AE 5301 – Finite Element Methods in Fluid Mechanics and Heat Transfer

Final Project Report

James Grisham

May 2015

1 Introduction

The purpose of this project was to develop a general three-dimensional elliptic solver using the Galerkin finite element method. The final code is templated, object-oriented C++ and is compatable with all element types (i.e., tetrahedra, prisms, pyramids, and hexahedra). It is also able to handle non-uniform source terms and variable coefficients (e.g., orthotropic conductivity). The Armadillo C++ library was used for sparse matrix capabilities and the linear system solver. It is verified to be second-order accurate using trilinear elements. The code is templated so that it can be used to compute the sensitivity of the solution to design variables using the semi-analytic complex variable method. The code reads AFLR3 formatted grid files and writes ASCII Tecplot output.

In the next sections, the formulation is explained, the code is validated using an exact solution, and two set of results are presented. The first set of results involves simulation of steady-state heat transfer for a ball grid array (a surface mounted chip carrier). The entire geometry has 9 different materials, each with different material properties. The second set of results is for the inviscid, irrotational flow of an incompressible fluid around a sphere.

The code was documented using Doxygen.

2 Formulation

An example of the type of elliptic equation being solved is

$$-\nabla \cdot (k \nabla T) = Q \tag{1}$$

where $k = k(x, y, z)$, $Q = Q(x, y, z)$. Defining the residual as

$$\mathcal{R} = -\nabla \cdot (k \nabla T) - Q \tag{2}$$

Multiplying the residual by a weight function and integrating over a hexahedral element yields

$$\int w(-\nabla \cdot (k \nabla T) - Q) d\Omega = \pi \tag{3}$$

where π is a functional. This is the unsymmetric weak form. Integration by parts will be applied so that only first-order derivatives are left. In tensor notation, the equation is

$$-\int_{\Omega} w(kT_{,i})_{,i}\, d\Omega - \int_{\Omega} wQ\, d\Omega = \pi \tag{4}$$

The integration by parts in n-dimensions is

$$\int_{\Omega} uv_{,i}\, d\Omega = \oint_{\Gamma} uvn_i\, d\Gamma - \int_{\Omega} vu_{,i}\, d\Omega \tag{5}$$

Letting $u = w$ and $v = kT_{,i}$, the first integral can be written as

$$\int_{\Omega} w(kT_{,i})_{,i}\, d\Omega = \oint kwT_{,i}n_i\, d\Gamma - \int_{\Omega} kw_{,i}T_{,i}\, d\Omega \tag{6}$$

Written in vector notation, the result is

$$-\int_{\Omega} w(\nabla \cdot (k\nabla T))\, d\Omega = -\oint_{\Gamma} kw(\nabla T \cdot \hat{\mathbf{n}})\, d\Gamma + \int_{\Omega} k(\nabla w \cdot \nabla T)\, d\Omega \tag{7}$$

Now, the weak form becomes

$$-\oint_{\Gamma} kw(\nabla T \cdot \hat{\mathbf{n}})\, d\Gamma + \int_{\Omega} k(\nabla w \cdot \nabla T)\, d\Omega - \int_{\Omega} wQ\, d\Omega = \pi \tag{8}$$

The weight function can be written as

$$w(x, y, z) = \sum V_j N_j \tag{9}$$

where the N_j term represents the shape functions and V_j represents a vector of unknown coefficients. In the Galerkin method, the same basis that is used to form the shape functions for the weight function is also used to form the approximation function,

$$\widetilde{T}(x, y, z) = \sum c_i N_i \tag{10}$$

Assuming zero Neumann boundary conditions, and inserting the approximation and weight functions,

$$\pi = \int_{\Omega} \left[\left(\sum V_j \nabla N_j \right) \cdot \left(\sum c_i \nabla N_i \right) \right]\, d\Omega - \int_{\Omega} Q \left(\sum V_j N_j \right)\, d\Omega \tag{11}$$

There are two vectors of unknown coefficients (i.e., V_j and c_i). This problem can be solved by making the functional stationary with respect to the V_js. We are effectively trying to find a solution (c_i) that minimizes the functional with respect to the unknown coefficients. Dropping the summation notation and taking the partial of the functional with respect to the V_j terms yields

$$\frac{\partial \pi}{\partial V_j} = \int_{\Omega} k \left(\nabla N_i \cdot \nabla N_j \right)\, d\Omega - \int_{\Omega} QN_j\, d\Omega = 0 \tag{12}$$

