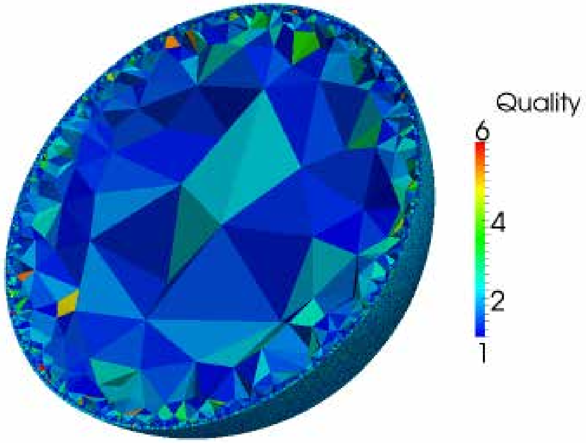
# Parallel Optimization of Meshes on Heterogeneous Computing Systems




Eric Shaffer

Department of Computer Science
University of Illinois at Urbana-Champaign

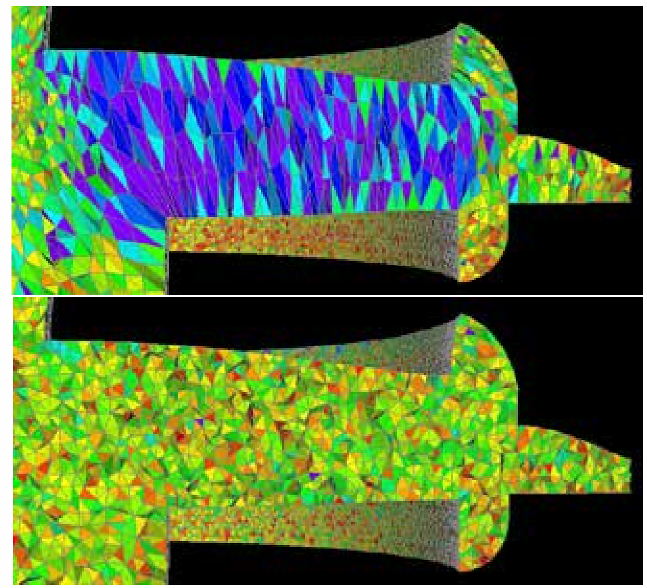
# Collaborators

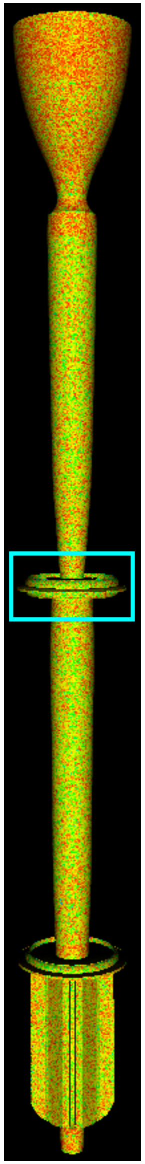
**Work done with**

- **Zuofu Cheng (Illinois)**
- **Raine Yeh (Purdue)**
- **George Zagaris (LLNL)**
- **Luke Olson (Illinois)**

# Mesh Quality

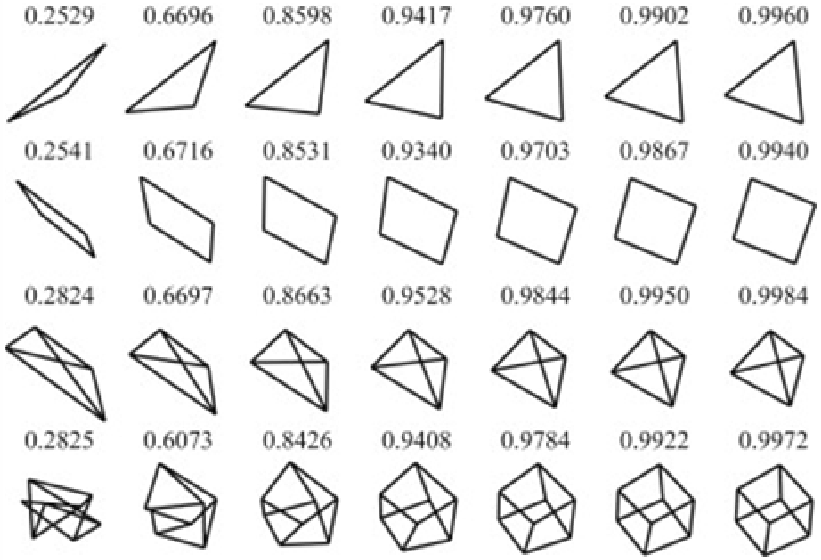
- **Mesh quality is a key concern in engineering simulations**
- **Mesh element shape impacts efficiency and accuracy**
- **Mesh optimization seeks to improve mesh quality**
- Image from DOE ASC Center for Simulation of Advanced Rockets (1997-2007)





