Managing Deformable Objects in Cluster Rendering

Thomas Convard, Patrick Bourdot, and Jean-Marc Vézien

LIMSI-CNRS, Université Paris XI
{convard, bourdot, vezien}@limsi.fr

Abstract. This paper presents DRS (Distributed Rendering System), a software library for rendering dynamic 3D scenes on low cost Virtual Reality (VR) computers. This library allows the distribution of object descriptions and geometries over several graphics nodes of a PC cluster for synchronized rendering. The objects can be dynamic: in particular DRS can manage deformable objects over time. Data compression is put to profit to optimize transfer of objects geometry over the cluster nodes. This library allows real-time VR rendering on PC clusters for applications where objects are highly deformable, such as solid modeling or simulation. DRS is available as a set of C++ classes to manage the 3D objects as well as their synchronized distribution and rendering. We present experimental results in the form of examples of applications successfully ported to inexpensive distributed Virtual Reality hardware thanks to DRS.

1 Introduction

Thanks to the development of graphic accelerators in recent years, PC platforms have become powerful enough to be considered for Virtual Reality (VR) demanding applications. However, at the present time, due to some technical limitations, one single PC cannot support a full immersive environment with several stereoscopic projections such as CAVE [1] or Workbench [2]. Recent hardware solutions, such as NVidia® SLI™ or Alienware® Video Array™, allow multiple graphic boards in a single PC. But these solutions are designed to improve the rendering on a single display and are not yet validated for VR use. Therefore, at the present time, PCs have to be clustered to support VR multiple displays. PC clusters provide a low cost visualization solution, but the software development on this kind of platform is much more challenging than on high-end (thus expensive) shared memory systems commonly used for VR.

DRS is a simple library designed to distribute a graphic objects database over a PC cluster, with a special emphasis on the management of deformable objects (i.e. objects whose geometry is changing over time). Managing such objects is required to support specific applications in VR, such as 3D modeling, where the geometry of objects is ceaselessly changed by the user. The dynamic definition of objects can represent a huge amount of data that must be distributed across

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 290–297, 2005.

the rendering PC nodes in order to maintain a coherent view of the VR scene on the different displays. Thus the problem is to optimize bandwidth of the network transmitting the data.

The objectives behind the design of DRS were:

1. Simplicity: DRS was designed to integrate well with other VR toolkits. Thus DRS is a software library designed to distribute rendering and perform management of 3D objects databases: it is not a VR toolkit managing devices or performing application control. Most of other scene graph distribution libraries, for instance Syzygy [3] or Avango [4] require developpers to build applications using dedicated frameworks.
2. Genericity: the aim is to support a wide range of VR applications. The primary targets of the present study were an immersive CAD demonstrator and a bioinformatics VR software previously running on a high-end visualization system.
3. Performance when dealing with deformable objects: while most available scene graphs managers handle the dynamics of scene nodes appropriately, the distribution solutions are not optimized for deformable objects.

2 Related Work

There are three main approachs in the literature for cluster graphics, each corresponding to a level of distribution: the application level, the scene graph level and the graphical commands level.

The simplest option is to replicate the VR application on each node of the cluster (Figure 2). This is the approach chosen in Net-Juggler [5], a cluster extension for VR-Juggler [6]. In this case, applications must share input events and must be synchronized at each frame buffer swap. This is particularly efficient when an application was already developped with VR-Juggler. However, this scheme is the opposite of a true distributed system: for a complex application, resource consuming tasks (surface computations, collision handling, etc.) will be replicated "as is" on each node of the cluster. In addition, for commercial applications requiring licencing, one licence must be bought for each node.

The second approach developped in Syzygy [3] or Avango [4] is to replicate only the scene descriptor (generally represented as a graph) over the PC cluster. This approach enables a single master application to manage and share synchronized updates of a scene with slave nodes using a distributed scene graph. Software proposing this solution are generally big application frameworks. Therefore, porting an existing application to such a distributed rendering environment would prove to be a tedious task.

