Interfacing Dir3 with a Finite Element Code

Alexander P. Rand and Noel J. Walkington

November 3, 2009

1 Introduction

The mesh generator `dir3` reads a `.plc` file[1] representing a region $\Omega \subset \mathbb{R}^3$ and produces a `.tplc` file representing a tetrahedral decomposition of Ω. To assist the development of applications codes which use such meshes we have provided code to read `.tplc` files and iterate over the tetrahedra and boundary triangles in the mesh. More functionality is available from a second code which interfaces directly with the `dir3` libraries. Currently only a minimal subset of the possible geometric predicates available from `dir3` have been exported to this interface. A simple finite element code to solve Poisson's equation is provided as a prototypical application.

2 Running the Finite Element Code

Within the `SimpleFem` directory typing `make poisson` will make the executable file `poisson`. This is a basic finite element code that reads a `.tplc` file and solves Poisson's equation on the triangulated domain. Run the code by typing `poisson Data/pyr.0.200`. The file `./Data/pyr.0.200.tplc` contains data representing a triangulation of a pyramid. The output should be similar to that in Figure 1.

The purpose of this code is to illustrate how the mesh generation capabilities of `dir3` may be integrated within an application code. The finite element technique is very versatile, and we have provided a small suite of options to illustrate

[1] The `.plc` file extension denotes "piecewise linear complex" (plc) and the `.tplc` denotes a "triangulated plc".

```
bash-3.2$ ./poisson Data/pyr.0.200
Sample finite element code:
Example 0: Laplaces Equation
Solution of 3d Poisson equation
Initializing dir3Mesh(Data/pyr.0.200): (reads .tplc file)
Number of vertices read in = ..................... 452
Number of boundary triangles = ................... 634
Number of tetrahedra = ........................... 1728
Initializing feMesh()
Number of degrees of freedom = ................... 2948
Leaving constructor
Number of basis functions per element = .......... 10
Number of unknowns = ............................. 2948
Half band width = ................................ 2723
Time to initialize mesh and matrix classes ....... 0.296
Number of elements = ............................. 1728
Time to assemble element matricies ............... 0.453
Number of boundary elements = .................... 634
Time to assemble boundary element matrices ....... 0.015
Time to solve linear equations ................... 66.894

                  L-2 norm      H-1 semi-norm     H-1  norm
Exact Solution   3.902742e+00   1.662577e+01     1.707769e+01
F. E. Solution   3.903708e+00   1.662201e+01     1.707426e+01
Errors           1.219720e-02   5.371590e-01     5.372974e-01
Time to compute norms ............................ 0.203
Total CPU time ................................... 67.861
~bandMV() serial banded matrix class
bash-3.2$
```

Figure 1: Output from `poisson Data/pyr.0.200`

various levels of integration a user may wish to work with. Manipulating the definitions at the top of `poisson.h` toggles the following options.

1. **Different Solutions:** Two solutions of Poisson's equation, $-\Delta u = f$ have been coded.

   $$u(x,y,z) = \cos\left(\pi x/\sqrt{2}\right)\exp\left(\pi y\right)\sin\left(\pi z/\sqrt{2}\right), \quad \text{with} \quad f(x,y,z) = 0,$$

   is the default (zeroth) solution and explicitly selected with the second command line argument, `poisson Data/pyr.0.200 0`. The other (first) solution,

   $$u(x,y,z) = x^2 + 2y^2 \quad \text{with} \quad f(x,y,z) = -6,$$

   and is selected from the command line using `poisson Data/pyr.0.200 1`.

   These functions are declared at the end of `poisson.h` and coded in `examples3d.C`; additional solutions are easily added. The function $u(x,y,z)$ is used to specify Dirichlet boundary and the normal derivative required for Neumann boundary data is computed using $\nabla u.n$. Only boundary values are required for the finite element solutions. In order to compute the $L^2(\Omega)$ and $H^1(\Omega)$ errors u and its gradient, ∇u, are required everywhere.

2. `dir3` **Interface:** The lines

   ```
   #define TPLC
   // #define PLC
   ```

   determine the interface with the mesh generator. With the `TPLC` option the finite element code reads a `.tplc` file to obtain the mesh using the interface provided in `./Geometry/dir3TPLC.C`.

   The `PLC` option uses the library functions provided by `dir3`. In this case a `.plc` and configuration file `.cfg` are read and `dir3` generates the mesh. This interface has the capability to access all of the features available from `dir3`. Currently the only additional feature implemented is point location. However, `dir3` implements a full suite of iterators and geometric and topological predicates, all of which would be easy to access through the interface provided in `./Geometry/dir3TPLC.C`.

   Using the `PLC` requires the `makefile` is set up to find the correct directory containing the `dir3` library. The relevant lines of the `makefile` are given below.

```
DIR3_INC  =
DIR3_LIB  =
#DIR3_DIR  =  $(HOME)/dir3-0.9
#DIR3_INC  =  -I$(DIR3_DIR)/inc
#DIR3_LIB  =  $(DIR3_DIR)/lib/dir3.a
```

The first two lines are appropriate for using the `TPLC` option while the final three are needed for the `PLC` option. The `DIR3_DIR` variable should be set link to the `dir3` code in your environment.

