Integrated Parallel Rendering for AVS/Express

Louise M. Lever and James S. Perrin
Manchester Computing, University of Manchester

Enabling integrated multi-processor computation and rendering for High-Performance Visualization

Introduction

AVS/Express is a leading visualization application development package, which was described in CSAR Focus Issue 10. To summarize, the AVS/Express visualization package was extended by the Manchester Visualization Centre, in collaboration with AVS, KGT and SGI. The first goal was to provide a multi-pipe parallel renderer, enabling AVS/Express to be used in immersive environments such as a CAVE or RealityCenter, while increasing rendering performance by parallelizing the rendering as much as possible. The shortcoming of this project was that many users were left with a single CPU for the computation of their visualization application. Our second goal was therefore to develop a toolkit to enable parallel computation within AVS/Express. The two projects are respectively known as Multi-Pipe Express (MPE) and the Parallel Support Toolkit (PST).

This article continues with the progress of the integration of the MPE and PST projects, to produce a powerful visualization architecture for high-performance computation and visualization.

Overview

AVS/Express MPE and PST provide a strong visualization application development system. PST allows users to use parallelized modules within their applications to perform visualization tasks across many processors, in both shared memory systems and clusters. The end result of PST computation is typically a mesh dataset e.g., the sets of triangles produced in an isosurface calculation. For desktop users and existing MPE users, this data would normally be returned to the main application, pre-processed for rendering i.e., conversion to triangle strips and mapping of datamaps to RGBA colours, and then passed to the rendering system. For MPE users wishing to visualize large datasets this is a serious drawback, as all renderable geometry is channelled through the single threaded main application for pre-processing and rendering distribution. Even shared memory systems are impaired by the serialized pre-processing stage. Figure 1 shows the standard rendering architecture.

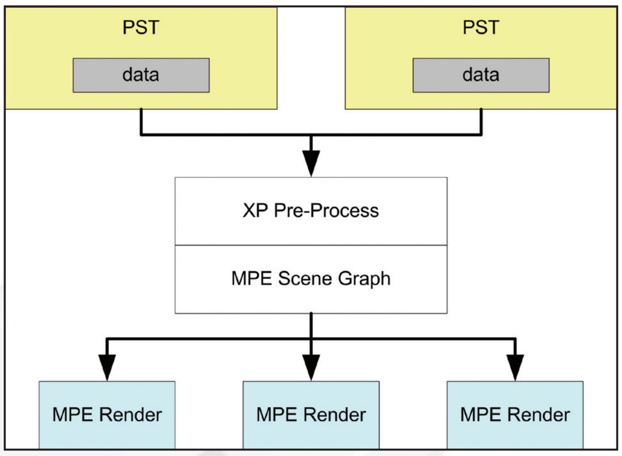

Figure 1: Standard Rendering Architecture, where PST output data is sent back to the main application for pre-processing before being distributed for parallel rendering.

The first step towards the development of an integrated system was to parallelize the pre-processing stage. In this mode of operation the PST nodes are responsible for pre-processing of their generated output. For desktop or single-pipe users this parallelization still provides a considerable performance gain. Figure 2 shows the Level 1 Parallel Rendering Architecture, where pre-processing is performed in parallel by PST before being sent back to the main application for rendering distribution. The second step is to enable tighter integration with the MPE rendering system. Typically, renderable data is encapsulated by the MPE renderer and distributed to the rendering nodes. Recent changes to MPE now effectively enables "empty" objects to be placed into the scene-graph for rendering. The main application is now only responsible for controlling the position and orientation of these objects with the context of the scene. In this mode, PST is now also responsible for encapsulation of its renderable data, ready for rendering by MPE. On both shared memory systems and cluster solutions one or more Distributed Object Handlers (DOH) listen for incoming objects from the PST nodes, the contents of which are placed into the "empty" objects. Figure 3 shows the Level 2 Parallel Rendering Architecture, where both pre-processing and rendering encapsulation are performed by PST in parallel, with the resultant data sent directly to the rendering nodes via the Distributed Object Handlers.