The approach of Chromium [7] (formerly WireGL [8]), is to broadcast OpenGL commands emitted by the master to the slaves. This approach was developed in order to perform tiled rendering, but is also viable for multi-wall VR rendering systems. The main advantage of this method is that desktop applications can be ported to PC clusters without any modification of their codes, by tuning Chromium configuration files.

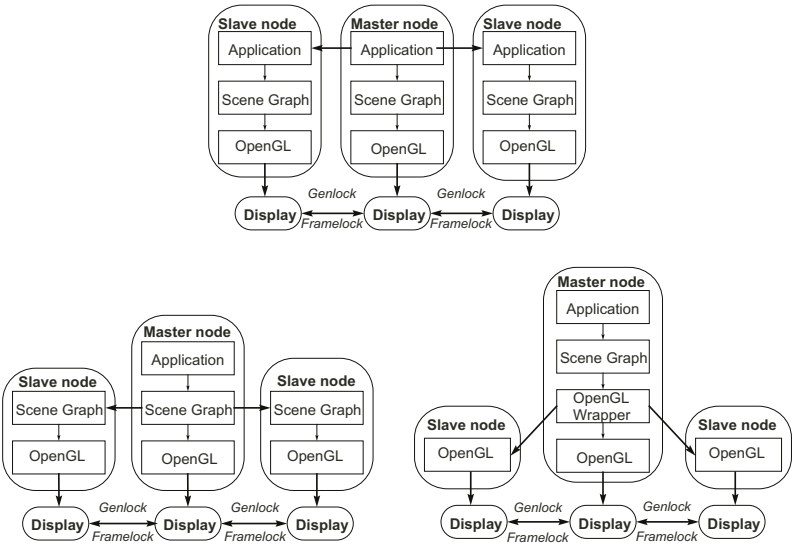

Fig. 1. (top) Application replication approach. (left) Scene graph distribution approach. (right) OpenGL commands broadcasting approach

3 DRS Approach

3.1 Architecture

DRS implements a master-slave approach (see Figure 2) by distributing VR 3D objects from the master node to the slaves nodes via networking. Each slave is a dedicated graphical node connected to a given display (e.g. a CAVE wall). VR database traversal and rendering optimizations such as frustum culling are performed on the rendering nodes of the cluster.

In addition to framelocking, the swap buffer commands on the rendering nodes must be issued within a very short time interval in order to synchronize rendering of the different nodes. DRS provides such a rendering mechanism in a transparent fashion, using sync message passing between the nodes. The master application is only in charge of telling DRS slave nodes when changes relative to the current rendered frame are to be commited.

DRS does not provide the programmer a full scene graph implementation but rather a set of classes to manage a 3D distributed database. Indeed it can be remarked that scene graphs are specific of each application field. For example CAD scene graphs are best described by a combination of CSG and B-Rep, and would prove excessively cumbersome to match CAD data to a generic scene graph format. Let us now describe in further details the DRS API.

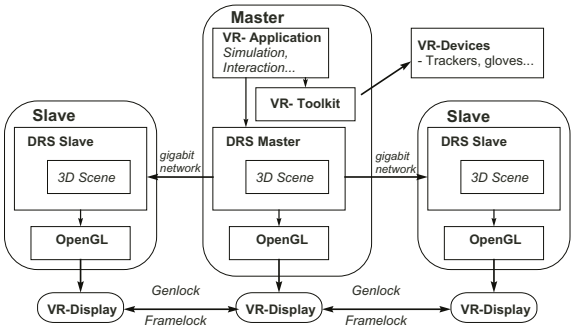

Fig. 2. DRS Architecture for a 3-PCs cluster

3.2 Objects Management

DRS maintains a database of graphical objects on both master and slaves nodes. Objects managed in the current prototype include triangular meshes, polygonal lines, points clouds, lights, materials and textures. Transformation matrices for object positions are also shared across the cluster. Projection parameters such as interocular distance or clipping planes distances can be dynamically sent by the master to the slaves in order to match the physical projection setup.