3. **Linear or Quadratic Elements:** Classical linear or quadratic Lagrange finite elements may be selected with the lines

```
//#define LinearElements
#define QuadraticElements
```

Running the code with linear elements will result in a less accurate solution.

4. **Two or Three Dimensions:** Aside from the basis functions, the structure of a finite element code is independent of dimension. To illustrate this, an interface which reads the files produced by the two dimensional mesh generation code `triangle` is provided in `./Geometry/triangleMesh.C`. The lines

```
#define NDIM 3
// #define NDIM 2
```

in `poisson.h` are used to select two or three dimensions. The default solution in two dimensions is

$$u(x,y) = \cos(\pi x)\exp(\pi y), \qquad \text{with} \qquad f(x,y) = 0.$$

The files `./Data/mesh6.1.*` and `./Data/mesh6bl.1.*` were produced by `triangle`. Executing `poisson Data/mesh6.1` will solve the corresponding Poisson's problem.

5. **Linear Solver:** The finite element methodology gives rise to a large sparse system of linear equations and there are many specialized packages to efficiently solve this class of problems. The linear solver provided is a

banded Gauss elimination routine. This is simple, robust, but slow. Also, the mesh generation routines do not order the vertices to minimize the band width of the linear system so bandwidth (and hence memory required) is unnecessarily large. (All efficient direct linear solvers internally reorder the variables to minimize bandwidth and/or fill.)

Interfaces to the PETSC iterative and Spooles direct solvers are provided. These are selected from poisson.h with the lines

```
#define BandMatrix
//#define SpoolesMatrix
//#define PetscMatrix
```

Aside from having these packages installed on your system, to use these solvers it is necessary to modify the makefile to point to their location.

(a) PETSC **Interface:** PETSC is a package of iterative solvers which will run on serial and parallel machines. The PETSC installation process should set the following environment variables:

```
noelw@noelwpc:~> export | grep PETSC
declare -x PETSC_ARCH="linux-gnu-opt"
declare -x PETSC_DIR="/usr/local/petsc-2.3.3-p4"
```

The following lines in the makefile then access the *serial version* of the PETSC package.

```
#PETSC 2.3
include ${PETSC_DIR}/bmake/common/base
#PETSC 3.0
#include ${PETSC_DIR}/conf/base
```

and

```
poisson: $(femFiles)
# banded matrix
# g++ $(CFLAGS) -o poisson poisson.C $(BOOST_INC) $(DIR3_INC) $(DIR3_LIB) -lpth
# spooles
# g++ $(CFLAGS) -o poisson poisson.C $(BOOST_INC) $(DIR3_INC) $(DIR3_LIB) $(SPO
#PETSC 2.3
$(CLINKER) ${PETSC_INCLUDE} -o poisson poisson.C $(BOOST_INC) $(DIR3_INC) $(DIR
#PETSC 3.0
```

```
# g++ -O ${PETSC_INCLUDE} $(BOOST_INC) $(DIR3_INC) -o poisson poisson.C $(DIR3_
```

The code in `./LinearAlgebra/petscMatrix.C` solves the linear system using an ILU preconditioned Krylov subspace iterative solver. This package can easily solve the Poisson problem with a million variables on a modest laptop.

(b) `Spooles` **Interface:** The `Spooles` package is a sparse direct solver which uses nested dissection. While it was designed to be compiled in `C`, it can be compiled with the `g++` compiler by following the directions documented at the beginning of the file in `./LinearAlgebra/spoolesMatrix.C`. Granted this, the following lines in the `makefile` will access this solver.

```
SPOOLES_DIR=$(HOME)/Spooles
SPOOLES_I = -I$(SPOOLES_DIR)/LinSol
SPOOLES_A = $(SPOOLES_DIR)/LinSol/srcMT/BridgeMT.a $(SPOOLES_DIR)/LinSol/srcST/
```

where the first line must point to the directory where `Spooles` is installed. Then the line to build `poisson` with this solver is

```
poisson: $(femFiles)
# banded matrix
# g++ $(CFLAGS) -o poisson poisson.C $(BOOST_INC) $(DIR3_INC) $(DIR3_LIB) -lpth
# spooles
g++ $(CFLAGS) -o poisson poisson.C $(BOOST_INC) $(DIR3_INC) $(DIR3_LIB) $(SPOOL
#PETSC 2.3
# $(CLINKER) ${PETSC_INCLUDE} -o poisson poisson.C $(BOOST_INC) $(DIR3_INC) $(D
#PETSC 3.0
# g++ -O ${PETSC_INCLUDE} $(BOOST_INC) $(DIR3_INC) -o poisson poisson.C $(DIR3_
```