DIRECT NUMERICAL SIMULATION OF THE

FLOW IN A PEBBLE BED

A Thesis

by

PAUL D. WARD

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2014

Major Subject: Nuclear Engineering

ABSTRACT

The flow in a tightly packed array of spheres is important to various engineering fields. In nuclear engineering applications, for instance, researchers have proposed core geometries of the pebble bed reactor (PBR) type cooled by gas or molten salt. Proper core cooling, both at operation and during accident conditions, is a key issue that must be addressed in any reactor design; and the limited amount of data available for the complicated geometry of PBR cores makes this task even more complex. A detailed understanding of coolant flow patterns and properties must be developed in order to meet safety requirements and ensure core longevity.

We addressed this issue by using the spectral-element computational fluid dynamics code Nek5000, developed at Argonne National Laboratory, to conduct both large eddy simulation (LES) and direct numerical simulation (DNS) of fluid flow through a single face-centered cubic sphere lattice with periodic boundary conditions. Multiple LES were conducted with varying Reynolds numbers in an effort to determine how the Reynolds number affects the development of asymmetries within the flow patterns. The DNS focused on the development of turbulence and were used to compute the turbulent kinetic energy budgets. A set of statistical analyses were also conducted to support the validity of the results.

ACKNOWLEDGEMENTS

This work was supported by U.S. Department of Energy (DOE) Office of Nuclear Energy as part of the NEAMS Program. Argonne is a U.S. Department of Energy (DOE) Office of Science laboratory, operated under contract DE-AC02-06CH11357.

I would like to extend special thanks to Elia Merzari for his help, training, and advice, without which, this project would not have been completed. I am also grateful to Yassin Hassan for organizing this research opportunity and for his advice and support throughout the process.

TABLE OF CONTENTS

LIST OF FIGURES

Page

Page

1 INTRODUCTION

1.1 *Project Description*

The last two decades have seen a resurgence in nuclear research fueled by the ever-increasing demand for cheap energy. New reactor designs are being developed to replace the aging reactors currently in service. These new designs must be both safer and more efficient than their predecessors and a particularly large emphasis has been placed on passive safety measures. One of the more promising designs to come out of this resurgence is the High Temperature Gas Reactor (HTGR).

The vast majority of nuclear reactors in operation today use water as the primary coolant. This water is either boiled directly in the core or kept as a liquid in a pressurized system and used to boil water in a secondary loop. In both cases, the resulting steam is used to turn a turbine and produce electricity. Generally, this process takes place at temperatures that peak around 300 C. The HTGR is designed to operate at much higher outlet temperatures up to 1000 C. This increased outlet temperature allows for better efficiency in the power generation cycle and drives a number of design changes. Water is not a practical coolant at these temperatures and, as a result, most HTGR's are designed to use helium as a coolant. The choice of helium as a coolant has added benefits. Helium is inert meaning it will not react with any other materials in the system. Additionally, helium is not activated, or made radioactive, by exposure to the radiation in the core. Certain high temperature reactors use liquid metal or salts as coolants, but this study will focus solely on gas-cooled designs.

The fuel elements in HTGR’s are also quite different than their counterparts in water-cooled reactors. Water cooled reactors generally use pellets of fissionable material clad in a protective coating. These pellets are stacked forming rods and arranged in the core. The water coolant often acts as the neutron moderator in these reactors. HTGR’s use microspheres of fissionable material coated with multiple protective layers. These layers include graphite which also acts as a moderator. By including the moderator in the fuel elements, this design is more stable than water reactors which can lose moderator in accident scenarios, exacerbating the situation. The microspheres are packed into a matrix and formed into whatever fuel element shape is desired.

The higher operating temperatures require changes to be made to the reactor structure as well. Many high-temperature designs use graphite as a primary structural material. Graphite is heat-resistant, strong, and capable of withstanding the heavy radiation found in the core environment. It also provides further moderation where necessary. High-temperature alloys similar to those used in turbine engines must be used in place of steel for other structural components.

HTGR reactors tend to fall into two broad categories based on core designs: prismatic and pebble bed. Prismatic designs use hexagonal rods of fuel elements and reflector material to create the core structure. The prismatic design has the advantages of having a fixed structure, simple flow patterns, and a much greater pool of simulated and experimental data from which to draw from. An example of a prismatic core and a pebble bed core can be seen in Figures 1 and 2, respectively.