The interface between the application and DRS consists of three principal operations on each category of object:

- Create: adding a new object in the DRS database.
- Update: commiting changes on an already created object.
- Delete: suppressing an object from the database.

In the case of structured geometry, e.g. triangular meshes, three different update commands must be considered depending on what changed in the mesh (see Figure 3):

- Update topology: only the topology of the mesh has changed. In our case, the topology consists, for each triangle, in three indices corresponding to the three vertices of the triangle.
- Update geometry: only the geometry (vertices coordinates, normals or texture coordinates) of the mesh has changed. This case often occurs with triangulations performed on a regular lattice. When the surface definition changes, the coordinates of vertices in the triangulation are updated but the triangle connectivity and topology remains identical.
- Update topology and geometry: the two previous updates are performed one after the other.

3.3 Data Compression

In order to lower bandwidth requirements, mesh geometry is compressed by the master before being distributed, and the slaves must decompress it to perform

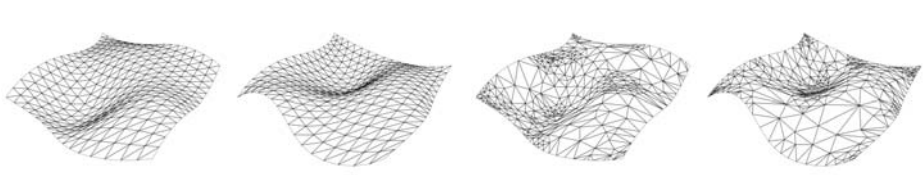

Fig. 3. Modification of NURBS surface with regular triangulation, (a) and (b), only vertices coordinates and normals need to be updated, the triangle connectivity remains the same. With adaptative triangulation, (c) and (d), both geometry and topology of the mesh change

rendering. To achieve a real optimization, the cumulated time of compressing the data, sending the compressed mesh and decompressing the data must be less than the time needed to send the uncompressed mesh. Really fast methods, without pre-computing steps, are thus needed. For example, in the case of modeling, objects are created in real-time and pre-computing complex data structure is out of the question. This constraint precludes the use of sophisticated compression algorithms, for example the ones of Taubin [9] and Rossignac [10] in which a prerequisite is the computation of the mesh connectivity graph.

To reduce the network bandwidth even further, multiresolution objects could have been envisaged. However, this kind of structure generally require costly pre-computing steps, and cannot be used in real time for distribution purposes.

Geometry compression. To compress vertex coordinates, we follow Deering's quantization approach [11]: first, vertex coordinates are normalized into a unit cube. This step requires computing the bounding box of the mesh, then performing a division per coordinate. Second, normalized coordinates are rounded off to fixed length integer. The size of each integer can be controlled, but 12 bits generally give good results for compression (36 bits per vertex instead of 96) and lead to no noticeable loss in quality for most 3D models.

Normals and texture coordinates are compressed in the same way, however the normalization step is omitted as normals components are in the $[-1, 1]$ range and texture coordinates in the $[0, 1]$ interval. They are represented on 10 bits for each component, resulting in 30 bits per normal and 20 bits per texture coordinates.

In traditional methods [11], following quantization, geometry storage is reduced with variable length encoding, generally using Huffman codes. To compress this kind of data correctly, vertex predictors are used, e.g. coding the difference between the current vertex and his ancestor in a vertex spawning tree of the mesh.

To perform well, Huffman coding requires to build a value histogram and a prefix code tree for any given dataset. Unfortunately, our experiments showed that this procedure is too time-consuming for any gain over sending raw quantized values.

