaningful and believable behaviour. The problem was to populate a large artificial world with tens of v-humans running on a single PC, because of the limited computational and memory resources. Most of the current v-humans either "live" in a small artificial world (*e.g.*, in a room, not in a village), or do exhibit only a small portion of human-like behaviour (*e.g.*, only object-grasping, walking, or a few tasks, not weekly human activities). An application or a technique that would challenge large simulations is missing. Such a technique would be extremely useful in the fields of computer games, and virtual storytelling.

At the time, we are working on a project IVE (*an intelligent virtual environment*) [2], [7], which is focused on v-humans in large and extensible simulations. One of the goals of the project is to explore and implement the *level-of-detail technique* (LOD). This technique is widely used in computer graphics for reducing computational cost. Our aim is to use it at the behavioural level; it means to transfer it to the domain of artificial intelligence.

The LOD technique for behaviour of v-humans is based on the simple idea: there often exist only few places in the artificial world important at a given simulation time and the unimportant places do not need to be simulated precisely. If the artificial world is simulated only partially, the demands of the simulation can be reduced significantly. However, there are three problems coming out with the implementation: 1) how to identify the important places, 2) how to simplify the simulation in the unimportant places, 3) how to gradually simplify the simulation between an important and an unimportant place?

In this paper, we present our approach to LOD technique at the behavioural level. We are motivated by the growing need of large simulations with v-humans in the domains of computer games, and virtual storytelling. The goal is to address the three aforementioned problems. The algorithms presented here are already implemented in the on-going project IVE.

The rest of the paper proceeds as follows: First, we describe related work on the LOD technique in artificial simulations. Second, we briefly present our framework and its view of the artificial world. Then, in Section 4, we present main concepts of the simulation LOD followed by a brief description of our implementation, in Section 5. Finally, we conclude in Section 6.

2 Related Work

The LOD technique is widely used in computer graphics, but not often in behavioural simulations. The idea behind is simple: compute only such details that are important at a given simulation time. At the behavioural level, that means places observed by the user, and other places important for the overall course of the simulation.

Sometimes, this idea is exploited in computer games, but only to a limited degree. Behaviour of the creatures out of the sight of the user is not simulated at all typically. This often causes a storyline inconsistency - the simulation is not believable. Instead of "non-simulation", there is a need for a gradual simulation simplifying.

More robust idea how to use the LOD at the behavioural level using hierarchical finite state machines is presented in [3], but it is only a sketch not further explored.

Sullivan *et al.* utilised the LOD for conversational behaviour by means of rules and roles filtering [10]. In a simplified fashion, the rules and roles can be viewed as pieces of a code layered upon a basic v-human. If the v-human is not seen, the role is not passed to it, and consequently only the basic behaviour is performed. Contrary to our approach, they simplify only the behaviour of v-humans, not the overall simulation.

A robust approach to (non-pre-emptive) scheduling of processor time to individual v-humans is presented in [13]. However, as the not-scheduled behavioural scripts are not run, this approach seems to fit into the realm of “yes/no simulation”.

3 Project IVE

Let us first describe our framework and our view of the artificial world. In our framework we distinguish between *objects*, *actors* and *processes*. All physical objects in the artificial world are objects. Special objects that can manipulate with other objects are called actors. The only way how to affect objects is to perform a process.

In our framework, the world consists of *locations* on which objects can be located. Locations are organised in a hierarchical structure related to the LOD technique. The structure is always a tree and levels of the tree correspond to the LOD levels. *LOD value* of an object is defined as a corresponding LOD level of the location on which the object is situated.

Objects presented in our framework are *smart* (in the sense of [8]), which helps with world's extensibility. They contain necessary graphical information and description of low-level actions (*e.g.,* grasping the object). However, they do not contain the artificial intelligence itself. Our objects also provide *affordances* [5] - each object is able to give a list of processes it is designed to participate on.

In our framework, processes can be also labelled as smart. Each process has a number of sources on which it operates. When executing a process, objects are substituted as these sources. Some of the sources have a special actor position. From the view of the process, the position of actors differs from the other sources especially in the connection to the LOD, as we shall see later. Our processes have also an ability of *suitabilities* - they can say how much are the given objects suitable for being substituted to the process. In other words, each process can say, if it is a “good idea” to be executed with some particular objects as sources or not.