# Measuring Mesh Quality

- Lots of metrics
- Some measure actual error…best approach
  - e.g. adjoint error estimation
- Measuring actual error is hard
  - Computationally expensive
  - Generally not solver-independent
- …alternative is to base quality on element geometry
  - mean ratio
  - dihedral angle



# Quality Metric: Inverse Mean Ratio (IMR)

- For tetrahedral elements, assume equilateral is ideal
- Given an element with vertices (a, b , c, d) we form a 3x3 matrix A of edges and a 3x3 matrix W representing the ideal element

$$A = \begin{bmatrix} b-a \\ c-a \\ d-a \end{bmatrix}$$

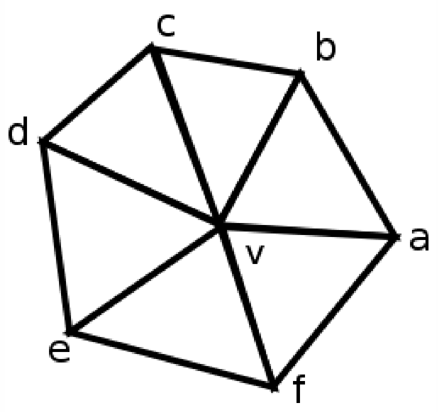
$$W = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{6} \\ 0 & 0 & \frac{\sqrt{2}}{3} \end{bmatrix}$$

# Inverse Mean Ratio (IMR)

- Inverse Mean Ratio is then
$$\frac{\left\|AW^{-1}\right\|_F^2}{3\left|\det(AW^{-1})\right|^{\frac{2}{3}}}$$
- $AW^{-1}$ is identity when A=W
- If element is just ideal scaled, $AW^{-1}$ is the scaling factor
- Metric is invariant to scaling rotation and translation
- Values range from 1 to ∞
- Big is bad

# Optimization of Tetrahedral Meshes: Our Approach

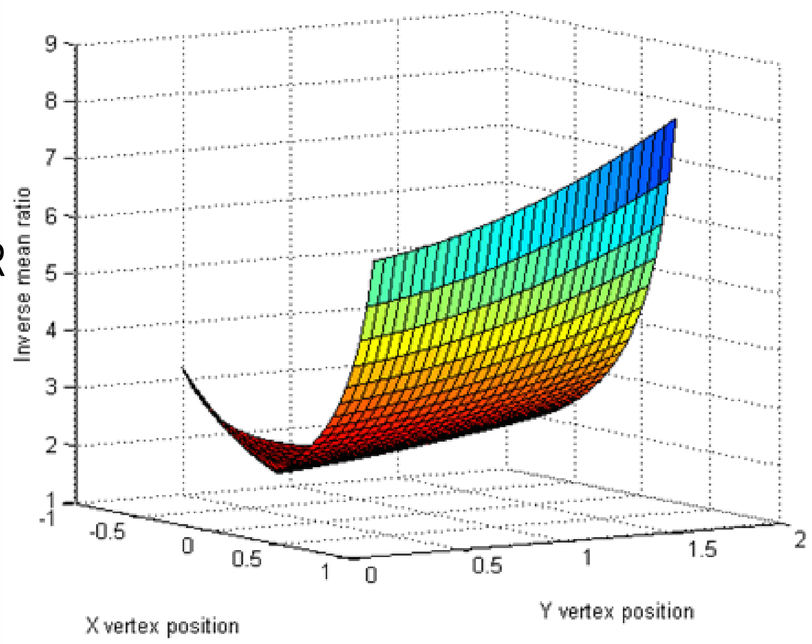
- **Optimize each interior vertex *locally***
  - Generate a position to get min max IMR in 1-ring of elements
  - Taken from Freitag, Jones, Plassmann [1999]




