VIRTAINER: GRAPHICAL SIMULATION OF CONTAINER STORAGE YARD

Miguel Escrivá
Marcos Martí
José Manuel Sánchez
Emilio Camahort
Javier Lluch
Roberto Vivó

Computer Graphics Section
Department of Computer Science
Polytechnic University of Valencia, Spain
E-mail:{mescriva|mmarti|josanchez|jlluch|camahort|rvivo}@dsic.upv.es

KEYWORDS

Stacked object, container block, scene graph, cull, vertex array, device tracking.

ABSTRACT

Currently, 3d graphics are being applied to an increasingly wide range of applications. Industrial processes monitoring is an area where this type of simulation is rarely used, being much more common to represent the information in traditional 2d interfaces.

We introduce here an application of data visualization for maritime container terminals. This system is modular and adaptable to any stacked objects problem. It is architected on a real data base and on the operation processes of a logistic industry. In this paper we describe the architecture of this system, as well as the graphic acceleration techniques that allow us to maintain the frame rate independently of the data size.

INTRODUCTION

Stacked objects are very common in the industry, especially in logistic enterprises. Maritime container terminals are a particular case of this statement. The management of this processes information is usually done by systems with poor alphabetical interfaces which lack the ability to adapt to changes. Advanced graphical simulations model the real process with high fidelity and show changes in a much more convenient manner.

This kind of applications require the access to the system information, usually stored in a database, and the ability to receive the changes occurred in that information through a communications layer. We will show the problems found in the non-graphic parts of these systems, and the approaches we have taken to solve them.

Large data-sets, characteristic of these industries, pose a challenge to the development of graphical simulation systems. The techniques we describe here solve two main problems: real-time visualization of very large data-sets and the modification of this data with an event oriented model. Visibility culling techniques (Cohen-Or et al., 2001), such as geometric simplification, occlusion (El-Sana, 1997) or impostors, have been developed to reduce the amount of data employed to render the scene while maintaining the image quality.

Another indispensable part in any Virtual Reality application is the tracking of input devices. We have implemented a tracking application, based on *OpenTracker* (Reitmayr and Schmalstieg, 2001), which can be configured using standard XML files.

Existing commercial applications, such as those described in (Gambardella and Rizzoli, 2000), offer text or 2d graphics interfaces, and are unable to respond to real time events. Our system offers a richer interface in a virtual representation of the maritime terminal. It is a multi-platform system with integrated data services, XML configuration files, standard database access, and a geometric layer based on an open source scene graph engine. It integrates several graphic accelerations which exploit the spacial coherence of stacked objects to speed up the rendering system. Virtual reality is completely supported, as our system works with any configuration of processing machines, screens and devices. Frame synchronization is done at swap buffers level, allowing the simulation to run simultaneously in any number of computers.

The structure of this paper is described now. First we begin explaining the antecedents and main goals of the project. Then we give some details about the graphic acceleration techniques we have used. A description of devices and interaction in *Virtainer* follows. The next section details *Virtainer's* architecture, its constituting modules and how they work. Finally we present some conclusions and the main directions for future works.

BACKGROUND

This project was conceived as an extension of a previous 2d monitoring application, *Visual Gama* (Jorquera et al., 2003), adopted in the enterprise Marítima Valenciana, one of the most important container terminals of the Mediterranean sea. We felt that the 2d interface wasn't rich enough, and we thought that the natural extension was adding real-time 3d graphics to create a more realistic representation of the terminal with smooth animations to offer higher fidelity and more up-to-date information.

Virtainer is built on top of completely standard open source projects which deliver compatibility and great performance. *OpenSceneGraph* (Burns and Ostfield, 2003) is employed as our graphics engine, providing Virtainer with object management, standard devices handling and other facilities. *OpenProducer* gives us a rendering context and camera control, and *OpenThreads* (Haines and Lagendoen, 1997) provides thread management. The communication layer can employ any protocol, but we are currently using *SCore* (Poza et al., 2002; Posadas et al., 2002), because it is the system adopted in the container terminal to send and receive event messages. Finally, *OpenTracker* encapsulates the devices events in a message which is received and parsed by *TrackerManipulator*. In *Gama* we dispose of data access and communication layers, so *Virtainer* is on the top of *Gama* and fully integrated with it.

The goals of our system are real-time navigation with sustained frame-rate, realism and virtual reality support. An event oriented system is required to maintain the real state of the yard, and multi-view and multimode inspection (fly, walkthrough, cab views, etc) are also desired. The system should offer interaction facilities for selection and camera control.