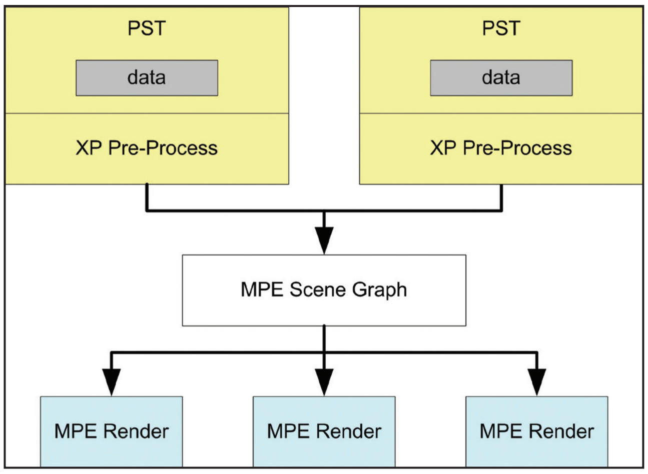

Figure 2: Level 1 Parallel Rendering Architecture, where PST output is pre-processed in parallel before it is returned to the serial application for rendering distribution.

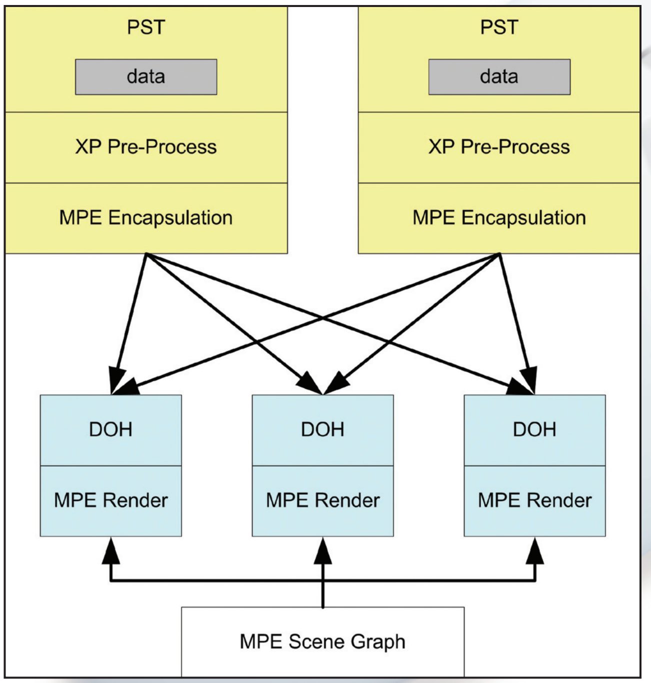

Figure 3: Level 2 Parallel Rendering Architecture, where PST output is both pre-processed and encapsulated ready for rendering and is sent directly to the Distributed Object Handlers as needed. Only empty objects are contained within the main scene graph.

The previous bottleneck is now avoided and reference objects are rendered within the application scene-graph. By providing a set of DOH threads, further improvements have been made, adding modes of operation allowing the reference objects to be rendered synchronously or asynchronously. Most visualization tasks within AVS/Express (and PST) are carried out in chunks, so for example one isosurface is comprised of many chunks of a chosen number of triangles. Such chunking of data improves handling of memory and rendering speed. Under asynchronous behaviour, chunks of renderable data are rendered and hence cached by the graphics hardware immediately. This approach has two benefits, a) users can see the visualization progressively appear as it is produced, which is particularly useful for large datasets and b) the graphics pipes can cache more easily when geometry is streamed into them.

Synchronous behaviour is used when progressive updates are not desired and the user wishes to see the new output when it is all available and ready to display.

With appropriate multipipe configurations it is possible to achieve database decomposition techniques for both shared memory and distributed cluster architectures. Database decomposition allows many direct PST to MPE connections, where each MPE node renders only the partial scene-graph generated by a local PST node. Only a final compositing stage is required to produce the final image. Figure 4 shows a distributed architecture enabling Database Decomposition, where each rendering node only renders 1/3 of the isosurface data and frame-buffer compositing is used to get the final image. Figure 5 shows an example of the Lattice Boltzmann dataset as rendered using database decomposition.

