Finite Element Simulation of Nonlinear Elastic Dynamics Using Cuda

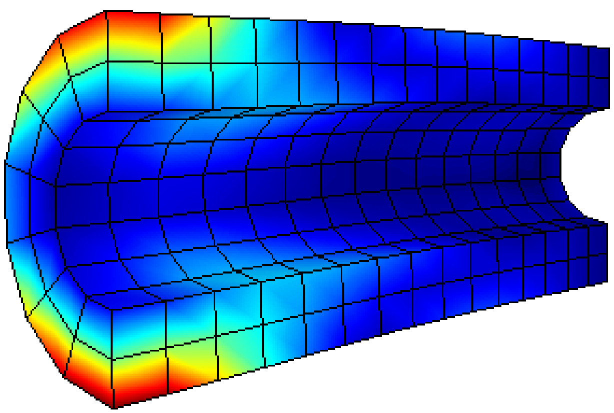
Christopher Cameron

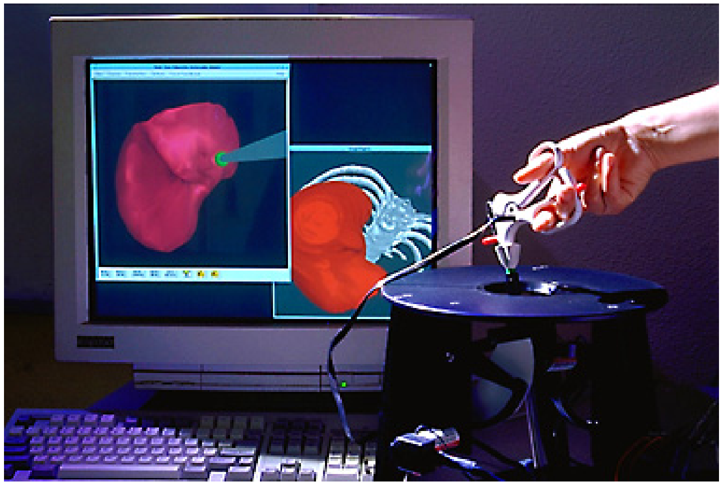
May 10, 2009

The Problem

Compute how an elastic object deforms over time when subjected to external forces.

- Many applications including stress-testing buildings and vehicles.
- Artifically constrain implementation to fit on one GPU (small-scale simulations are still have applications, e.g, surgical simulation).





The Discrete Time Differential Equation

Given the inputs

- A discretized domain Ω with n nodes and e elements
- A time-varying vector $q(t) \in \mathbb{R}^{3n}$ which describes the displacement of the nodes of the mesh
- A mass matrix $M \in \mathbb{R}^{3n \times 3n}$ describing the mass distribution in the body
- Internal force function resulting from deformation, $f_{\text{int}} : \mathbb{R}^{3n} \to \mathbb{R}^{3n}$, and its derivative $f'_{\text{int}} : \mathbb{R}^{3n} \to \mathbb{R}^{3n \times 3n}$
- A time-varying external force function from user input $f_{\text{ext}}(t) \in \mathbb{R}^{3n}$

Solve the differential equation (looks like Newton's second law)

$$M\ddot{q}(t) = f_{\text{int}}(q(t)) + f_{\text{ext}}(t)$$

Solution Details

Running a simulation consists of, for each time value t_i in $t_1, ..., t_m$, computing $q(t_{i+1})$ from $q(t_i)$. This involves

1. For each element, compute f_{ext} and f'_{ext} for just that element.
   - For the 8-node brick elements used, $f_{\text{ext}} \in \mathbb{R}^{24}$ and $f'_{\text{ext}} \in \mathbb{R}^{24\times 24}$
2. Assemble the per-element f_{ext} and f'_{ext} into whole-mesh f_{ext} and f'_{ext}
3. Solve a sparse symmetric positive definite linear system involving f_{ext}, f'_{ext}, and M

We focus on step 1 in this project. The remaining steps are very common and well-studied.

Implementation Details

- Implemented on a GeForce 8800 GT with 512 MB of memory.
- Only uses single precision floating-point.
  - GPUs with double precision are available (just expensive).
  - Has very severe stability implications.
- One thread per element.
  - No communication between elements is necessary until assembly stage.
  - More threads in flight means more opportunity to hide latency.
  - One thread per quadrature point possible, but more complicated and results in more communication or over-computation.
- Use texture to read thread input values (e.g, node positions, displacement, etc).
  - Texture has a cache to lower latency
  - Using texture removes need to do coalesced reads

Future work

- Exploit capabilities of newer hardware (e.g GeForce GTX 280).
  - Double precision support is available now.
  - Reading/writing coalescing constraints have been relaxed
    * No need to use texture for reading.
    * Potentially merge per-element computation step with assembly step.
- Full end-to-end solution on GPU
  - Perform assembly and and conjugate gradient solve on the GPU.
  - Should be much faster due to
    * No data transfer between CPU and GPU
    * GPU implementation of CG solve should be faster than CPU
- Extend to larger problems
  - Multi-GPU and multi-system implementations.
  - Double precision support allows larger systems to be stable.

The End