Efficient computation of distances between FEM solutions defined on different 3D meshes

CHANGE workshop, Leysin February 2018

Maxence Reberol, Bruno Lévy
ALICE team, Inria Grand-Est
Distance between FEM solutions

Subject of the talk:
How to compute $\| f_h - g_h \|_{L^2} = \sqrt{\int_\Omega (f_h - g_h)^2}$?
Motivations

How to evaluate FEM accuracy and performance? (influence of meshes, refinement, order, etc)

1. If analytical solution is known:
   - compute error ($L^2$, $H^1$, ..) with quadratures
   - problems can be built with **Method of Manufactured Solutions**:
     
     a) choose analytical solution
     \[ u(x, y) = \sin(2\pi x) \sin(2\pi y) \]
     
     b) inject in problem (domain + PDE + BCs)
     \[-\Delta u = f \]
     
     c) derive formula for source term and BCs
     \[ f(x, y) = 4\pi^2 \sin(2\pi x) \sin(2\pi y) \]

[Salari00, Roache02]
Motivations: evaluation of FEM accuracy

1. If analytical solution, convergence analysis is easy:
Motivations: evaluation of FEM accuracy

1. If analytical solution, convergence analysis is easy..
   but results are not representative of real-life performance or accuracy:

   - very simple domains (unit cube usually)

   - analytical RHS everywhere, no propagation from boundaries (contrary to real-life problems where RHS is null/constant)

   - measuring source term approximation (or coefficients, BCs, etc)
     e.g. $-\Delta u = f$, $f$ not in the approximation space
Motivations: evaluation of FEM accuracy

2. If no analytical solution, use a reference solution

For specific applications, compare relevant quantities:
- maximum stress (mechanics), drag coefficient (aerodynamic),
  eigenvalues, etc

For general purpose, how to compute (L^2, H^1,..) errors ?
- approximate error with distance to reference solution

New question: how to compute distances ?
Distance computation

Integral replaced by weighted sum:

\[(||f_h - g_h||_{L^2})^2 = \int_\Omega (f_h - g_h)^2 \approx \sum_{i=1}^{N} w_i (f_h(x_i) - g_h(x_i))^2\]

Proper way: it's FEM, so let's use quadratures (requires mesh and local interpolation)

3D with curved cells (hexahedra, quadratic tets, etc) ?

Bibliography:
- supermesh
- rendezvous mesh
- intersection mesh
Distance computation with regular sampling

Computer graphics approach:

- sampling of both fields on a (large) regular voxel grid (typical size: $1000^3$)
- slice by slice (low memory consumption)

(slice of a FEM field)
Distance computation: global algorithm

1. Initialization
   (upload data to GPU)

2. For each slice:
   - render field A
   - render field B
   - compute difference
   - store contribution to distance

3. Combine contributions to get global distance
Computing the field values

FEM interpolation is defined per cell:
- **mapping** from reference element to world space
- **interpolation** defined in reference element

At each pixel:

\[ f_h(x) = \hat{f}_K \circ \mathcal{M}_K^{-1}(x) \text{ for } x \in K \]
Computing the field values

\[ f_h(x) = \hat{f}_K \circ \mathcal{M}_K^{-1}(x) \text{ for } x \in K \]

- reference coordinates at triangle vertices with marching tetrahedra [AK91]

- rasterization for linear interpolation at pixel centers

no mapping inversion exact for linear mappings (tetrahedra)

shape functions evaluated exactly in the fragment shader

(pixel exact rendering e.g. [BH04, NHK11])
Curved (non-affine) elements

\[ f_h(x) = \hat{f}_K \circ \mathcal{M}_K^{-1}(x) \quad \text{for} \quad x \in K \]

\( \mathcal{M}_K \) is approximated by a piecewise-linear mapping

i.e. curved geometry approximated by subdivision reference coordinates, used in evaluation, are no longer exact (done on the GPU using the \textit{instance rendering} feature of OpenGL)
Summary of the OpenGL rendering

- Vertex shader: mapping decomposition if curved (mesh coefficients via VertexAttributes)

- Geometry shader: marching tetrahedra

- Rasterization (OpenGL): linear interpolation of ref. coords.

- Fragment shader: shape functions evaluation (field coefficients via SSBO)
Parameter sensitivity

- # samples (voxel grid resolution):

(10-100 millions samples inside at worst)

- approximation of curved cells:

(subdivision required !)
Parameter sensitivity

- voxel grid orientation (two examples):

(OK after a few millions of samples, base orientation is usually good)
Validation

- linear elasticity problem with analytical solution (built with MMS)
- exact errors computed with high-order quadratures (dotted lines)
- distances to reference solution (fine mesh with P4) for various meshes and orders
Validation
- linear elasticity problem with analytical solution (built with MMS)
- exact errors computed with high-order quadratures (dotted lines)
- distances to reference solution (fine mesh with P4) for various meshes and orders

