A COMPRESSION SCHEME FOR EFFICIENT REMOTE STREAMING OF DYNAMIC 3D CONTENT

Giuseppe Marino, Paolo S. Gasparello, Davide Vercelli, Franco Tecchia and Massimo Bergamasco
PERCRO, Scuola Superiore Sant'Anna, Pisa, Italy

Keywords: Distributed rendering, Distributed applications, Remote graphics, Delta compression, Geometric compression.

Abstract: Real-time 3D content distribution over a network (either LAN or WAN) requires facing several challenges, most notably the handling of the large amount of data usually associated with 3D meshes. The scope of the present paper falls within the well-established context of real-time capture and streaming of OpenGL command sequences, focusing in particular on data compression schemes. However, we advance beyond the state-of-the-art improving over previous attempts of "in-frame" geometric compression on 3D structures inferred from generic OpenGL command sequences and adding "inter-frame" redundancy exploitation of the traffic generated by the typical architecture of interactive applications.

1 INTRODUCTION

In this paper, we propose a method to allow efficient distribution of OpenGL streams over the Internet by means of various form of data compression. This work stems from our previous research in the context of cluster-based immersive visualisation; in this field, intercepting OpenGL or Direct3D calls by means of custom system drivers and distribute them over a network has proved to effectively decouple the application architecture from the underlying rendering system, allowing for a reconfigurable rendering nodes arrangement.

While previous work in this field (Humphreys et al., 2001)(Humphreys et al., 2002)(Yang et al., 2002) focused on multicasting command streams over Local Area Networks, our goal was to extend this kind of cluster-based rendering architectures to work over WAN networks too, as this would allow for a new generation of network streamed 3D applications. We propose here an approach that addresses the low-bitrate requirement and the real-time constraint challenges using a combination of inter-frame and intra-frame compression of OpenGL command streams as well as using light-weight compression/decompression schemes.

2 RELATED WORK

A number of approaches have been proposed to stream graphical commands over a network in a distributed rendering architecture. WireGL (Humphreys et al., 2001), is one of the most relevant examples. It is a software system developed to distribute the graphic load of an OpenGL application to a cluster of machines. A graphical command stream is generated by means of a custom device driver library capable of intercepting the OpenGL API calls invoked by a generic application.

The software project Chromium (Humphreys et al., 2002) is a further refinement of the WireGL concept. It inherits the interception mechanism of its predecessor, but system configuration become more flexible, allowing to build a graph that describes the complete network structure and can be used to configure each single node.

AnyGL (Yang et al., 2002), a large-scale hybrid distributed graphics system, deals with the problem of managing the large amount of data generated by a OpenGL streaming application, introducing the concept of data compression for the first time. In this case geometry data redundancies are reduced by using simple position predictors. Vertex normals (Deering, 1995) and colors are compressed in similar ways.

In our approach we propose the adoption of state-of-the-art geometry compression techniques (Alliez and Gotsman, 2005). In particular, we concentrate on the mesh *connectivity* compression. We adopt the

Marino G., Simone Gasparello P., Vercelli D., Tecchia F. and Bergamasco M. (2010).
A COMPRESSION SCHEME FOR EFFICIENT REMOTE STREAMING OF DYNAMIC 3D CONTENT.
In *Proceedings of the International Conference on Computer Graphics Theory and Applications*, pages 267-270
DOI: 10.5220/0002834802670270
Copyright © SciTePress

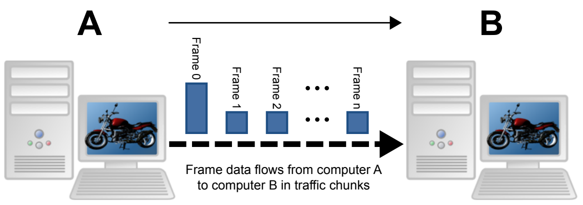
A
B
Frame 0
Frame 1
Frame 2
Frame n
Frame data flows from computer A to computer B in traffic chunks

Figure 1: Streaming OpenGL comands over a network connection from A to B: each frame generate a chunk of OpenGL data. Larger frames are usually generated at application bootstrap time.

valence driven approach, first introduced by Touma and Gotsman (Touma and Gotsman, 2000) and subsequently improved by Alliez and Desbrun (Alliez and Desbrun, 2001). In this technique the output symbol stream is the sequence of the vertex valence values, whose order is given by the conquest process. The mesh connectivity is compressed on average with 1.5 bits per vertex for regular meshes.

Differential compression is another approach to face the issue of data transmission. It consists in representing the differences between a source and a target file in a compact way. This technique is mostly based on the diff algorithm (Hunt and McIlroy, 1976). It produces, given a target and a source file, a *patch* containing a sequence of delete, change and append operations. In their work, Korn and Vo (Korn and Vo, 1995) proposed *vdelta*, an algorithm based on the Lempel-Ziv'77 approach (Ziv and Lempel, 1977). Afterwards they proposed a newer version vdelta called *vcdiff* (Korn and Vo, 2002). In our work we employ a Google code open source implementation of this algorithm (Open-vcdiff, 2008).