The stiffness matrix is then

$$K_{ij} = \int_\Omega k \left(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z} \right) d\Omega \tag{13}$$

The load vector is

$$F_j = \int_\Omega Q N_j \, d\Omega \tag{14}$$

Discretizing and performing the numerical integrations on all of the elements yields the following linear system:

$$K_{ij} c_i = F_j \tag{15}$$

Applying the same procedure with convection boundary conditions yields the following (in tensor notation):

$$\frac{\partial \pi}{\partial V_i} = \int_\Omega k N_{i,k} N_{j,k} c_j \, d\Omega + \oint_\Gamma h N_i N_j c_j \, d\Gamma = \int_\Omega N_i Q \, d\Omega + \oint_\Gamma h N_i T_\infty \, d\Gamma \tag{16}$$

The surface integrals are somewhat complicated by the fact that we must integrate a two-dimensional shape in three dimensions. They must be transformed to two-dimensional space and integrated there. The Jacobian determinant can be written as

$$\det(\mathbf{J}) \equiv J = d\boldsymbol{\xi} \times d\boldsymbol{\eta} \cdot d\boldsymbol{\zeta} \tag{17}$$

In tensor notation,

$$J = \epsilon_{ijk} \, d\xi_j \, d\eta_k \, d\zeta_i \tag{18}$$

where

$$d\xi_j = \frac{\partial N_j}{\partial \xi} \quad d\eta_k = \frac{\partial N_k}{\partial \eta} \quad d\zeta_i = \frac{\partial N_i}{\partial \zeta} \tag{19}$$

The 2D element with arbitrary orientation in $\mathbf{R}^3$ can be transformed by setting the Jacobian to

$$J = \epsilon_{ijk} d\xi_j d\eta_k n_i \tag{20}$$

where n_i is the face unit normal vector. This is essentially enforcing no variation in the ζ direction.

The shape functions for the trilinear element can be derived using a tensor product. The 1-D linear shape function can be written as

$$\psi(\xi) = \left\{ \frac{1-\xi}{2}, \frac{1+\xi}{2} \right\} \tag{21}$$

For the trilinear element, the shape functions are defined as follows:

$$N_a(\xi, \eta, \zeta) \equiv N_{ijk}(\xi, \eta, \zeta) = \psi_i(\xi) \, \psi_j(\eta) \, \psi_k(\zeta) \tag{22}$$

Care must be taken to ensure that the node numbering matches with the numbering in the tensor product. If not, the mapping must be defined.

3 Validation

Several meshes were generated for a unit cube (i.e., hexahedral, prisms, and tetrahedral). The order of accuracy of the code was verified using the exact solution to the following equation:

$$\nabla^2 T = -3\pi \sin(\pi x) \sin(\pi y) \sin(\pi z) \tag{23}$$

with the exact solution being

$$T(x, y, z) = \sin(\pi x) \sin(\pi y) \sin(\pi z) \tag{24}$$

The temperature on each exterior surface is set to zero using Dirichlet boundary conditions.

The orders of accuracy are listed below. They were computed using the $\mathcal{L}^2$ norm of the error. That is,

$$\mathcal{E}^2 = \int (T_{\text{num}} - T_{\text{exact}})^2 \, d\Omega \tag{25}$$

The integral was evaluated using 3-point Gaussian quadrature (corresponds to 27 points in 3D). The orders of accuracy are listed below:

```
Order of accuracy for hexes is 1.999946
Order of accuracy for prisms is 1.917929
Order of accuracy for tets is 2.146208
```