Connectivity compression. A popular way to compress connectivity information is triangle strip generation [12]. A triangle strip of N triangles is represented

by $N + 2$ vertices instead of $3N$ in immediate mode. Triangle strips reduce the storage space needed and improves the rendering performance as well because they are at the hardware level with OpenGL. However, in order to build triangle strips, an adjacency graph based on the triangles of the mesh is needed, and again the computation time of such a graph is prohibitive if not readily available along with the mesh itself. In conclusion, connectivity information can be compressed fast with simple methods such as the SGI [13] algorithm, if it is computed ahead of time, as it is the case in some applications.

4 Implementation and Results

For the sake of simplicity, the DRS library was implemented over the TCP protocol. Thus it requires the master establishes one connection for each rendering slave node. To avoid this potential bottleneck the master can be equiped with several network cards : this is a solution already provided by PC cluster manufacturers.

DRS graphical nodes use pure OpenGL commands for rendering. As a consequence, DRS can be used in conjunction with any VR-toolkit that creates and manipulates an OpenGL context for drawing. For example, the projection setup for VR display on the graphical nodes can be specified with VR-Juggler [6]. DRS itself provides methods to setup CAVE-like multi-wall displays, in case users do not want to rely on an extra VR toolkit.

DRS was tested on a system configuration composed of a master PC and two slave nodes equipped with 3DLabs Wildcat graphic cards for genlocking and framelocking capabilities. The network theoretical bandwidth is 1 Gbit/s (700 Mbits/s in practice). Framelock, which is required to synchronize the frame buffer swaps of the graphics PCs, is performed via an OpenGL extension[1]. Cluster nodes can run either on Linux or on Microsoft Windows operating systems.

As expected, we noticed that the benefit of geometry compression is dependent of the speed of processors and the bandwidth of the network: CPU speed determines compression and decrompression times whereas network speed sets the data transmission time. Our program test is a simple linear elastic system deforming a mesh of about 11K vertices (25000 faces). It has been tested on a single PC, then on a networked cluster (one master + 2 slaves) with and without geometry compression. The results are conclusive (Table 1) and can be further improved with low level assembly programming (such as SSE extensions) for the compression and decompression routines.

DRS has been validated on two applications (Figure 4) ported from a high-end, shared memory, dedicated multiprocessor machine to the testing PC cluster. In a VR-CAD system, users create and edit digital mockups in an immersive environment, and the system provides real-time feedback to designers when deforming objects with detailed geometry. A DNA molecule viewer [14] was also ported to PC cluster thanks to DRS. While less demanding in terms of deforma-

[1] http://oss.sgi.com/projects/ogl-sample/registry/I3D/wgl_swap_frame_lock.txt

Table 1. DRS benchs

Configuration	Performances
Single	54 frame/s
DRS 1 Master + 2 Slave	33 frame/s
DRS 1 Master + 2 Slave with geometry compression	43 frame/s

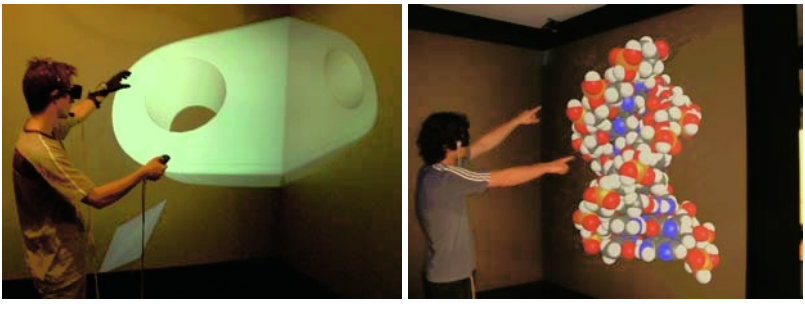

Fig. 4. Virtual Reality CAD application and DNA molecule immersive viewer in a CAVE-like system running on a PC cluster with DRS

tion, this later application typically manipulates thousands of DNA bases while displaying entire chromosomes.