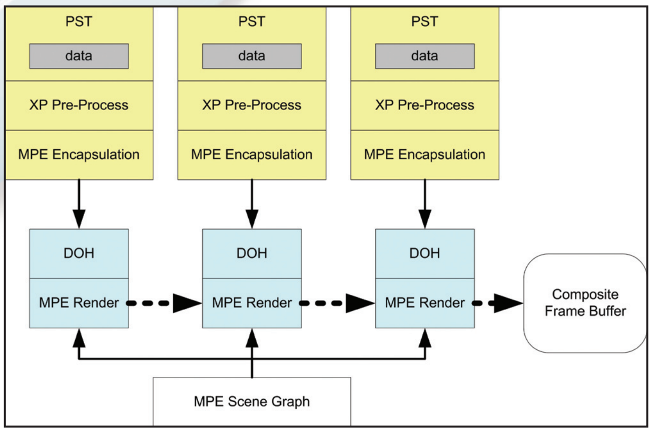

Figure 4: Example of a configuration for enabling Database Decomposition, where each render node only renders a partial scene-graph and frame-buffer compositing is used to get the final image.

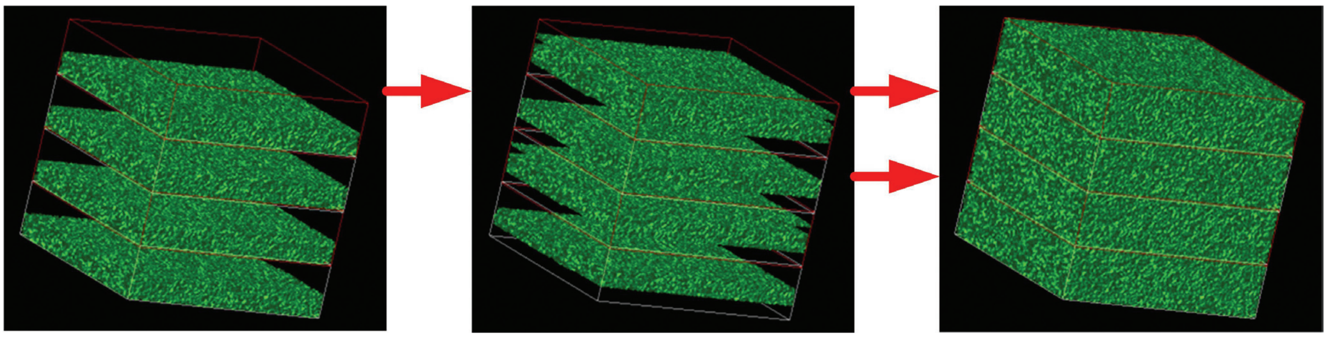
Figure 5: The Lattice-Boltzmann dataset as rendered with Database Decomposition and Frame-Buffer Compositing.

The Next Hurdle

As high-performance computation and visualization increase in complexity and sheer problem size, the availability of graphics resources becomes the next major bottleneck facing users. While many customers do possess systems with thousands of CPUs, most current visualization solutions are limited to a few graphics pipes, typically in the region of 4 to 16.

Given the size of some problems and the utilization of thousands of CPUs using PST, the relatively small number of pipes becomes a bottleneck in itself. Hundreds of CPUs producing renderable data will now swamp the pipes, leaving users unable to visualize satisfactorily.

Our next major task therefore is to develop a Massively Parallel Renderer (MPR) which will use the available CPUs as software based render nodes, bypassing the need for hardware graphics pipes. The AVS/Express MPE Renderer will be extended to scale up to possibly thousands of processors and enable a software based solution. To achieve this, the MPR will build upon the existing OpenGL solution by employing the MesaGL library. In this manner the existing framework within MPE and PST will be able to migrate without the need for a whole new render system to be developed. However, there are still remaining issues to be considered, which will require investigation if the MPR is to be successful. The main obstacles to success are a) scalability of PST computation, b) scalability of MPE render nodes and c) efficient management of large-scale frame-buffer composition. The latter problem requires careful consideration of distribution and composition techniques, if the MPR is to handle thousands of generated frame-buffers.

Contacts

Louise M. Lever:
louise.lever@manchester.ac.uk

James S. Perrin:
james.perrin@manchester.ac.uk

George W. Leaver:
george.leaver@manchester.ac.uk

http://www.sve.man.ac.uk/mvc/Research/mpe/
http://www.sve.man.ac.uk/mvc/Research/PST/