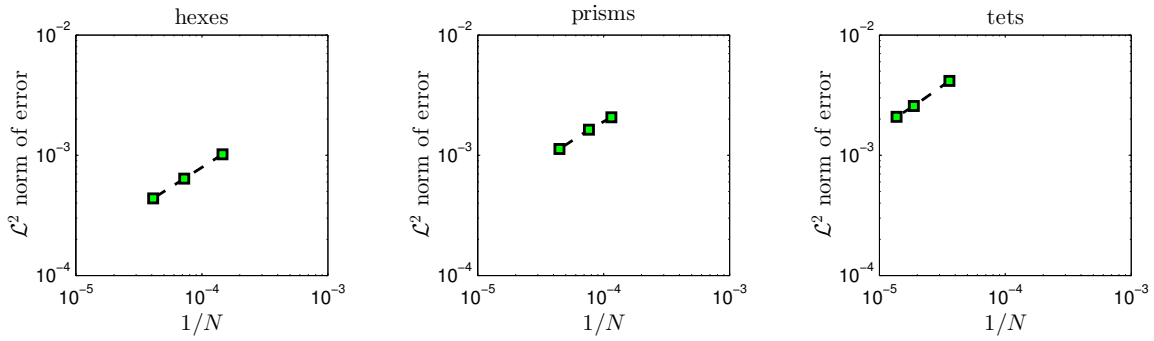


Figure 1: Plots showing the $\mathcal{L}^2$ norm of the error vs the inverse of the cube root of the number of elements.

The fine meshes used for each case are shown below:

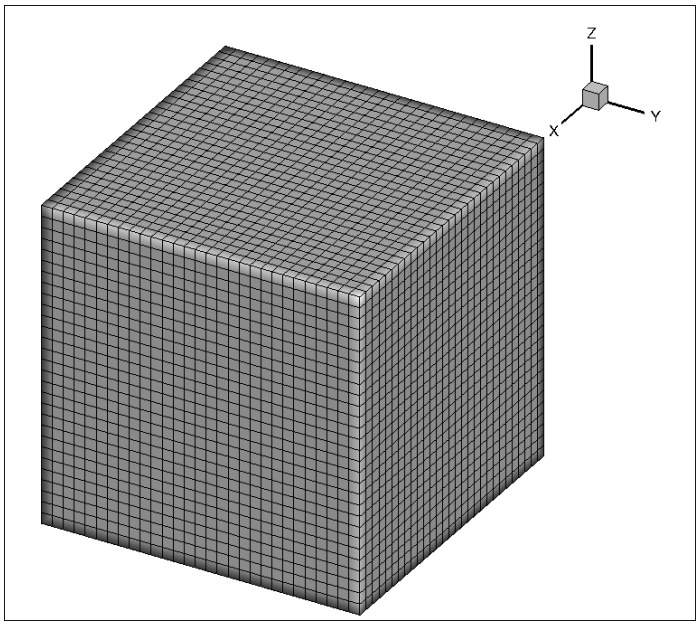


Figure 2: Hexahedral mesh.

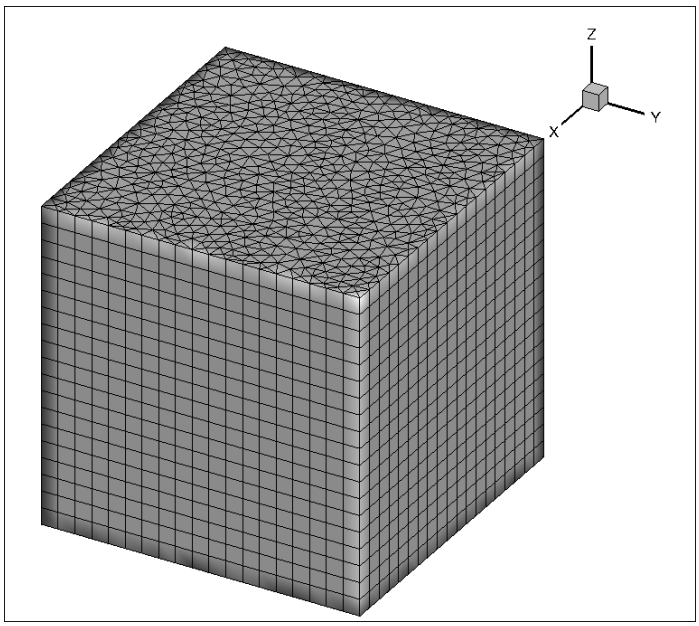


Figure 3: Prism mesh.