3 CAPTURING OPENGL STREAMS

We organized our work around our own software system for the capture and distribution of OpenGL calls performed by a graphical application. Our system exploits a Chromium-like network scheme: there is a *master* node, where the graphical application is running, and a number of *slave* nodes, that receive the OpenGL stream generated by the master node and broadcasted into the network. The master node uses a custom device driver interposed between the application and the OpenGL system library, capable of intercepting any call performed. Every time a call is intercepted and before its execution, the custom driver creates a ghost command code to be streamed over the network. In this way each slave, once received the stream, can replicate all the OpenGL calls, reconstructing the master's OpenGL state and graphical output.

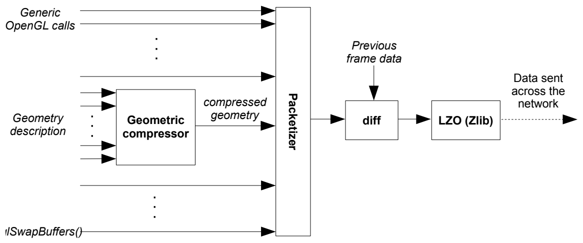

Figure 2: Overall system working scheme (master).

4 THE OVERALL ARCHITECTURE

Once the OpenGL calls have been intercepted with the mechanism described in the previous section, they are passed to the *packetizer* module (Figure 2). This component is in charge to encode all the information about performed calls and then stores them into a *command buffer*. If the frame contains geometry description commands, a geometric compression module will handle them, and send the compressed output to the packetizer. Once this process has been completed, the buffer content is compared against the previous frame by the *diff module*, as described in Section 6. The output of this operation is then compressed with a general purpose compressor (LZO or Zlib) and sent over the network. On the slave node the original data is reconstructed by decompression phases performed in reverse order with respect to the master side.

5 COMPRESSING FRAMES GEOMETRY

Transferring complete 3D model descriptions over the Internet may represent a serious problem. If models are large and/or the network link is not fast enough, slave node applications may stutter, being interrupted by long pauses spent for model synchronization.

We approach this issue by introducing a mesh encoder component, based on the combination of several general purpose and specific known compression techniques (Figure 3). Connectivity and geometry data is recognised by analyzing a sequence of OpenGL API calls, and then compressed with our scheme.

Some functional block of the process requires for the input model to be manifold. Therefore, in the

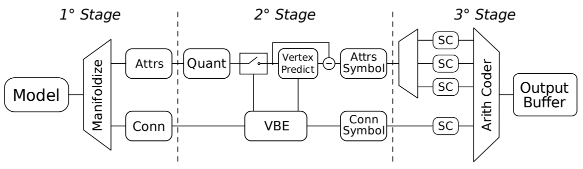

Figure 3: The architectural scheme of the 3D mesh compressor.

first stage, the mesh topology is checked and adapted (Gueziec et al., 1998) to comply with the manifold constraint. In this phase, vertex attributes and connectivity data are separated to feed the next stage.

In the second stage, the valence based approach (VBE block) (Touma and Gotsman, 2000)(Alliez and Desbrun, 2001) drives a mesh traversal using connectivity information, and outputs symbols whenever new topological elements are encountered. As soon as a vertex gets conquered, its attributes are quantized and predicted with the parallelogram rule (Touma and Gotsman, 2000), so that only differences are transmitted. A final compression stage provides entropy minimization by means of a fast adaptive context-based arithmetic coder (Salomon, 2004).

6 EXPLOITING FRAME TO FRAME COHERENCE

The OpenGL frame structure of a typical 3D application begins with some frames dedicated to the description of the 3D objects composing the scene. A lot of data is generated at this point, ant it can be compressed as described in the previous sections. After that, if the 3D model is no longer modified, the following frames often consist in the exploration of the model itself. Each frame is mainly a collection of drawing calls and calls to change the camera placement. It is clearly visible how the "exploration frames" are similar: they only differ by the position of the camera. So, once the drawing calls are sent for the first time (inside the OpenGL stream associated to the first frame) the only information really needed to reconstruct the subsequent frames are the updated camera positions.

To exploit frame-to-frame coherence we use a diff algorithm, as described in Section 2. Figure 4 shows how the diff operation is performed by our system. The OpenGL stream associated to each "exploration" frame can be seen as the sum of its geometry (*G*) and camera setting (*C*) sections. *G* sections are the same for all the frames, while the *C* sections are frame dependent. While the first frame is sent unmodified, for each of the subsequent frames an incremental packet

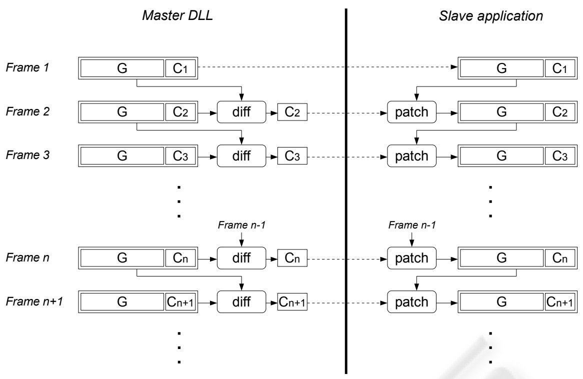

Figure 4: Working scheme of the diff compression. Dashed arrows represent packet sending over the network.