5 Conclusion

In this paper, a 3D database distribution library for VR rendering on a PC cluster was presented. DRS provides a simple yet comprehensive interface to programmers wishing to port their application to VR immersive displays, while retaining high performance regarding the distribution of deformable objects. DRS was successfully put to use by allowing two existing applications (a CAD modeling experimental software and a DNA molecule viewer) to be successfully ported from a high-end visualization system to a PC cluster.

DRS could be extended to managing other modalities, e.g. haptics. Indeed a similar 3D objects distribution problems arise in haptics when connecting more than one force feedback or vibrotactile device to the VR system. In association with a haptic rendering software, DRS can allow multiple haptic systems to share a synchronized version of a dynamic 3D database. Because haptics need high refresh rates and detailed geometry description, DRS will probably be especially suited to force-feedback distributed rendering applications.

A natural evolution of DRS would consist in using multicast protocol to lower the necessary bandwidth. Multicast would allow the master cluster node to send data only once to all the slaves. However, the drawbacks of multicast are packet loss and packet mis-ordering. To manage these problems, checking algorithms have to make sure the data received by the slaves are correct. These techniques induce significant overhead and reduce the available bandwidth. Several prelim-

inary tests on our system (consisting in 2 slaves and one master), showed that overhead is too important compared to the actual multicast gain. However in a 6 walls/6 slaves configuration, multicasting can be reconsidered.

For the moment, DRS can deal with triangular meshes, polylines and points clouds. The system could be improved by supporting NURBS curves and surfaces or conics for example. These primitives would allow to reduce the data flow for specific applications, such as CAD for the NURBS and molecular visualization for the conics.

References

1. Cruz-Neira, C., Sandin, D., DeFanti, T.: Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In: Computer Graphics (Proceedings of SIGGRAPH '93). (1993) 135–142
2. Kruger, W., Bohn, C., Frohlich, B., Schuth, H., Strauss, W., Wesche, G.: The Responsive Workbench: A Virtual Work Environment. IEEE Computer **28** (1995) 42–48
3. Schaeffer, B., Goudeseune, C.: Syzygy: Native PC Cluster VR. In: Virtual Reality Conference '03, IEEE (2003) 15–22
4. Tramberend, H.: Avango: A Distributed Virtual Reality Framework. In: Proceedings of Afrigraph '01. (2001)
5. Allard, J., Gouranton, V., Lecointre, L., Melin, E., Raffin, B.: Net Juggler : Running VR Juggler with Multiple Displays on a Commodity Component Cluster. In: Virtual Reality Conference '02, Orlando, Florida, IEEE (2002) 273
6. Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., Cruz-Neira, C.: VR Juggler: A Virtual Platform for Virtual Reality Application Development. In: Virtual Reality Conference '01, Yokohama, Japan (2001)
7. Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T.: Chromium: A Stream Processing Framework for Interactive Rendering on Clusters. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002) **21** (2002) 693–702
8. Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., Hanrahan, P.: WireGL: A Scalable Graphics System for Clusters. In: Proceedings of SIGGRAPH 2001. (2001) 129–140
9. Taubin, G., Rossignac, J.: Geometric Compression Through Topological Surgery. ACM Transactions on Graphics **17** (1998) 84–115
10. Rossignac, J.: Edgebreaker: Connectivity Compression for Triangle Meshes. IEEE Transactions on Visualization and Computer Graphics **5** (1999) 47–61
11. Deering, M.: Geometry Compression. In: Computer Graphics (Proceedings of SIGGRAPH '95). (1995) 13–20
12. Evans, F., Skiena, S., Varshney, A.: Optimizing Triangle Strips for Fast Rendering. In: Proceedings IEEE Visualization '96. (1996) 319–326
13. Akeley, K., Haeberli, P., Burns, D.: tomesh.c. C Program on SGI Developer's Toolbox CD (1990)
14. Hérisson, J., Gros, P.E., Férey, N., Magneau, O., Gherbi, R.: DNA in Virtuo: Visualization and Exploration of 3D Genomic Structures. In: Proceedings Afrigraph '04. (2004) 35–40