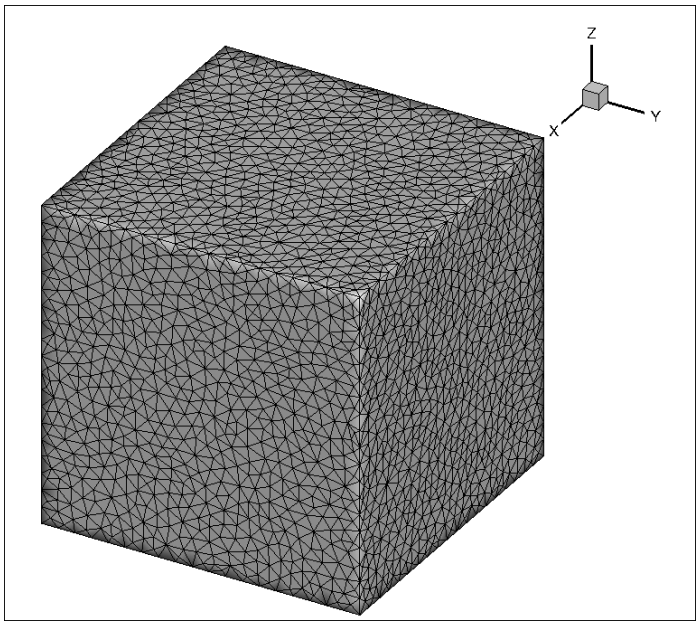


Figure 4: Tetrahedral mesh.

The solutions on each mesh are also shown below:

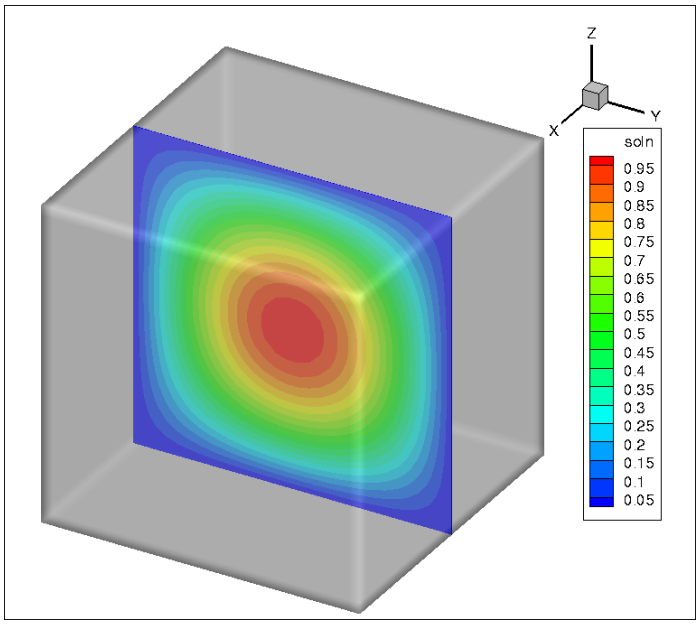


Figure 5: Hexahedral solution.

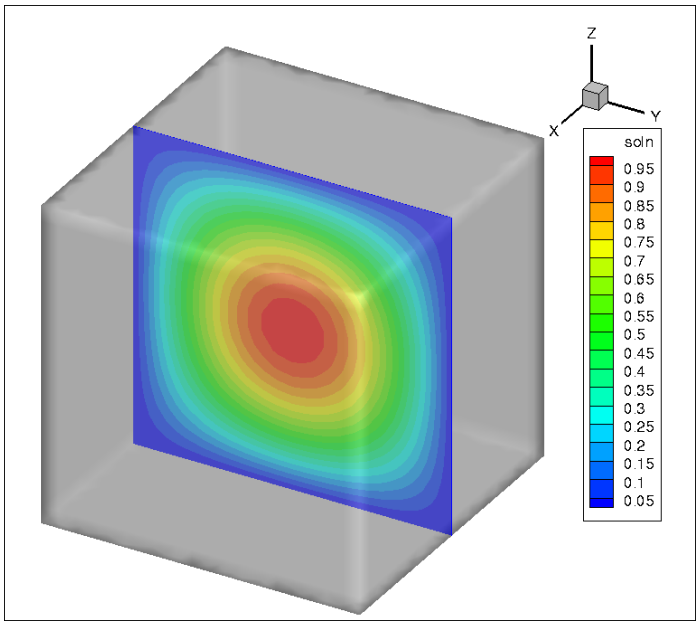


Figure 6: Prism solution.