Processes in our framework are organised in a hierarchical structure tightly connected with the LOD technique, as well. Each process can be performed atomically, or expanded to subprocesses. The structure is not as strict as in the case of the locations—process can be atomic at more LOD levels. Each executed process can stay in one of these states:

– *Not-existing* – the LOD value is too low, a super process is running. This process does really not exist in the world. If the process is running and the LOD value goes too low, it is stopped, partially evaluated and discarded.
– *Atomic* – the process is running atomically. It waits till its finish time and then changes the world’s state.
– *Expanded* – the process is expanded to subprocesses. Such process performs no action, it only waits for it’s subprocesses to do all the work. However, it can become atomic as a consequence of the LOD changes.

To define the process' state, we need for each process two border LOD levels - *minimum* level (border between not-existing and atomic process) and *maximum* level (border between the atomic and the expanded process). The state of the particular process is then determined by the LOD values of its actors in relation to the border levels. That means, the process is atomic (the only process' state in which the world's state is influenced) if the actor's LOD is between the process' minimum and maximum levels, similarly for the other states. The process can have any number of actors, but their LOD values must not require different states of the process (they all must have the LOD value at the same side of the border levels). It is up to the framework to adjust LOD values in the corresponding locations.

Hierarchical if-then rules are used to describe the processes. However, the subprocesses of an expanding process are not hardwired. The action selection is driven by a goal concept [4, 12]. The actor has a goal (an intention to reach some objective) and can try various processes to satisfy the goal. Processes also do not expand directly to subprocesses, but rather to the subgoals (see Fig. 1). In this concept, the actor obtains the list of goals needed to satisfy the parent process and its task is to find and perform appropriate processes.

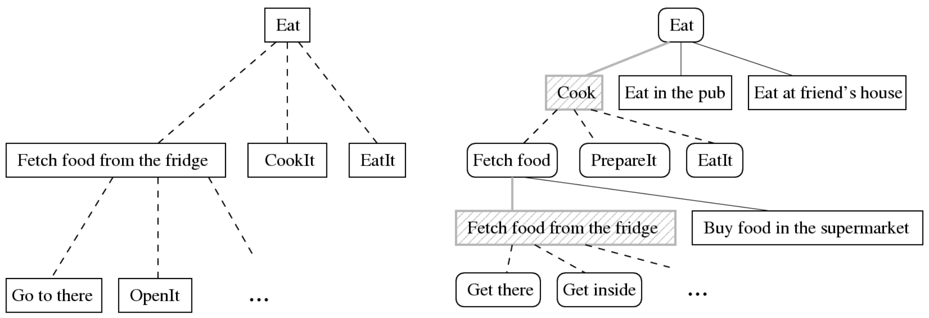

Fig. 1. Left – process hierarchy without goals. Right – goal-process hierarchy, where the process acts as an implementation of a goal.

The actual artificial intelligence is not encapsulated in the actor's object. This is the purpose of a presence of entities called *geniuses*. Genius is the one who chooses the processes, looks for the suitable sources and asks the framework to perform the chosen processes. An actor can have its own genius, which is actually his brain. But in addition to that, dedicated geniuses are present in our framework. These geniuses are specialized to particular activities and are able to control actors passed by another geniuses. This concept allows for example creation of dummy actors, which are driven by geniuses of locations as they travel within the world, or geniuses with ability to perform some non-trivial interaction among more actors. For example playing cards in a pub could be easily driven by a single specialized genius, while controlling such an action from more individual geniuses would be a tremendous task.

4 Simulation LOD

In the previous section we have introduced our view of the artificial world and both the location and process hierarchy. Now let us take a closer look at the simulation LOD. In this section, we shall introduce few rules that control location hierarchy expansion and then, in the next section, we shall describe implementation of the component that enforces their abidance.

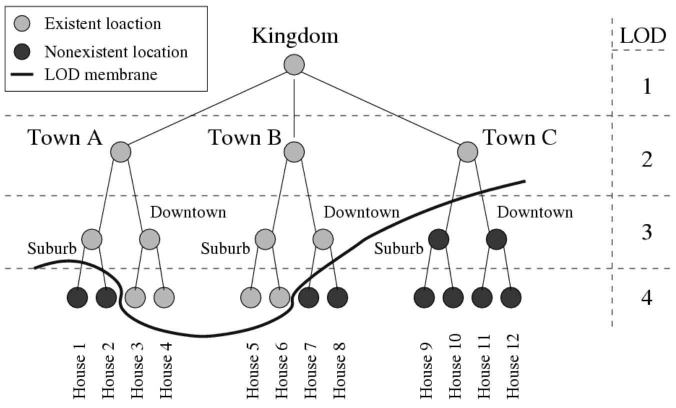

Fig. 2. The simulation LOD can be viewed as an elastic membrane cutting thought the location hierarchy. If no other force exists, the membrane presses LOD to low values (fewer details).

