Polygon Rendering for Interactive Scientific Visualization on Multicomputers

A Dissertation Proposal
David Ellsworth

June 12, 1992

Thesis

I can design a methodology that, for an arbitrary large-grain multicomputer, will specifiy the highest performance polygon rendering algorithm for interactive scientific visualization.

I plan to demonstrate this thesis using three assertions:

Assertion 1: Classes of Polygon Rendering Algorithms

> *All parallel polygon rendering algorithms can be grouped into four classes: sort-first, sort-middle, sort-last, and sort-twice.*

This can be shown by examining a simplified graphics pipeline, one without shading. The simplified pipeline consists of transformation, scan conversion, and hidden-surface elimination. The start of the pipeline must be parallelized by partitioning the input primitives and assigning them to processors. The end of the pipeline must be partitioned in screen-space, as hidden-surface elimination requires information about all the primitives for a pixel. Therefore, since the partitioning methods for the two ends of the graphics pipeline are different, there must be a switch between the two partitioning methods.

Three classes of algorithms can be found by varying the place in the pipeline where the switch occurs:

- Sort-first algorithms make the switch before hidden-surface elimination, so the change in partitions requires a sort in screen space. Sort-first algorithms uses the frame-to-frame coherency of primitive's screen position in order to reduce the amount of sorting. In one sense, it tries to put the sort before the transformation by using coherency.

- Sort-middle algorithms make the switch after transformation, so it also requires a sort in screen space. Conceptually, these algorithms sort from one set of processors that perform the transformation to a second set of processors that perform the rendering.

- Sort-last algorithms make the switch in the middle of hidden surface elimination, after each processor has computed an image using the primitives assigned to it. The final part of hidden-surface elimination is done using screen-space partitions.

The fourth class of algorithms, sort-twice, consists of algorithms that make more than switch of partitioning methods. These algorithms use a partitioning criterion other than screen-space partitioning in the second set of partitions, then use screen-space partitioning for the third set of partitions. Because these algorithms make two switches between partitioning methods, each involving a sort, this class of algorithms is called sort-twice.

Assertion 2: Algorithm Class for Scientific Visualization

> *For interactive scientific visualization applications running on multicomputers, the polygon rendering algorithms with the highest average performance are in the sort-middle class of algorithms.*

I plan to show this by the use of analytic models to predict the performance of the first three classes of algorithms. I will argue that the algorithms in the last class, sort-twice, can be either shown to either reduce to special cases of the first three classes, or to add unnecessary work to other, more simpler, algorithms.

I have completed a preliminary set of models for the fastest algorithm expected for each class running on the Touchstone Delta and Paragon Sigma. The algorithms for each class are:

- Sort-first: uses adaptive screen-space partitions, with a single processor in charge of each polygon. Off screen polygons randomly distributed.
- Sort-middle: assigns fixed screen partitions to processors using triangle counts calculated during transformation.
- Sort-last: does not compress computed images to save on composition cost.

The models assume:

- 20% triangles trivially rejected, 5% clipped, 50 pixel triangles
- Perfect load balancing
- High-resolution frame buffer (1280 by 1024)
- Gouraud shading

The processing times used in the models come from reported values, and measured times from similar systems.

Arguments for Sort-Middle

- According to the model, sort-middle is roughly as fast as sort-first on the Touchstone Delta and Paragon Sigma. Each is faster than the other for some set of conditions.
- Sort-middle is much faster than sort-last according to the model.
- Sort-middle allows recalculation of the model in every frame without the application needing to maintain the screen-space coherency required by sort-first algorithms. Maintaining screen-space coherency would slow down the application.
- Sort-middle allows simple editing of hierarchical display lists. Hierarchical display lists must be flattened to be used with sort-first algorithms, adding to the cost.
- Sort-middle gives smoother performance than sort-first, as the system's performance does not change as rate of movement changes.
- When using specialized rendering hardware, sort-middle simplifies balancing the performance of the transformation and rendering processors. Algorithms in the sort-middle class do not depend on a correspondence between transformation and rendering processors.