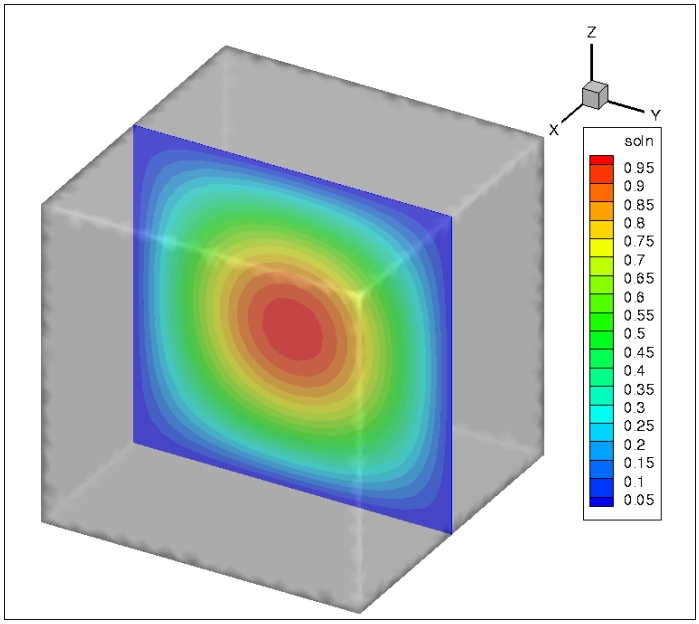


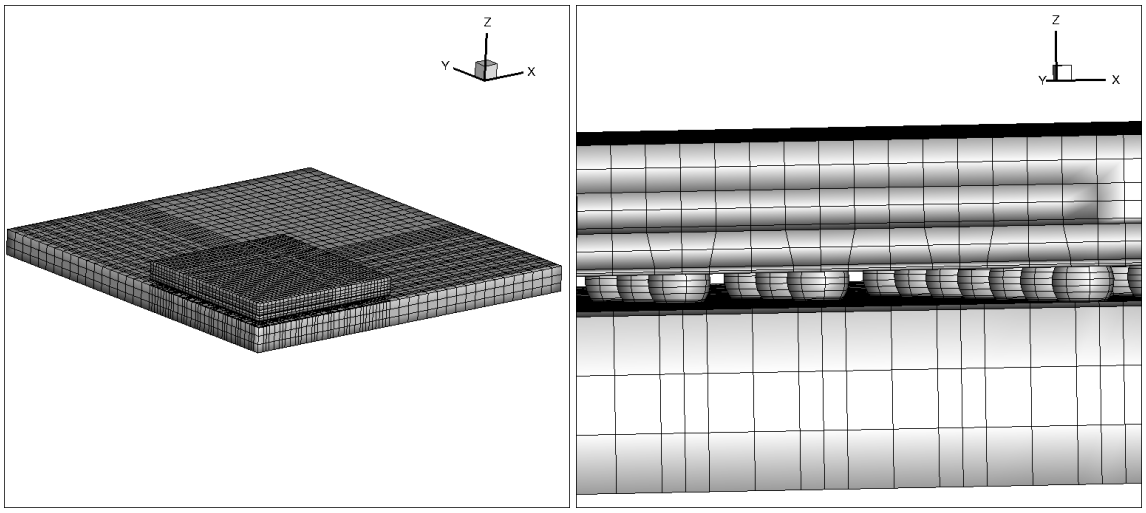
Figure 7: Tetrahedral solution.

4 Results

The code was applied to two problems, namely, steady-state heat conduction in a chip made up of 9 different materials and inviscid, irrotational flow of air around a sphere. The

