include “functions.h”

using namespace Eigen;

void fe_electroStatics_normal(double time) {

```
MatrixXd* nodes = mesh[0].getNewNodesPointer();
MatrixXi* elements = mesh[0].getNewElementsPointer();

int nel = mesh[0].getNumElements();
int nnode = mesh[0].getNumNodes();
int nnel = mesh[0].getNumNodesPerElement();
int sdof = nnode;

MatrixXd electrical_kk = MatrixXd::Zero(nnode, nnode);

MatrixXd electrical_kk_element = MatrixXd::Zero(nnel, nnel);
VectorXd electrical_force = VectorXd::Zero(nnode);
VectorXd electrical_force_element = VectorXd::Zero(nnel);
VectorXd VP = VectorXd::Zero(nnode);
VectorXd I = VectorXd::Zero(nel);

std::cout << "Debugging in Electrostatics 1" << "\n";

// Element Data
VectorXd xcoord      = VectorXd::Zero(nnel);
VectorXd ycoord      = VectorXd::Zero(nnel);
VectorXd zcoord      = VectorXd::Zero(nnel);

//fe_apply_bc_current(I, time);

for (int i = 0; i < nel; i++) {

    for (int j = 0; j < nnel; j++) {
        int g = (*elements)(i, j + 2);
        xcoord(j)      = (*nodes)(g, 1);
        ycoord(j)      = (*nodes)(g, 2);
        zcoord(j)      = (*nodes)(g, 3);
    }

    //VectorXd I_element = VectorXd::Zero(nnel);
    //fe_gather_pbr(I, I_element, (*elements).block<1, 8>(i, 2), sdof);

    int nglx = 2;
    int ngly = 2;
    int nglz = 2;
```

```
VectorXd points  = guass_points(nglx);
VectorXd weights = guass_weights(nglx);

VectorXd dndr(nnel);
VectorXd dnds(nnel);
VectorXd dndt(nnel);
VectorXd dndx(nnel);
VectorXd dndy(nnel);
VectorXd dndz(nnel);
MatrixXd jacobian(ndof, ndof);
MatrixXd invJacobian(ndof, ndof);
VectorXd shapes(nnel);

MatrixXd conductivity = 0.1 * MatrixXd::Identity(ndof, ndof);
MatrixXd electrical_shape_mat = MatrixXd(nnode, ndof);

for (int intx = 0; intx < nglx; intx++) {
    double x = points(intx);
    double wtx = weights(intx);
    for (int inty = 0; inty < ngly; inty++) {
        double y = points(inty);
        double wty = weights(inty);
        for (int intz = 0; intz < nglz; intz++) {
            double z = points(intz);
            double wtz = weights(intz);

            fe_dniso_8(dndr, dnds, dndt, x, y, z);
            jacobian = fe_calJacobian(ndof, nnel, dndr, dnds, dndt, xcoord, ycoord, zcoo
            double detJacobian = jacobian.determinant();
            invJacobian = jacobian.inverse();
            fe_dndx_8_pbr(dndx, nnel, dndr, dnds, dndt, invJacobian);
            fe_dndy_8_pbr(dndy, nnel, dndr, dnds, dndt, invJacobian);
            fe_dndz_8_pbr(dndz, nnel, dndr, dnds, dndt, invJacobian);
            fe_electrical_shapeMatrix(electrical_shape_mat, dndx, dndy, dndz);

            electrical_kk_element = electrical_kk_element + (electrical_shape_mat * cond

            shapes = fe_shapes_8(x, y, z);
            electrical_force_element = electrical_force_element + (I(i) * shapes * wtx *
        }
    }
}

fe_assemble_electricStiffness(electrical_kk, electrical_kk_element, (*elements).block<1,
fe_scatter_electricalForce(electrical_force, electrical_force_element, (*elements).block
```

```
}

fe_apply_bc_potential(electrical_kk, electrical_force, time);
VP = electrical_kk.inverse() * electrical_force ;

mesh[0].readNodalElectroPhysics(VP);
fe_vtuWrite(0, 0, mesh[0]);

nodes = NULL;
elements = NULL;
}
```