The best way to imagine the simulation LOD is an elastic membrane cutting through the location hierarchy (see Fig. 2). Only the locations above this membrane do currently exist and so only these locations are simulated. Objects are located only in leaves of the clipped hierarchy tree and their LOD values are equal to the LOD value of these leaf locations, as described in the previous section.

The base framework aims to keep details low (the membrane presses upward) in order to simplify (and thus speed-up) the simulation. On the contrary, simulated objects press the membrane down to ensure enough details necessary for their own simulation.

Each object has two values: the *existence* level and the *view* level. The existence level marks the border LOD value below which the object is not simulated. The view level is a LOD value which is enforced by the object if current LOD value is greater or equal to existence level (see Fig. 3). Situation, in which the actual LOD would be between the view and existence level, is considered invalid and our framework either expands or shrinks all such locations to adjust LOD value out of this interval.

All objects in given location and their existence and view levels define possible LOD values that would not result in the invalid state (invalid states forms invalid areas on the Fig. 3). This is the basic rule that our framework must obey and that can be violated whenever an object changes its location.

Typical use for an important object (such as the user's avatar) is low existence level and high view level which enforces high LOD values in the location where the object stays. On the other hand, unimportant objects would have the existence level close to the view level and both quite high. This does not enforce nearly any changes in LOD (corresponding invalid area is small) but rather only specifies whether such an object exists or not.

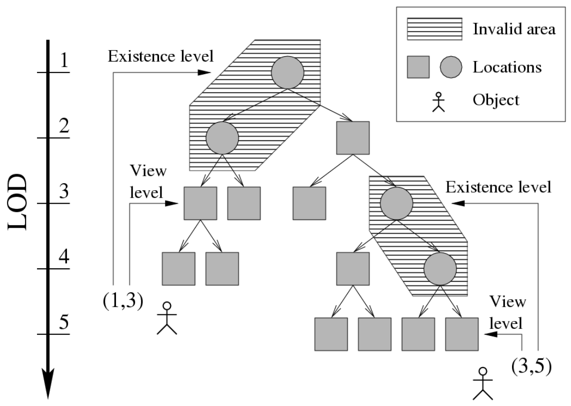

Fig. 3. Valid and invalid LOD values based on the existence and view levels of more objects

Unfortunately, this is not enough to ensure fluent simulation. Problem occurs when an important object frequently moves between two locations from different branches of the location hierarchy. This could cause many expansions and shrinks constantly which is definitely not acceptable. We would prefer a LOD membrane to create a 'crater' rather than a narrow 'hole' around the important objects. This would make neighbouring locations to prepare for a visit of the VIP object before it reaches their borders by gradually increasing their LOD value (expanding to sublocations).

This idea is handled by adding *LOD influence* between locations. It is a relation which marks nearby locations by a certain number. This number defines how much can the LOD values differ between these two locations.

These rules, when enforced, answer the three basic questions from Section 1. Important places are identified as places containing objects with low existence level and high view level. Even more, these levels can also say how much important the objects are. Answer to the second and the third question is inherent in the fact that the whole world is hierarchical with each hierarchy to some extend corresponding to the LOD levels. So we can easily simulate on different levels. Gradual LOD changes are assured by the use of LOD influence between neighbouring locations.

5 Implementation

In this section, we describe a component called LOD Manager which enforces abidance of the rules mentioned in the previous section.

LOD Manager aims to expand and shrink locations to achieve valid LOD values with regard to objects' view and existence levels and LOD influence. On the other hand, it keeps LOD value as low as possible to assure a fast simulation.

Another requirement, that LOD manager must comply, is to avoid ineffective expands and shrinks when an object constantly moves around the border between two locations. Such situation would happen with a naïve solution that would shrink locations as soon as their existence was not strictly enforced. Such an object could cause unnecessary load on the system. We could solve this requirement by defining a second LOD crater with the same centre and greater radius. The inner crater would affect expansions and the outer one shrinking. This approach would need additional information describing outer crater in the world description, so we decided to implement another solution. It resembles the garbage collection technique known from programming environments.

LOD manager is gathering information about objects' position and movement from the framework by a simple interface containing methods addObject(object, location), removeObject(object, location) and moveObject(object, oldlocation, newlocation). Beside these methods, LOD Manager can be asked to remove locations that are not needed—to push the LOD membrane upwards. It is up to the framework to decide when to invoke cleanup()—time by time or when the simulation goes too slow.