- **Can apply similar strategy to surface vertices**
  - Move a vertex to optimize IMR within the one-ring
  - **Constrain the search space**
    - Keep vertex in tangent plane to the mesh
      - Maintains fidelity to original shape (more or less)
      - Also allows us to use 2D optimization instead of 3D
  - **Further constrain movement to avoid**
    - element inversion
    - fold-over on surface
- **Note that *global* mesh quality never decreases**

# Min-Max Optimization is Non-Smooth

- **Worst element is the limiting condition**
- **Optimization proceeds by minimizing maximum IMR**
  - Shift vertex positions to achieve lower IMR
- **Worst quality element shifts from one to another as optimization proceeds…**
  - Max IMR is a non-smooth function
- **Example: two triangular elements sharing a vertex**



# Optimizing a Non-Smooth Function

- **Ignore the problem and hope space is smooth enough to find a good solution**
  - e.g. use gradient descent method
- **Use a derivative-free method**
  - e.g. Pattern Search [1] or Nelder-Mead
- **Change quality metric**
  - e.g. use average instead of worst [2]

[1] Lori Freitag, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of two optimization methods for mesh quality improvement. In Proceedings, 11th International Meshing Roundtable, pages 29–40, September 2002.

[2] Lori Freitag, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of inexact newton and coordinate descent mesh optimization techniques. In Proceedings of the 13th International Meshing Roundtable, pages 243–254, Williamsburg, VA, September 2004.

# One Research Goal: Compare Numerical Optimization Methods

- **Tried several different numerical methods**
  - Goal was to determine which worked best….
  - Gradient Descent
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
  - Nelder-Mead simplex method
    - Derivative-free method
- **Gradient-based methods require derivatives**
  - You can estimate them numerically
  - Or…you could compute it analytically at a point

# Gradient Descent

Gradient Descent

- Uses the negative of the function gradient as the search direction.
- We use a central difference approximation $\delta_h[f](x) = f\left(x + \frac{1}{2}h\right) - f\left(x - \frac{1}{2}h\right)$ to estimate the gradient.
- Pick direction as $\mathbf{p}_{\mathrm{k}} = -\nabla f(x_k)$, perform line search.
- Parameters are step size and line search density (we cannot make any assumptions due to non-smooth nature of problem, so we have to sample).

# BFGS

Broyden-Fletcher-Goldfarb-Shanno (BFGS)

At step k, the search direction $\mathbf{p}_\mathrm{k}$ is solved using:

$$\mathbf{B}_\mathrm{k}\mathbf{p}_\mathrm{k} = -\nabla f(x_k)$$

and $\mathbf{B}_\mathrm{k}$, the approximate Hessian, is updated at each step with

$\mathbf{B}_{\mathrm{k+1}} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}$ where $\boldsymbol{s}_k = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$

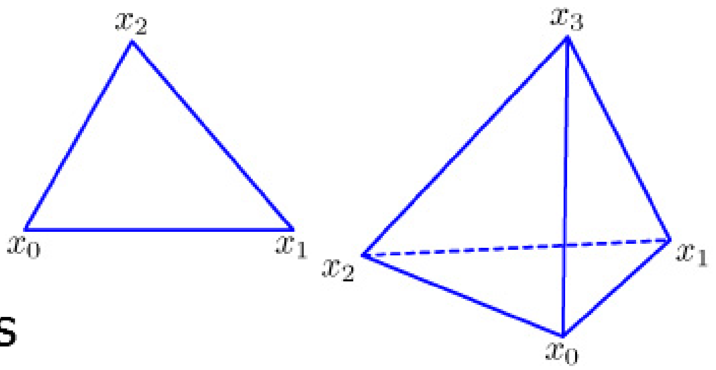
and $\boldsymbol{y}_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

Shown to have good performance even for non-smooth optimizations [4].