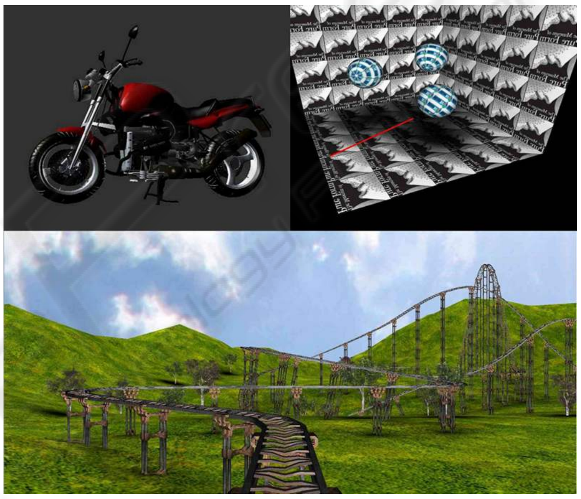

Figure 5: Some view of the testbed applications.

is generated and sent over the network.

With this technique it is possible to avoid sending a large percentage of the frame data every time. In many cases frames with sizes of the order of several kilobytes can be represented with just a few dozen bytes.

7 MEASUREMENTS

We have conducted formal testing of the compression methods: measurements were taken using as a testbed a chromium-like cluster rendering system that we have developed in house over the years. We selected three test applications to be rendered on our framework, selected for their visual complexity, in order to cover a wide range of real-life scenarios: a simple scene with just a small number of moving objects, a complex CAD model being manipulated in real-time and a complex scenario traversed by the camera with a first-person perspective (Figure 5). We connected two computers over a network as in Figure 1: at one side of the connection (A) the original application runs. Its OpenGL command stream is intercepted and sent to the slave node (B) for being remotely executed. The goal of our test is to evaluate if the generated traffic is compatible with typical Internet bandwidth, therefore we measured the size of the data chunks in various conditions.

Preliminary results show that geometric compression reduces the traffic load at application bootstrap (and every time some new geometry is kicked-in), while frame-to-frame compression minimizes the amount of incremental data that needs to be exchanged as frames advance, exploiting the high redundancy in data flow for interactive applications. Numbers obtained suggest an average reduction for the first frame to $1/3$ of the original data size and for subsequent frames to $1/8$.

8 CONCLUSIONS

We have presented a method to allow efficient distribution of real-time generated content using on the fly compression and decompression of OpenGL command streams. We advance beyond the state-of-the-art improving over previous techniques of in-frame geometric compression on 3D structures inferred from generic OpenGL command sequences and adding inter-frame redundancy exploitation of the traffic generated by the typical architecture of interactive applications. Measurements reveal for this combination of techniques a very effective reduction of network traffic obtained with a modest CPU overhead. This suggest a significant application potential of the technique whenever the amount of data bandwidth is limited, such in the case of Internet-based 3D streaming.

REFERENCES

Alliez, P. and Desbrun, M. (2001). Valence-Driven Connectivity Encoding for 3 D Meshes. *Computer Graphics Forum*, 20(3):480–489.

Alliez, P. and Gotsman, C. (2005). Recent advances in compression of 3D meshes. *Advances in Multiresolution for Geometric Modelling*, pages 3–26.

Deering, M. (1995). Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 13–20. ACM New York, NY, USA.

Gueziec, A., Taubin, G., Lazarus, F., and Horn, W. (1998). Converting sets of polygons to manifold surfaces by cutting and stitching. *IEEE Visualization*, 98:383–390.

Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., and Hanrahan, P. (2001). Wiregl: a scalable graphics system for clusters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 129–140, New York, NY, USA. ACM.

Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. (2002). Chromium: A stream-processing framework for interactive rendering on clusters.

Hunt, J. W. and McIlroy, M. D. (1976). An algorithm for differential file comparison. Technical Report CSTR 41, Bell Laboratories, Murray Hill, NJ.

Korn, D. and Vo, K. (1995). vdelta: Differencing and compression. *Practical Reusable UNIX Software. John Wiley & Sons.*

Korn, D. G. and Vo, K.-P. (2002). Engineering a differencing and compression data format. In *ATEC '02: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, pages 219–228, Berkeley, CA, USA. USENIX Association.

Open-vcdiff (2008). An encoder/decoder for the vcdiff (rfc3284) format. http://code.google.com/p/open-vcdiff.

Salomon, D. (2004). *Data Compression: The Complete Reference*. Springer.

Touma, C. and Gotsman, C. (2000). Triangle mesh compression. US Patent 6,167,159.

Yang, J., Shi, J., Jin, Z., and Zhang, H. (2002). Design and implementation of a large-scale hybrid distributed graphics system. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 39–49. Eurographics Association Aire-la-Ville, Switzerland, Switzerland.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343.