LOD manager changes the state of the location tree by invoking expand() and shrink() methods on particular locations. During execution of expand() method, the location generates a net of its sublocations and objects specific for this location (*e.g.*, flowers in the garden). All objects fall down to the new sublocations. Also an option of object's expansion to subobjects is plausible but we have not implemented it yet. Method shrink() makes the target location atomic. Its location subtree is forgotten. Objects form the subtree are either placed on the shrunk location (if their existence level is lower than its LOD value) or cease to exist (these are typically specific for the given location and can be generated again during the expand() call).

We say that the location holds the *basic condition* if there is no object placed in its subtree such that LOD value of the location would be between the object's existence (included) and view (excluded) level. Such a location could easily be atomic without causing an invalid state (see Section 3) and so the shrink() method can be called on it. However, we could still violate influences between locations and lose the 'crater' optimalization. Fig. 3 shows 11 locations that hold the basic condition as squares and 4 that do not as circles.

During the simulation, LOD Manager is accepting notifications about objects' movement (via methods mentioned above). It keeps notion about locations that hold the basic condition and also controls LOD influence. Thus it can dynamically detect an invalid state or violated influence. In both cases, it reacts by calling expand() on offending locations.

This way, locations can get only expanded. The shrink() method is not called dynamically but only during the cleanup() method execution. In this method, LOD Manager finds a cut through the location hierarchy that is closest to the root while still acceptable with respect to the basic condition and LOD influence.

For this purpose, we use the depth first search (DFS) algorithm. If a location is intended to be shrunk it is marked. In the instant moment, if the location is marked, no of its descendants or ancestors can be marked. We can see three types of edges in the graph of currently existing locations:

- *Disabled* – heads from the location that holds the basic condition to each its child. This edge cannot be used during the traversal (unless shortcut by an influence edge).
- *Enabled* – other edges that head from the parent to its child. These can be freely used during the traversal.
- *Influence edge* – edges that correspond to the LOD influence. These are used too, but it could happen that an ancestor of influence target is already marked. In such a case the ancestor is unmarked and all its children are traversed.

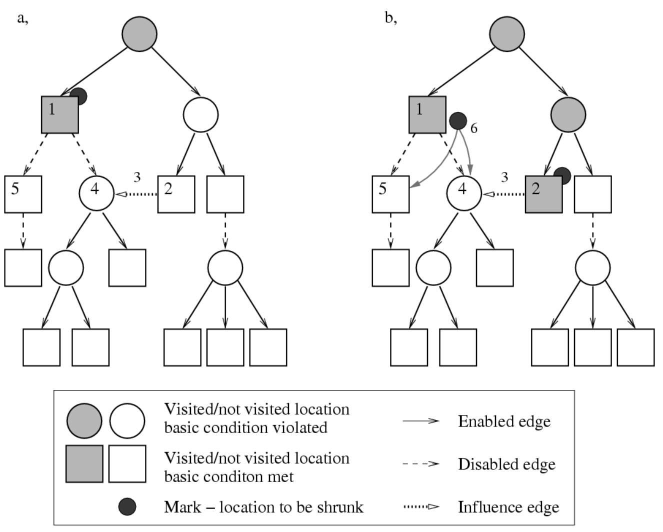
a,
b,
1
2
3
4
5
6
Visited/not visited location basic condition violated
Visited/not visited location basic conditon met
Mark – location to be shrunk
Enabled edge
Disabled edge
Influence edge

Fig. 4. On the picture a, DFS algorithm have visited location 1 and have found no enabled edge to continue. So location 1 is marked and DFS continues by traversing the second child of the root location. This is shown on the picture b. On this picture DFS have found the influence edge 3 at location 2. Because there is no enabled edge going from location 2, it is marked. DFS then continues by traversing the influence edge to location 4. In this case there is a marked ancestor of influence target. So we must remove its mark 6 and traverse all its children (location 4 and 5). In this case the influence value is zero - general case is slightly different.