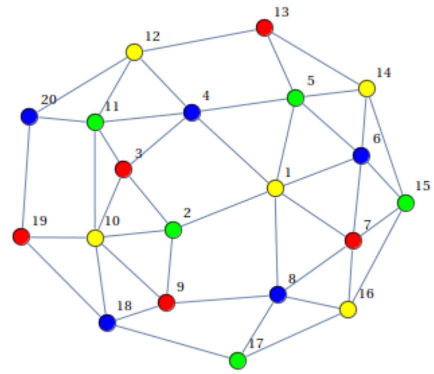
[4] A.S. Lewis and M.L. Overton. Nonsmooth optimization via BFGS. Submitted to SIAM Journal of Optimization, 2009.

# Nelder Mead

- Optimization is evaluated across simplex: convex hull of n+1 vertices for n-dimensional problem.
- For tetrahedral mesh, simplex is also a tetrahedron (not to be confused with the elements of the mesh).
- Performs a series of transformations of simplex to decrease function value at vertices.
- Terminate when function value is small enough (early termination), when simplex is small enough, or when function value at simplex points are close enough.
- Derivative free.



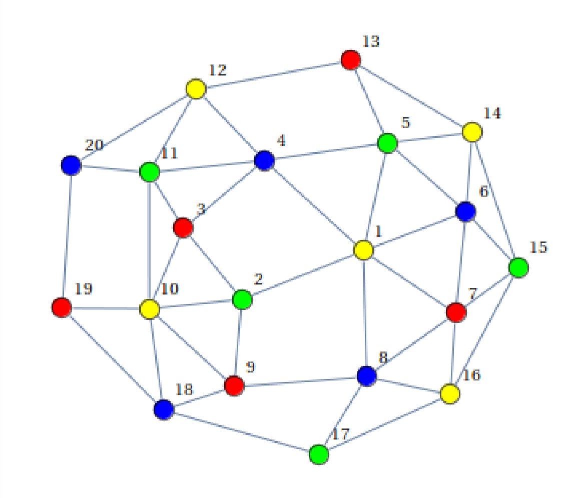
# Parallelization




- **Want to use all processors available on a system**
- **Per-thread optimization of a vertex position**
  - Use both CPU and GPU cores
  - Local approach exposes fine-grained parallelism
- **Cannot simultaneously optimize neighboring vertices**
- **Any independent set of vertices can be optimized in parallel**
  - Use graph-coloring heuristic First Fit to create independent sets
- **First Fit**
  - Serial, greedy coloring of vertices…
  - Colors are integer labels
  - For each vertex and assign lowest integer label not used on a neighbor
  - Optimal coloring minimizing number of sets is NP-Hard

# Issues with First-Fit Coloring

- **It is the bottleneck in terms of scalability**
  - For large meshes non-locality causes bad memory access pattern
  - There are distributed/parallel algorithms that we have not explored
  - Advice would be welcomed….
- **Different orderings of vertex optimizations produce different final qualities**
  - Not clear how (or if) coloring could be biased to produced better orderings



# Load Balancing

- **Which vertices are optimized on the CPU vs. GPU**
- **Current implementation uses an admittedly poor heuristic**
  - Surface optimized on CPU and volume on GPU
- **Why?**
  - Surface is a 2D object…should generate smaller sets
  - Smaller sets would hide GPU latency less well
  - Architecture of current software made it the easiest approach
- **Better approach would be to use a threshold size for a set**
  - Determine threshold based on bus latency estimate

# GPU Implementation (Fermi)

Nelder-mead uses too many registers (capped at 63 in pre-K20 systems, limiting occupacy to 33%).

Instead, use GPU as a streaming processor, cache entire neighborhood in shared memory.

Entire shared memory (48KB) is consumed by 1 block, so only 1 block can occupy SM. Each vertex is float3 + index = 16 bytes. Connectivity table is also stored in shared memory.

Typically, can still use 64 thread blocks (on some high connectivity meshes, only 32 thread blocks are possible).

Occupancy goes way down (4%), but paradoxically, performance is increased by 25%.

