Parallel Implementation: Finite Element Framework and Charm++

The SDG method is easy to parallelize when it is implemented on patchwise causal meshes, such as those generated by Tent Pitcher, because the solution on each new patch depends only on its immediate neighbors along its inflow boundary. The amount of data per patch is small, making it inexpensive to communicate patch data between processors.

Our parallel implementation is based on the Finite Element Framework, a software development environment that handles parallel details with minimal effort by the applications programmer. The framework builds on top of AMPI or native MPI and allows access to Charm++ (http://charm.cs.uiuc.edu).

To achieve this end, we take advantage of the modern programmable graphics processing units (GPUs); these heavily pipelined devices contain up to 64 parallel pixel units on a single card. We use this power to evaluate solution polynomials and to calculate surface normals and lighting, all on a per-pixel basis. This leads to high-accuracy visualizations (see Fig. 12), and allows us to distinguish solution artifacts from rendering artifacts.

Even with very tight error bounds, our SDG solutions can generate slightly different values in neighboring elements due to floating point imprecision. This can lead to pixel drop-outs that overstate the discontinuity. We overdraw all polygon edges to eliminate the pixel drop-outs while retaining legitimate solution discontinuities (Fig. 13).

Continuing Work

We will continue to develop the spacetime discontinuous Galerkin technology, while increasing our emphasis on its application to a variety of materials systems, including:

- Austenite-martensite transitions in shocked shape memory alloys
- Dewetting of inclusions in composites
- Solutions of the time-dependent Schroedinger equation in TD-DFT.
- New atomistic-continuum coupling strategies

Further development of the SDG technology will include:

- Simultaneous parallel/adaptive solutions
- Interface tracking (inclined tent poles)
- Extend to 3 space dimensions x time

Pixel-Exact Visualization of Spacetime Data Sets

Special post-processing and rendering techniques are needed to visualize data sets computed on unstructured spacetime grids, such as the ones displayed in Fig. 7. We take the trace of the data on a series of constant-time planes to produce animations and still images, such as the ones shown in Fig. 9.

Traditionally, visualizations are produced using standard polygon rendering algorithms that typically drawing one planar polygon per element, with linearly interpolated color. These approximate rendering methods can obscure the true quality of solutions computed with a high-order polynomial bases. In contrast, we wish to produce pixel-exact renderings at interactive rates.

Fig. 9 Sequence of images showing scattering of a shock wave by a crack tip. The shock front approaches the crack from the top edge and scatters from the crack tip at bottom center. The shear and pressure components of the scattered waves are clearly visible. The height field depicts velocity magnitude; color depicts strain energy density on a logarithmic scale.

Fig. 10 Parallel software architecture

Fig. 11 Domain decomposition of space mesh (left), linear scaling of solution rate with number of processors (right), and processor utilization for a 16-processor run (bottom).

Fig. 12 Comparison of per-vertex rendering (left) with pixel-exact rendering (right)

Fig. 13 Comparison of rendering without (left) and with (right) anti-aliasing along edges

Fig. 14 Scattering around circular inclusions