ARCHITECTURE

Usually container terminals store their information system in a centralized server, using different technologies like ODBC, OLE/DB or web services based on SOAP or XML-RPC. This database stores at any moment the state of the terminal, so the only information needed by our system resides here. Containers, trucks and cranes information is stored in a certain relational scheme that is accessed to represent these elements graphically. It is generally possible to associate a trigger to each interesting event of the database. When any change in the information system occurs, this trigger can broadcast the modification to any application or software agent.

Virtainer is a system based on standard C++ libraries and constructed over open source standard projects. It consists of a set of seven modules, implemented as dynamic libraries, each one handling a different aspect of the simulation.

vrtUtil provides logging facilities and a connection to the Xerces-XML library used to parse configuration files. *vrtDB* encapsulates the code required to connect to a database and load information from its relational schemes. All the logical information of the terminal is stored in *vrtData*. Up-to-date logical positions, destination, vendor and other container data are maintained here. The same applies to trucks, cranes and other machinery. *vrtGeo* manages the geometric aspect of the simulation. It loads the machine models, handles its animations, and is responsible of the creation, destruction and manipulation of the containers geometry. The communication layer resides in *vrtComm*, which is the module employed to receive terminal events and to interchange synchronization messages between the elements of a cluster in a mutiple machines configuration. *vrtGUIHandler*. Keyboard, mouse events and related aspects are managed here, including key bindings and objects selection. *virtainer* serves as an interface to the library set, providing methods to initiate and finish the execution of the library modules, as well as a Viewer class to control the visualization.

When the application begins, the data is loaded from the server database using the *vrtDB* module. This information flows to *vrtData*, and then to *vrtGeo*, where it is used to create the graphic simulation. The changes in this information are received via *vrtComm*, are passed to *vrtData*, and then to *vrtGeo*, where the simulation is updated according to the event occurred. Figure 1 shows this scheme.

Each module has one or more associated configuration files in standard XML format where the relevant parameters for that module can be specified.

GRAPHIC TECHNIQUES

Frustum culling, explained in (Hadwiger and Varga, 1997), is the most common method of culling used in computer graphics. Objects outside the camera frustum are removed from the rendering pipeline. The scene graph structure makes this technique very efficient, as complete object groups can be removed with a simple comparison between the frustum and the bounding volume of the group. *OpenSceneGraph* employs two object lists populated by the culling process. The

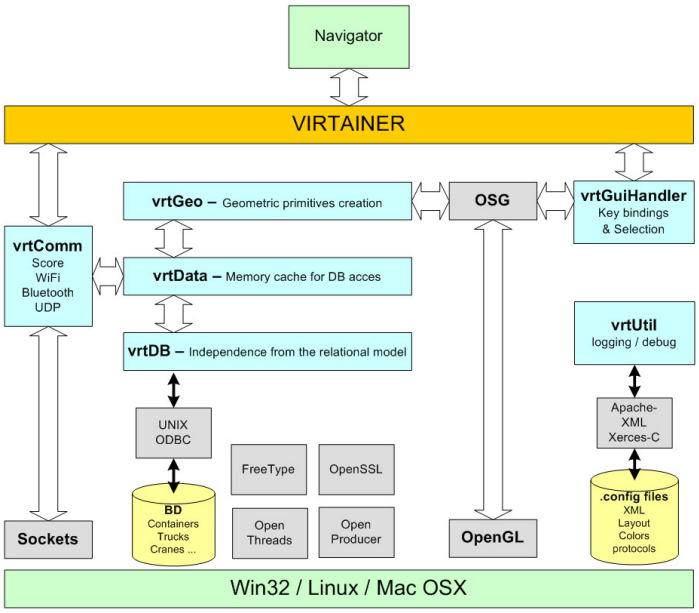

Figure 1: Virtainer modular structure

first list stores the opaque objects ordered by state (material, texture, vertex programs, etc) and the second the transparent objects, ordered by depth. Different bounding volumes are used to improve performance: boxes for geometry and spheres for object groups.

Multi-resolution objects are the next technique implemented. Objects in *Virtainer* have static pre-computed levels of detail, to reduce the number of polygons rendered without reducing noticeably the quality of the resulting images.

Impostors, as explained in (Schaufler, 1995), are a technique which consists of rendering to a texture the objects which are far from the observer, and substituting the real objects with billboards containing that texture. As long as the camera does not move we can achieve very high frame-rates. We have also developed a specific method to create impostors for the container blocks, which consists of using the faces of the bounding box of each individual block as pre-rendered impostors. This method is used when the container block is far from the observer, reducing the geometry of a whole container block to only five quads.