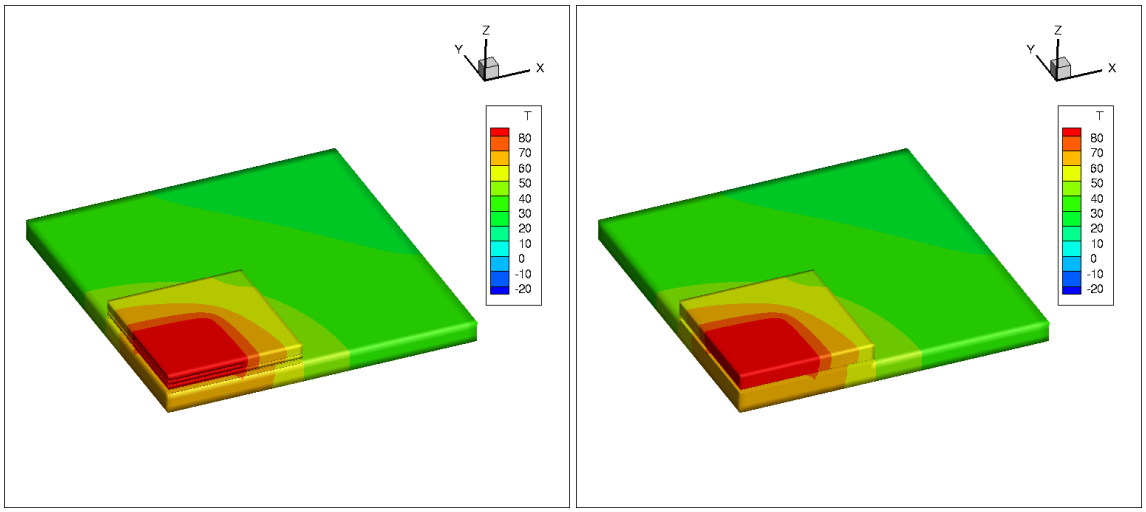
heat conduction solution was validated using an existing code that has been thoroughly validated.

4.1 Heat conduction



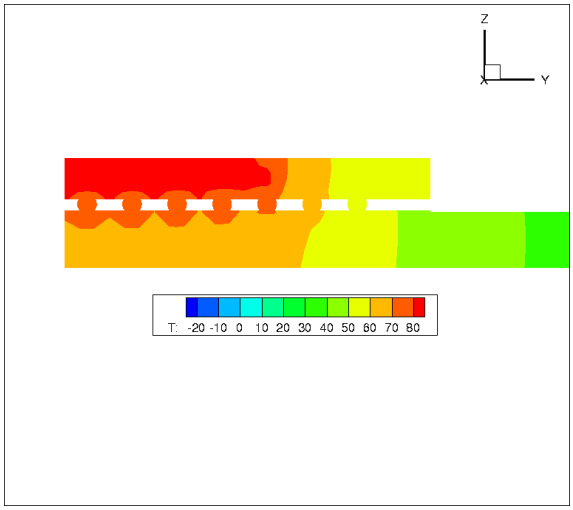
(a) BGA mesh.

(b) BGA mesh–close-up.

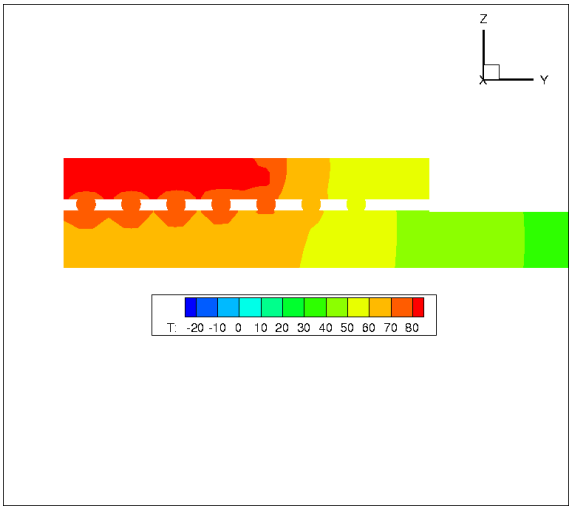


(a) Contours of temperature using well-validated code.

(b) Contours of temperature using the new code.



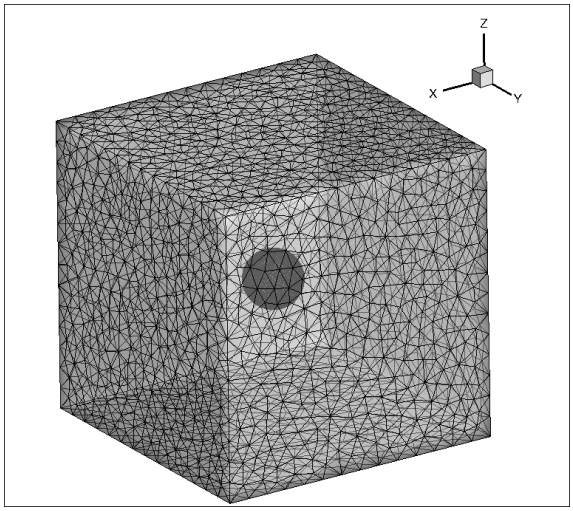
(a) Volume slice showing contours of temperature using well-validated code.



