John Wiggins
Semester Project Specification and Analysis
3/25/2003

Specification:

For my project, I will be creating a parallel volume renderer. The method of rendering will be raycasting. In raycasting, rays are projected from the "eye" or "camera" into a 3D volume dataset. An integration is done along each of the rays using the scalar values in the volume. Thus, each resulting pixel of the output image represents a "density" in that region of the volume. Volume rendering is an effective visualization method for many sources of data such as CT/MRI scans, cryo electron microscopy, geological formations, and artificial sources such as output from software simulations.

The parallel algorithm I plan to implement is a combination of methods which I have observed in other volume renderers. (but not in any raycasters that I know of) The processing resources are divided into two groups. Nodes which manage blocks of volume data(cache nodes) and nodes which do the actual rendering(render nodes). The first step of the process is to prepare the data. This step is very time consuming, but it only needs to be done once. The dataset is distributed evenly among the cache nodes in order to keep the processing load mostly balanced. To do this, the dataset is first split into equal sized blocks. Groups of these blocks are written to each cache node's local disk in order to take advantage of parallel I/O. For the actual rendering process, the output image is divided into rectangular patches which are evenly distributed (non-contiguously) among the render nodes. The reason for spliting the image into patches is to take advantage of locality when accessing volume data (which tends to be quite large, on the order of gigabytes). The patches are non-contiguous for load balancing reasons. If each render node renders patches from all over the image, it is more likely that each node will be doing roughly the same amount of work (because some rays may be terminated early or may not even intersect the volume). For each patch of screen, a frustum volume is created to determine which blocks of the volume data are needed to render the patch. These blocks are then fetched from cache nodes concurrently on the render node as rendering begins (using pthreads). The blocks are put into an LRU cache on the render node which is large enough to hold an average frustum's worth of volume data (this may not always be possible). The rendering is allowed to proceed as blocks are being fetched. If a ray encounters volume data not present in the cache, its state is saved and the ray is put aside while other rays are worked on. The ray's computation can be restarted later once the needed data is present. Once all render nodes have rendered their allocation of patches, the outputs from each render node is composited to form the final image.

Analysis:

NxN pixel output image / X*Y*Z voxels of volume data
NpxNp pixels per patch / Pc cache nodes / Pr render nodes
B = (X/Xb)*(Y/Yb)*(Z/Zb) blocks of volume data
Vb = Xb*Yb*Zb voxels per block of volume data

Assuming that the amount of work needed to generate a pixel falls within an average range for each pixel of the output image:
Each render node must render N*N/Pr pixels.
Each render node must fetch between B/Pr blocks (ideally) and B blocks (bad).
The final image can be merged in log(Pr) steps.

Speedup(Pr) = (N*N + B*Vb) / ((N*N/Pr) + (B*Vb/Pr) + log(Pr))

Efficiency(Pr) = Speedup(Pr) / Pr

Isoefficiency is a different matter. I plan to make the number of nodes dedicated to caching and the number of nodes for rendering settable at startup time. I'm guessing that this one parameter will have the greatest impacton the scalability of this algorithm. Other factors affecting scalability are the size of the patches that the image is divided into, the blocksize, the viewing transform (for instance, the zoom could be such that the entire volume is only visible in one patch of the image), and even the transfer function (which maps values in the volume to pixel values in the image). Because of these factors I can only begin to conjecture about the scalability of this approach.