Non-visible geometric culling is other method applied. Container faces which are not visible from any view are removed from the scene graph before visualization. This technique consists of rendering only the surface of each container block, and leads to a noteworthy reduction of the polygon number and an increase of the frame-rate.

Using *Back-face culling* (Kumar et al., 1996; Hultquist, 1990) we can avoid rendering the back parts of the containers. It is easy to see that of the six faces of a container we can only see three at the same time, so with this technique we can reduce the number of polygons to be rendered in a 50%.

Our system can use portal rendering (Elmqvist, 1999a,b) to accelerate the visualization in walkthrough mode. When the application starts, the information about containers is loaded from the database, and it is then used to create cells and portals automatically following the *Breaking the walls* (Lerner et al., 2003) algorithm. Our method, based on (Luebke and Georges, 1995), introduces portal reconstruction to adapt to the dynamic geometry of the container terminal.

Vertex arrays is the last acceleration technique used in *Virtainer*, and consists of grouping the geometry of the container blocks in a certain way which is better handled by the graphics card.

Multiple machines using *Virtainer* can be interconnected to increase the computer power when rendering to more than one display device. We have chosen a master-slave architecture where the master computer controls the camera and sends the model view matrix to all the slaves. Then the camera can be modified to adapt the view to the physical display configuration. Our election of this architecture does not mean we are limited to connecting the devices directly to the master: any machine can receive this kind of events and send them to the master, which will then distribute the changes through the cluster.

INTERACTION

We have developed a utility to manage input devices which offers support for any controller supported by the operating

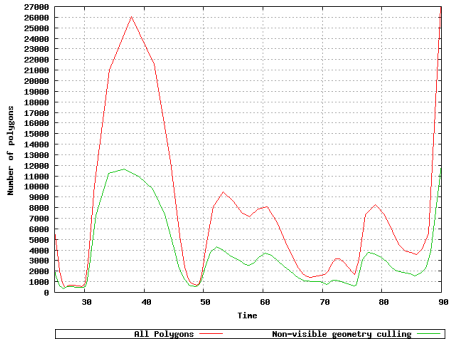

Figure 2: Number of polygons rendered using or not Non-visible Geometry Culling

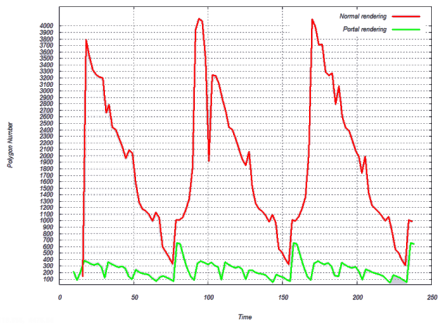

Figure 3: Number of polygons rendered with and without Portal Rendering

system. It is based on *OpenTracker*, benefiting of its advantages: configuration based on XML files, device abstraction and transparent network access. Our tracking system can be launched in a different host that the main application, allowing the device management to run in a dedicated host. Finally, we use XML configuration files to map tracking data to movements and select the interaction type.

RESULTS

We present here the impact in the frame-rate of the graphic techniques implemented. Removing non-visible container faces deliver a noteworthy Figure 2, but the frame-rate is still low; some more improvement is required.

We have reduced considerably the number of polygons using portals, as can bee seen in figure 3. This results in approximately a 75% increase in the frame-rate.

Finally, as you can see in Figure 4, employing vertex array lists delivers a great increase of the frame-rate. Modern graphics cards are capable of dealing with a very large number of polygons, provided that they are organized in a compact structure.

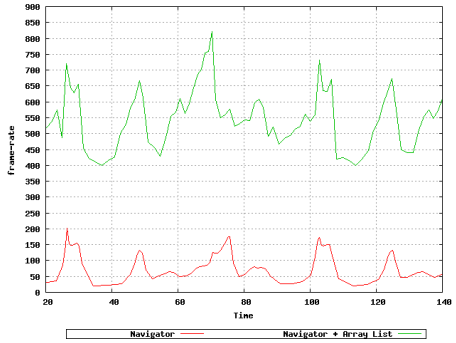

Figure 4: Frame-rate comparison using or not Vertex Arrays

Figure 5: Virtainer running in a cockpit view configuration

We have used joysticks, game-pads and a wheel to control *Virtainer*, and we have also a tracker which follows the user head movements, enabling the user to look around the terminal. The screen configurations tested are jumbo-box, a cockpit view with three monitors and a head mounted display. An example can be seen in figure 5.