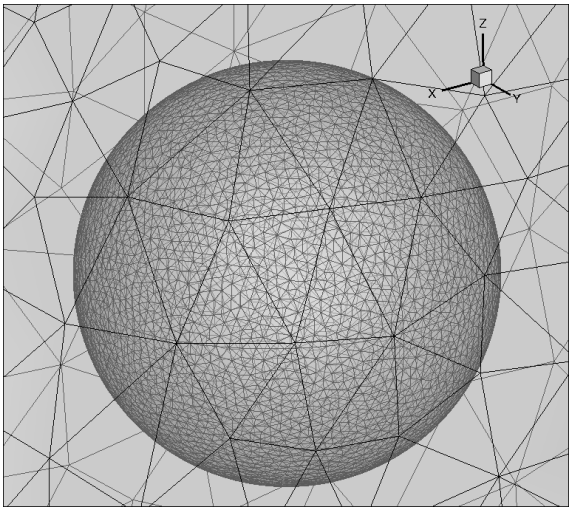
(b) Volume slice showing contours of temperature using the new code.

4.2 Potential flow

The code was also applied to potential flow around a sphere. The freestream velocity was set to 10 m/s. A hybrid mesh consisting of tetrahedral and prism cells was generated using Pointwise. Prisms were extruded from the surface of the sphere.



(a) Mesh for potential flow case



(b) Close-up of sphere inside mesh for potential flow case.

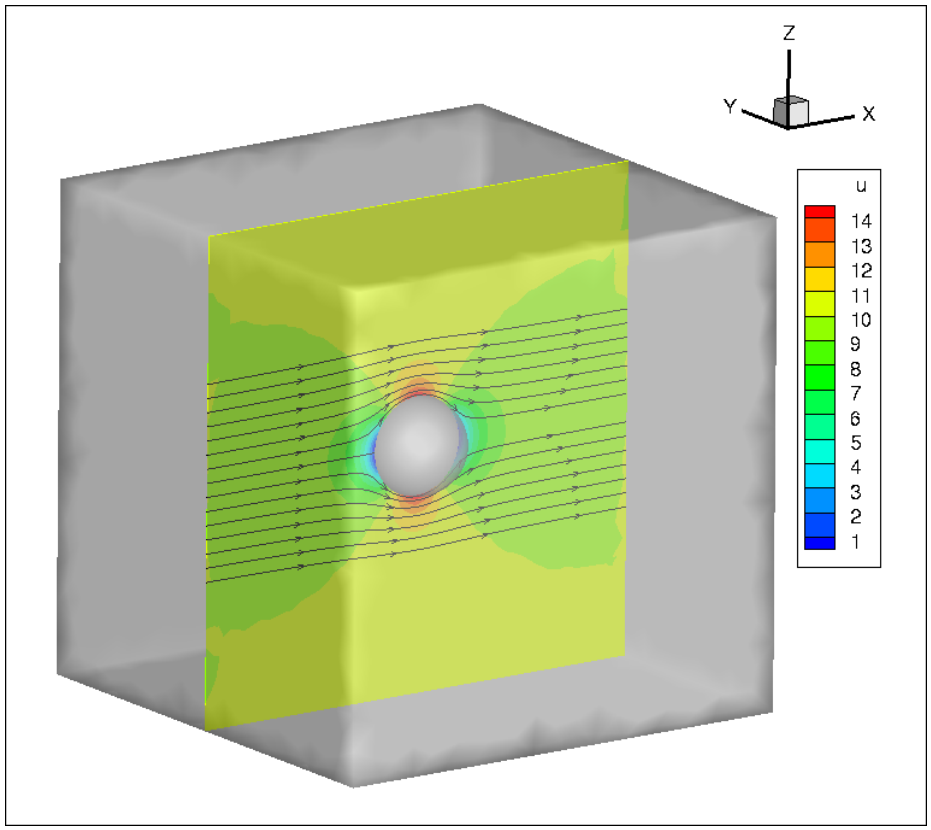


Figure 12: Slice showing contours of the x-component of velocity along with streamlines.

5 Future work

The code will be slightly modified so that sensitivities to design variables can be computed using the semi-analytic complex variable method. Also, the current setup uses a direct solver. Iterative solvers will be added.