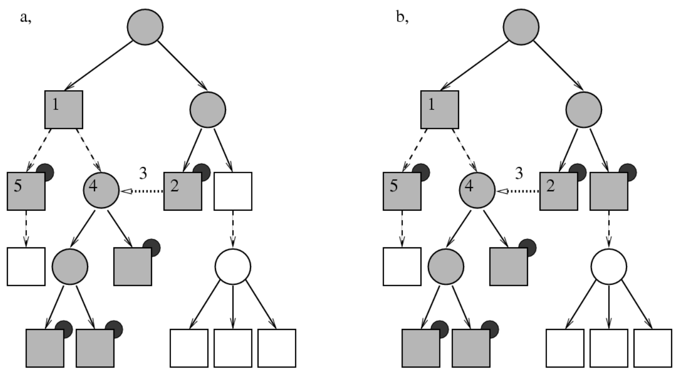

Fig. 5. a, DFS have finished traversal caused by the influence edge. Locations in the subtree of location 1 are marked properly. Picture b, shows state of the location tree after whole DFS traversal. All marked locations will become atomic.

Location is marked if there is no usable edge heading to any of its children. After completion of the DFS traversal, method shrink() is called on every marked location. This makes all marked locations atomic. See Algorithm 1, 2 and Fig 4, 5 for further details.

Algorithm 1: `Cleanup`

```
markRecursive(rootLocation);
foreach (loc : marked locations)
   loc.shrink();
```

Algorithm 2: `MarkRecursive(loc)`

```
if (location loc was already traversed)
   return;
if (no enabled edge from the location loc)
   mark(loc);
else
   foreach (target : targets of enabled edges)
      markRecursive(target);
foreach (edge : influence edges from location loc) {
   target = edge.target;
   while (loc.LODvalue - target.LODvalue < edge.value)
      target = target.parent;
   while (there is a marked ancestor of target) {
      ancestor = the marked ancestor of target;
      unmark(ancestor);
      foreach (child : children of ancestor)
         markRecursive(child);
   }
}
```

6 Conclusion

Simulation of large artificial worlds with tens of v-humans with complex behaviour is a task requiring a special technique that can cope with limited computational and memory resources. In this paper, we have presented our approach to this issue. The approach is based on level-of-detail technique (LOD) that decreases simulation quality at unimportant places. Contrary to the common use of LOD in computer graphics, we have used it in the domain of artificial intelligence. Contrary to a few exploitations of LOD in the domain of computer games [3], [10], [13], our approach is robust. That means it simplifies the quality of simulation gradually, and it simplifies not only behaviour of v-humans, but also an underlying topology of the artificial world. The contribution is obvious: owing to the smoothness, we achieve better believability while preserving reasonable computational and memory demands.

The technique is used in project IVE [2], [7]. The project itself is still in progress.

References

1. Bojar, O., Brom, C., Hladík, M., and Toman, V.: The Project ENTs: Towards Modelling Human-like Artificial Agents. In SOFSEM 2005 Communications, Liptovský Ján, Slovak Republic (2005) 111-122
2. Brom, C.: Virtual Humans: How to Represent Their Knowledge. In: Proceedings of ITAT 2005, Slovak Republic (to appear) (in Czech)

3. Champandard, A.J.: AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders, USA (2003)
4. Georgeff, M., and Lansky A.L.: Reactive Reasoning and Planning. In: Proceedings of the 6th National Conference on Artificial Intelligence, Seattle, Washington (1987) 677-682
5. Gibson, J.J.: The Ecological Approach to Visual Perception. Boston: Houghton Muffin, (1979)
6. Huber, M.J.: JAM: A BDI-Theoretic Mobile Agent Architecture. In: Proceedings of the 3rd International Conference on Autonomous Agents (Agents'99). Seatle (1999) 236-243
7. IVE, the project homepage: http://mff.modry.cz/ive/
8. Kallmann, M., and Thalmann, D.: Modelling Objects for Interaction Tasks. In: Proceedings of EGCAS 98 (1998) 73-86
9. Mateas, M.: Interactive Drama, Art and Artificial Intelligence. Ph.D. Dissertation. Department of Computer Science, Carnegie Mellon University (2002)
10. O'Sullivan, C., Cassell, J., Vilhjálmsson, H., Dingliana, J., Dobbyn, S., McNamee, B., Peters, C., and Giang, T.: Level of Detail for Crowds and Groups. In: Computer Graphics Forum **21** 4 (2002) 733-742
11. Prendinger, H., Ishizuka, M.: Introducing the Cast for Social Computing: Life-Like Characters. In: Predigner, H., Ishizuka, M. (eds.): Life-like Characters. Tools, Affective Functions and Applications, Cognitive Technologies Series, Springer, Berlin (2004) 3-16
12. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons (2002)
13. Wright, I., and Marschall, J.: More AI in Less Processor Time: 'Egocentric' AI. In: Gamasutra on-line (2000) http://www.gamasutra.com/features/20000619/wright_01.htm