Movements of the terminal machinery are reflected by generating smooth animations of the models. When a machine starts to move, an event message is received, and its representation reacts accordingly. All the machines send a message with its position regularly, thus the error of the positions is minimal.

CONCLUSION AND FUTURE WORK

We have presented here a multi-platform simulation system capable of representing a maritime container terminal in 3d with a sustained frame-rate. It offers support for virtual reality configurations and any device can be used to control the navigation. It is possible to select relevant elements of the terminal to show related information. We have created a web site where we publish project related information. The address is http://www.virtainer.org.

We are currently working in improving the quality of the image even more using shader based techniques, like those in

Figure 6: New water shader and Displacement Mapping in containers

(Dumont et al., 2003; Wang et al., 2003), and in designing an event simulator to produce animations outside the maritime terminal network. Figure 6 shows some of these improvements.

Other interesting areas are the improvement of the portal culling and the use of *occluders* (Coorg and Teller, 1997) for systems with lower specifications or even larger scenes. There is also some room for improvement in the synchronization method used; a peer-to-peer scheme may be better suited for our system than the current master-slave configuration.

ACKNOWLEDGMENTS

The *Virtainer* project is being supported by grant TIC2002-04166-C03-01 from the Spanish Ministry of Science and Technology and IST-2-004363-STP from the GameTools project. We thank *Virtainer*, *Gama* teams for their work and collaboration. We are also grateful to Marítima Valenciana for their support of our research efforts.

REFERENCES

Burns, D. and R. Ostfield. Open Scene Graph - An Open Source Solution for Real Time Image Generation. 2003. IMAGE Society.

Cohen-Or, D., Y. Chrysanthou and C. Silva. A survey of visibility for walkthrough applications. 2001. IEEE TVCG.

Coorg, S. and S. Teller. Real-time Occlusion Culling for Models with Large Occluders. 1997. IEEE TVCG.

Dumont, R., F. Pelacini and J. A. Ferwerda. Perceptually-Driven Decision Theory for Interactive Realistic Rendering. 2003. In *ACM Transactions on Graphics*.

El-Sana, J. Integrating Occlusion Culling with View-Dependent Rendering. 1997. IEEE TVCG.

Elmqvist, N. Axis-Aligned Bounding Box Culling For Portal Rendering. 1999a.

Elmqvist, N. Introduction to Portal Rendering. 1999b .

Gambardella, L. M. and A. E. Rizzoli. The role of simulation and optimisation in intermodal container terminals. 2000. European Simulation Symposium.

Hadwiger, M. and A. Varga. Visibility culling. 1997. The Institute of Computer Graphics and Algorithms, Vienna University of Technology.

Haines, M. and K. Lagendoen. Platform-independent runtime optimizations using OpenThreads. 1997. In *11th International Parallel Processing Symposium*.

Hultquist, J. Backface culling. 1990. In *Graphics Gems*, 346–347.

Jorquera, P., T. Barella, D. Llobregat and R. Vivó. Visual GAMA. 2003. EUROGRAPHICS.

Kumar, S., D. Manocha, W. Garrett and M. Lin. Hierarchical back-face computation. 1996. In *Eurographics Rendering Workshop*, 235–244.

Lerner, A., Y. Chrysanthou and D. Cohen-Or. Breaking the Walls: Scene Partitioning and Portal Creation. 2003. In *Pacific Graphics 2003*.

Luebke, D. P. and C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. 1995. In *Proceedings 1995 Symposium on Interactive Computer Graphics*, 105–106, Department of Computer Science, University of North Carolina.

Posadas, J.L., P. Pérez, J.E. Simó, G. Benet and F. Blanes. Communications structure for sensory data in mobile robots. 2002. *Engineering Applications of Artificial Intelligence* 15, 341–350.

Poza, J.L., J.L. Posadas, J.E.Simó and A.Crespo. Data and Event Management in a Maritime Terminal of Containers. 2002. *New Technologies For Computer Control* 269 – 274.

Reitmayr, G. and D. Schmalstieg. An Open Software Architecture for Virtual Reality Interaction. 2001. ACM.

Schaufler, G. Dynamically generated impostors. 1995. In *D. W. Fellner, editor, Modeling Virtual Worlds - Distributed Graphics*, 129–136.

Wang, L., X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H. Shum. View-Dependent Displacement Mapping. 2003. In *ACM Transactions on graphics, 22*.