Distributed Rasterization using OpenGL

David Calvert  
Department of Computing and Information Science  
University of Guelph, Guelph, ON, Canada  
dcalvert@uoguelph.ca

David Thompson  
School of Engineering  
University of Guelph, Guelph, ON, Canada  
dthomp02@uoguelph.ca

Abstract

This work examines the facility of using a large distributed memory system for rasterization of computer graphics using the OpenGL and GLUT libraries. Issues examined include the performance increases achieved through parallel processing and the effects of different methods for dividing the framebuffer over multiple processors.

Keywords: Rasterization, computer graphics, distributed system, OpenGL.

1. Introduction

The focus of this work is the parallel rasterization of images generated using OpenGL on a distributed Beowulf cluster. Benefits from this system include the simplicity with which it can be added to existing applications, the ease of implementation in conjunction with existing OpenGL rendering systems, and a modest performance increase.

2. Implementation Details

The parallel extension to OpenGL are written as a set of macros which replace existing function calls to GLUT commands. This has the benefit of adding the parallel functionality to existing programs by the inclusion of a single header file and recompilation of the source code.

Each processor receives a complete copy of the scene geometry and a description of the area of the framebuffer which that process is responsible for rendering. The process then renders its subset of the framebuffer from the given geometry and transfers the partial framebuffer to the parent process where they are assembled into the completed scene.

The system was developed on the University of Guelph SHARCNet facility. This consists of a Beowulf cluster of 108 Alpha ES50 processors connected with a 1.2 Gigabit, low latency network. MPI libraries were used to implement parallelism and communications between processes.

Testing was performed on polygon meshes of 948 to 67,240 polygons which were obtained through [7]. Polygons in all tests were lit from a single light source. Tests were made with both textured and non-textured polygons. Each test involved translating, rotating, and scaling the mesh through a predefined sequence of 180 frames. Images were rendered at 1024x768 resolution.

3. Results

Test renderings demonstrated that there was no performance increase, and there was often a slight decrease, in rendering non-textured polygons. When rendering textured and lit polygons the tests to render the scene in smaller segments demonstrated that the performance increases were affected by the method by which the framebuffer was divided. Divisions into rows of pixels, each rendered by one processor, was more effective than divisions based on columns or a mix of rows and columns. Table 1 shows the effect of dividing the framebuffer into from 1 to 30 segments. The greatest reduction in rendering time occurred when the framebuffer was divided into 30 rows. The improved performance derived from row based divisions is likely due to implementation characteristics of the rendering algorithm which benefits from row-based continuity.

The improvement in polygons per second throughput can be seen to improve with the number of polygons in the mesh. Figure 1 shows the effect of number of polygons on throughput. Although the number of polygons per second throughput increases with the number of polygons, the proportional size of the increase can be seen to diminish as the number of polygons increases.

The effect of multiple processors applied to different sized collections of polygons can be seen in figure 2 where each line represents a separate data set. The largest increase in performance occurs with the addition of a second processor. As more processors are added, the benefits achieved decrease. The improvement in overall performance can be seen to be relatively consistent regardless of the number of polygons in the scene. The performance improvements are
Table 1. Seconds to render 180 frames of a 69451 polygon model. Rows represent the number of divisions in the x axis, columns represent the number of divisions in the y axis.

therefore more pronounced on smaller collections of polygons which start with a smaller initial rendering time and receive a relatively greater effect through parallelization.

4. Conclusions

The system presented is easily integrated into existing systems and requires little knowledge of parallel programming. For relatively small collections of polygons the system provides a reasonable increase in performance. When used with larger collections of polygons the system’s performance decreases to very modest levels.

References