![Graph showing the relationship between relative $L^2$-distances to reference solution and the cube root of degrees of freedom, with different orders denoted by P1, P2, P3, Q1, Q2, Q3, and quadratures. The graph highlights insufficient hex decomposition.]
Performance

Less than one second for standard meshes (~ 1 million cells) (timings obtained with Nvidia 1080 GTX)

<table>
<thead>
<tr>
<th>model</th>
<th>dim</th>
<th># cells</th>
<th>basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1: joint (fig. 5.3)</td>
<td>1</td>
<td>10k</td>
<td>$P_2$ (tet)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>65k</td>
<td>$P_1$ (tet)</td>
</tr>
<tr>
<td>#2: hanger (fig. 5.10)</td>
<td>3</td>
<td>1,739k</td>
<td>$P_1$ (tet)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>343k</td>
<td>$P_2$ (tet)</td>
</tr>
<tr>
<td>#3: hanger (fig. 5.10)</td>
<td>3</td>
<td>36k</td>
<td>$Q_2$ (hex)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>290k</td>
<td>$Q_1$ (hex)</td>
</tr>
<tr>
<td>#4: carter (fig. 5.12)</td>
<td>1</td>
<td>1,210k</td>
<td>$P_1$ (tet)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>373k</td>
<td>$P_2$ (tet)</td>
</tr>
<tr>
<td>#5: 747 (fig. 5.14)</td>
<td>1</td>
<td>1,385k</td>
<td>$P_1$ (tet)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>577k</td>
<td>$P_2$ (tet)</td>
</tr>
<tr>
<td>#6: 40heads (fig. 5.14)</td>
<td>1</td>
<td>2,905k</td>
<td>$P_1$ (tet)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,350k</td>
<td>$P_2$ (tet)</td>
</tr>
</tbody>
</table>
Performance on large fields

Field A: 2905k tets P5, 64M dofs
Field B: 2305k tets P7, 146M dofs
sparse model structure: ~20% of bbox

convergence: ~ 1 minute
Visualization of slice difference

Visualization is free (textures are available on the GPU memory) convenient for debugging or investigating FEM behavior
Visualization of slice difference

Difference: Hex-Tet P2/Q1 vs P1
Application: convergence analysis on 3D models

Rel. $L^2$ error vs $1/h_{max}$

Rel. $L^2$ error vs solver time (s)
Application: convergence analysis on 3D models

![Graphs showing convergence analysis results for different 3D models.](image)

Rel. $L^2$ error vs. $1/h_{max}$ for different finite element approximations:
- $P_1$
- $P_2$
- $Q_1$
- $Q_2$
- $Q_1$-octree
- $Q_2$-octree
- $Hyb_1$
- $Hyb_2$

Solver time (s) vs. $1/h_{max}$ for different approximations.
More information

- Article (detailed explanation):
- Software: github.com/mxncr/FFES, file format:

```json
{
  "version": "0.1",
  "groups": [
    {
      "mesh dim": <int: dimension of the mesh, should be 3>,
      "primitive": <string: TET or HEX, other decompositions not implemented>,
      "nb mesh cp_per cell": <int: nb of control points per cell in the mesh>,
      "field dim": <int: dimension of the field, e.g. 1 for scalar field>,
      "nb field cp_per cell": <int: nb of control points per cell in the field>,
      "interpolation": <string: interpolation function>,
      "field ctrl points": <string: Base64 encoding of field coefficient>
    },
    {
      // another group definition if multiple ElementGroups
    }
  ],
  "arbitrary_key": <string: other metadata can be stored in this json, they are ignored>
}
```

- Limitations: OpenGL 4.2., memory, 32bits float rasterization

| vec3 element_mapping(const vec3 ref_pos, const vec3 values[4]){
| vec3 result =
| (1.0f-ref_pos.x-ref_pos.y-ref_pos.z) * values[0] + ref_pos.x + ref_pos.y + ref_pos.z
| return result; } |

Mapping and interpolation included in file format

Support for arbitrary mapping/interpolation: write the GLSL functions in the input file
Conclusion

- A fast (real-time) and flexible approach to compute distances between FEM solutions

- Usage: increase parameters (samples, subdivision) to verify convergence of the distance computation

Perspectives

- Integration into Graphite/Geogram for better interactive visualization

- More efficient curved cell mappings (Newton-like correction)

- CPU implementation for portability
Thank you for your attention

Questions?
References

[Salari00] Salari, Kambiz and Knupp, Patrick

Code Verification by the Method of Manufactured Solutions, 2000

[Roache02] Roache, Patrick J.

Code Verification by the Method of Manufactured Solutions, 2002

[BH04] Brasher, M. and Haimes, R.

Rendering planar cuts through quadratic and cubic finite elements, 2004


GPU-Based Interactive Cut-Surface Extraction From High-Order Finite Element Fields

[AK91] Akio, Doi and Koide, Akio

An efficient method of triangulating equi-valued surfaces by using tetrahedral cells, 1991