Argument against Sort-Middle

- It requires O(number polygons + number pixels) global communication, while sort-last requires O(number pixels) local communication . As polygon performance increases, sort-last algorithms will become faster then sort-middle algorithms. However, anti-aliasing and deferred shading increase sort-last bandwidth by factors of 5 and 4.

Plan

I will continue to refine the analytic models, and support their results by implementing and testing sort-middle algorithms on two multicomputers. The simulation studies described later in the proposal will also help refine the models.

Assertion 3: Design Methodology

> *There is a clear design methodology for choosing the best sort-middle algorithm, given the characteristics of a multicomputer architecture.*

Sort-Middle Algorithm Space

The space of possible sort-middle algorithms has the following characteristics:

- Sort-middle algorithms need to partition and load balance both the transformation task and the rendering task.
- The two partitioning problems can be solved independently, as the sort between the two sets of processors sorts primitives from one partitioning method to the other.
- Different methods of partitioning and load balancing often trade better load balancing with an increased implementation cost.
- Using specialized rendering hardware puts constraints on the partitioning method, as it may constrain how the screen is subdivided or whether only a fraction of the screen can be rendered at a time.

Partitioning and Load Balancing Transformation

A different method is needed to partition and load balance the transformation task for the two types of application interfaces:

1. For applications using a hierarchical retained-mode interface, the display list can be partitioned by sending a skeleton of the structure hierarchy and attribute changes (transformation matrices, colors) to all processors, and sending successive primitives to different processors.
2. For applications using an immediate mode interface, the model partitioning must be done by the user, as the graphics library has no control over the placement of the model.

Plan

- Both solutions are running on Pxpl5.
- I will measure how well they balance the loads and the amount of parallel overhead.

Partitioning and Load Balancing Rendering

- An algorithm for partitioning and load balancing the rendering task must choose a screen-space partitioning method, how often the assignments are done, and a processor assignment method.
- The partitioning method affects the amount of triangle parallelism, the number of processors working on a single triangle. Having more processors work on a triangle helps the load balancing but also increases the communication cost.
- The number of partition-to-processor assignments per frame affects the amount of information available to processor assignment method. Assignments done later can use more up-to-date information. The possible choices are:
  - static assignment
  - assignment once per frame
  - assignments done during a frame

  Making more assignments per frame allows better load balancing, but at the cost of making and communicating the assignments.

Plan

- Find best screen-space partitioning method by simulating four candidate methods:
  - static assignment of interlaced regions
  - per frame assignment of fixed regions
  - intra-frame assignment of fixed regions
  - per-frame assignment of adaptive partitions
- Use recorded user interactions to drive simulations
- Calculate the total costs of the methods by adding up parallel overhead costs and work imbalance cost.

Implementations to Confirm Models and Simulation

While analytic models and simulations give results showing the performance of various polygon algorithms and partitioning methods, actual implementations are needed to confirm their results.Plan

- I have implemented a sort-middle algorithm on an architecture with specialized rendering processors, Pixel-Planes 5.
- I will implement an algorithm on a system without specialized processors, the Touchstone Delta.
- The delta implementation will be simple: it will display a static model using Gouraud shading.
- The two implementations will use different methods for partitioning and load balancing the rendering.
- I will characterize the implementations using an analytic model for Pxpl5, and measuring the components of parallel overhead for Delta

Schedule

Proposal Approved	June 92
Record user interactions	June 92
Simulation of load balancing algorithms complete	July 92
Finish Pxpl5 performance evaluation (with Jonathan Cohen)	July 92
Delta implementation prototype running on Pxpl5	August 92
Finish draft of first two chapters	August 92
Progress report to committee	End of August 92
Delta implementation running	September 92
Record Delta performance data	October 92
Write two more chapters and SIGGRAPH paper	Nov-Dec 92
Progress report to committee	Early Dec. 92
Do more analyses, simulation before writing	January 93
Write rest of chapters	February 93
Finish writing	March 93
Defend	April 93