# GPU Implementation (Kepler SMX)

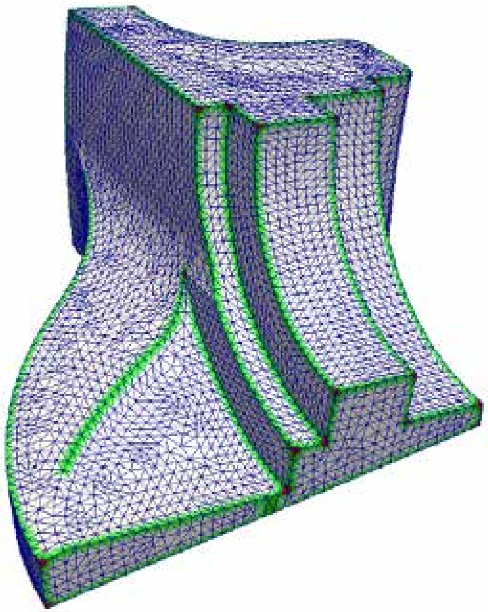
- **Higher number of registers per-thread**
  - Register spill is avoided
- **Speedup of 2.5 over Fermi class**
- **On either class of hardware**
  - No register spill for BFGS or Gradient Descent

# Sidenote: Surface Mesh Feature Preservation

- **Medial Quadric suggested by Jiao and Bayyana [2008]**
- **Normal tensor for a vertex v**

  $$\mathbf{M} = \sum_i w_i \mathbf{n}_i \mathbf{n}_i^{\mathrm{T}}$$

  - Sum of area weighting outer-product of face normal around v
- **Eigenvalues of tensor classify vertices**
  - smooth (three distinct eigenvalues)
  - ridge (two distinct eigenvalues)
  - corner (one distinct eigenvalue)
- **Computation is local and parallelizable**
- **On ridges optimization can be done by golden section search**
  - Convergence guarantee for unimodal functions



# Experimental Setup

- **Xeon X5650 CPU, 6 cores at 2.67 GHz, 16GB main memory**
  - C2050 GPU (Fermi class) 448 cores and 2.6GB memory
  - GTX Titan 2688 cores and 6GB memory
- **OpenMP+CUDA**

# Results: Quality Comparison

**Table 1** Quality of optimized meshes compared to original

| Mesh | Number of Elements | Maximum IMR after No Optimization | Quality Interior Only Optimization | Quality Full Optimization |
|---|---|---|---|---|
| Small Rocket | 468,623 | 2.229 | 1.992 | 1.84 |
| Big Sphere | 4,720,255 | 8.645 | 5.813 | 3.705 |
| Big Rocket | 14,992,367 | 14.971 | 5.0897 | 3.566 |

**Table 2** Quality of combined optimization with different methods on GTX Titan (normalized to GPU time of 100 iterations of Nelder-Mead)

| Mesh | Number of Elements | Quality Nelder-Mead | Quality BFGS | Quality Gradient Descent |
|---|---|---|---|---|
| Small Rocket | 468,623 | 1.84 | 1.99 | 1.99 |
| Big Sphere | 4,720,255 | 3.705 | 4.34 | 4.44 |
| Big Rocket | 14,992,367 | 3.566 | 3.93 | 3.84 |

# Results: Speedup over serial

**Table 3** Performance comparison of parallel methods with serial

(a) Speedup over serial of interior vertex optimization

| Mesh | Interior Vertices | Serial (s) | OpenMP (Speedup) | C2050 (Speedup) | GTX Titan (Speedup) |
|---|---|---|---|---|---|
| Small Rocket | 58,981 | 60.7 | 8.4 | 8.0 | 14.0 |
| Big Sphere | 290,739 | 571.1 | 7.1 | 9.4 | 21.5 |
| Big Rocket | 2,202,793 | 1967.2 | 7.7 | 5.9 | 16.4 |

(b) Speedup over serial of surface vertex optimization

| Mesh | Surface Vertices | Serial (s) | OpenMP (Speedup) |
|---|---|---|---|
| Small Rocket | 38,673 | 16.0 | 3.1 |
| Big Sphere | 1,048,578 | 341.1 | 2.5 |
| Big Rocket | 576,688 | 381.5 | 4.1 |

(c) Speedup over serial of combined optimization (GTX Titan)

| Elements | Total Vertices | Serial (s) | Parallel CPU+GPU (Speedup) |
|---|---|---|---|
| 468,623 | 97,654 | 76.8 | 14.6 |
| 4,720,255 | 1,339,317 | 912.2 | 10.9 |
| 14,992,367 | 2,779.481 | 2348.7 | 15.7 |