ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics laboratory

Physically-Based Simulation in CG

Exercise 2: Finite Element Method

(Assigned: October 11; Due: October 25)

How to submit the exercise

Send your version of the files `FEMElementTri.cpp` and `SimpleFEM.cpp` by email to Byungsoo Kim (kimby@inf.ethz.ch), along with your names, by October 25 (midnight).

Introduction

In this exercise you will complete a FEM solver for a two-dimensional Poisson and Laplace problem

$$-\Delta\, \boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y}) \tag{1}$$

on the unit square domain $\Omega = [0,1] \times [0,1]$.
For the Poisson problem, boundary conditions are given as $u(x,y) = 3x^2 + 2xy^3, (x,y) \in \partial\Omega$ (which is also the analytical solution) and the source term as $f(x,y) = -6 - 12xy$.
For the Laplace problem, boundary conditions are given as $u(x,y) = e^x \sin(y), (x,y) \in \partial\Omega$ (which is also the analytical solution) and the source term as $f(x,y) = 0$.

The corresponding weak form of the problem is

$$\int_\Omega \nabla u(x,y) \cdot \nabla v(x,y)\, dxdy = \int_\Omega f(x,y) v(x,y) dxdy \quad \forall v(x,y). \tag{2}$$

The FEM discretization takes place on a regular triangular tessellation of the domain. Using the discretizations $u(x,y) \approx \sum_i u_i N_i(x,y)$ and $v(x,y) \approx \sum_i v_i N_i(x,y)$ we can rewrite this as

$$\sum_{e,i,j} u_i v_j \int_{\Omega_e} \nabla N_i(x,y) \cdot \nabla N_j(x,y) dxdy = \sum_{e,j} \int_{\Omega_e} f(x,y) N_j(x,y) dxdy, \quad \forall v(x,y) \tag{3}$$

leading to a linear system for the degrees of freedom u_i.

General specifications

This exercise consists of five tasks: the first four tasks complete the FEM solver by implementing routines that compute and assemble the contributions of a given element to the stiffness matrix and the right hand side. This amounts to evaluating the integrals on the left and right hand sides of equation (3). The fifth task verifies the convergence of the numerical solution.

We provide a CMake-based framework that initializes the computation and uses OpenGL/GLUT for visualization. There are two files to be modified, `FEMElementTri.cpp` and `SimpleFEM.cpp`. Once you have completed all tasks, send us the files and we will compile them with the rest of the framework. Please do not include any external libraries or files! The `main` function is in `SimpleFEM.cpp`.

Task 1: Implement basis function derivatives (25%)

```
void FEMElementTri::computeSingleBasisDerivGlobalLES(int nodeId,
Vector2 &basisDerivGlobal, const FEMMesh *pMesh) const
```

In order to compute the entries of the stiffness matrix *K*, you first have to compute the partial derivatives of the basis function with respect to *x* and *y*. This function should compute the derivatives (`basisDerivGlobal`) for a given node (`nodeId`) of a given element (represented by the class).

Hint: use the following properties

1) The basis functions are linear, i.e., $N_i(u,v) = a_i\, x + b_i y + c_i$
2) The basis functions satisfy $N_i(\boldsymbol{x}_j) = \delta_{ij}$

Properties 1) and 2) can be translated into a system of 3 linear equations for the 3 unknown coefficients of the basis function for node `nodeId`.

Use the `Matrix3x3` class and its function `Matrix3x3::inverse()` to assemble and solve the system.

Task 2: Geometric construction of global basis function derivatives (15%)

```
void FEMElementTri::computeSingleBasisDerivGlobalGeom(int nodeId,
Vector2 &basisDerivGlobal, const FEMMesh *pMesh) const
```

For triangular element shapes and linear basis functions there is also a different, geometric approach for computing the basis function derivatives. The gradient for a node is always normal to the opposite edge and its length is the inverse of the triangles height. Implement a method that computes a node's global derivative using this geometric construction. For testing, compare its results to the method implemented in Task 1.

Task 3: Assemble the stiffness matrix (20%)

```
void FEMElementTri::Assemble(FEMMesh *pMesh) const
```

Using the basis function derivatives, compute the element's contribution to the stiffness matrix and add these values to the corresponding entries. To add a value to the (ith, jth) entry of the stiffness matrix, you should use the function

```
pMesh->AddToStiffnessMatrix(i, j, value).
```

Remark: Note that (`i,j`) are global indices, not element-local indices. In order to transform an element-local node number `iLocal` (0<`iLocal`<2) to a global number `iGlobal`, use `iGlobal = GetGlobalNodeForElementNode(iLocal);`

Note also that the matrix is symmetric, and only the lower triangular part of the matrix is actually stored. This means that only entries whose indices (`i,j`) satisfy the condition `i >= j` can to be added to the matrix.

With the stiffness matrix assembled, the next step is to compute the right hand side of the system. However, you can already test your implementation by setting the parameter `laplaceProblem` in `SimpleFEM.cpp` to `true`. This changes the original Poisson problem to a Laplace problem, $-\Delta\, u(x,y) = 0$, for which the right hand side is zero.

Task 4: Assemble the right-hand side (25%)

```
void SimpleFEM::ComputeRHS(const FEMMesh &mesh, vector<double> &rhs)
```

Using the source term function `eval_f(x, y)`, compute the element's contribution to the right hand side and add these values to the corresponding entries. This amounts to evaluating the integrals on the right hand side of (3).

Hint: note that for each element, there are three nonzero integrals $\int_{\Omega_e} f(x,y) N_j(x,y)$, each of which corresponds to one of the element's nodes. Evaluate these integrals using a one point quadrature rule with the quadrature point at the barycenter of the element. Add the resulting values to the corresponding entries of the right hand side vector (see also remark in Task 3)

Task 5: Convergence of the solution (15%)

```
void SimpleFEM::computeError(FEMMesh &mesh, const vector<double>
&sol_num, vector<double> &verror, double &err_nrm)
```

Compare the numerical solution (sol_n) to the analytical solution (sol_a). To this end, start by computing the error vector component-wise as $v_{err} = |sol_n - sol_a|$.

A natural norm for the error vector is the inner product induced by the stiffness matrix K, i.e., $|err| = \sqrt{v_{err}^t \cdot K \cdot v_{err}}$.

Compute the error norms for different grid resolutions (*N=3,5,8,16,32*) and report the values.
Remark: This task should be performed for both the Poisson and Laplace problems. If, however, you did not succeed in task 4, you should still implement and report results for the Laplace problem.

Additional information on the code

You can change between the 2D and 3D view by pressing 'v', and rotate the view using the arrow keys. By pressing 'e', you can visualize the error instead of the value.

There are four parameters at the beginning of file `SimpleFEM.cpp`:

- The parameter `gridSize` determines the side length of a grid cell for regular grids.
- The parameter `gradedMesh` determines whether a graded or a regular mesh will be used

- The parameter `laplaceProblem` determines whether a simple Laplace problem or the Poisson problem (1) is solved (changes $f(x,y)$ and boundary conditions)
- The parameter `debugOut` determines whether additional information should be printed to the console

For code verification, please note that slide 1 of part 1 of the “Applied partial differential equations” lecture is a visualization of the exercise’s solution to the Poisson problem.