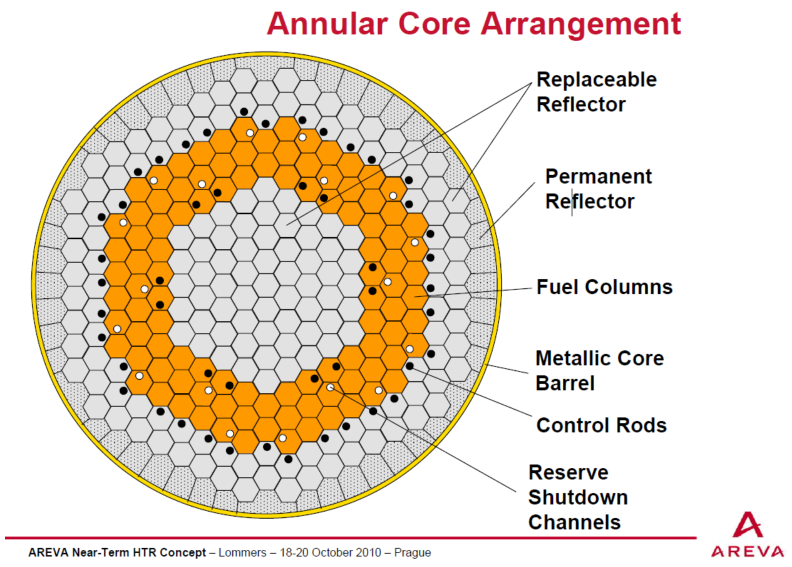


Figure 1. Prismatic Core Diagram [1]

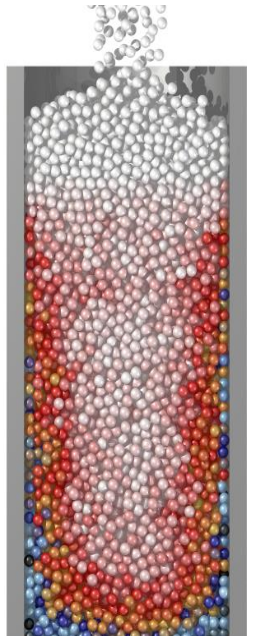


Figure 2. PBR Core Example [2]

Pebble bed reactors (PBR) use spherical fuel elements composed of many microspheres packed into a larger spherical shell. These so-called pebbles are poured into and circulate through the reactor vessel. Fuel elements may make multiple passes through the core allowing for the ability to monitor their condition mid-cycle. This configuration naturally benefits from easy fuel replacement, but is more complicated to design and study.

Unfortunately, these designs are not without drawbacks. Chief among them is the material deterioration caused by reactor's high temperature and radioactive flux environment. While there exist a number of candidate materials that may be able to withstand such an environment, a significant amount of testing must be done to ensure acceptable structural longevity. Additionally, the pebble arrangement results in complex coolant flow patterns that are difficult to model and may result in areas of insufficient flow leading to the formation of hot spots. Such hot spots can cause non-uniform expansion and wear of the pebbles and lead to structural failure.

The goal of this project is to perform various simulations of the flow between these pebbles in an attempt to characterize the details of the expected flow patterns. Two different computational approaches will be used: large-eddy simulation (LES) and direct numerical simulation (DNS). The two methods are differentiated based on scale. Energy is transported within a fast-flowing fluid by whorls and vortices in a process known as turbulence. These vortices are characterized by size ranges, or length scales, with energy flowing from larger to smaller until it is eventually dissipated as heat. This process is expanded upon in the following theory section.

LES only models the larger vortices and uses filtering or averaging to take the place of the smaller flow structures. This allows the simulation resolution, and therefore the required computational time, to be much lower while producing generally accurate results. This project will perform a number of LES at varying Reynolds numbers in an attempt to both verify the results of the DNS and to analyze the effect of Reynolds number on the flow pattern. These LES will complement the study's primary focus which consists of two DNS. DNS resolves the smallest turbulent eddies and, when properly conducted, produces nearly perfect simulations of the flow characteristics. The two DNS will be conducted at different resolutions and focus on the development of turbulence within the domain.

These simulations will characterize the fluid's velocity profiles and give insight into the transfer of energy throughout the flow. Such characterization of the flow will allow for the prediction of hot spot formation which would, in turn, aid in the modification of flow parameters to prevent such developments, making future reactors safer. The analysis will be performed using Nek5000, a Computation Fluid Dynamics (CFD) code developed at Argonne National Laboratory (ANL). The analysis will also be used to confirm and expand upon the results of similar simulations and relevant experiments such as those conducted by Shams et al. at NRG [3] and Hassan and Dominguez-Ontiveros[4]. Specifically, the NRG study found high levels of asymmetry in the velocity profiles across the central gap. A major focus of this study was the attempt to determine why the NRG simulations developed such asymmetries and whether or not Nek will produce similar results.

2 THEORY

2.1 *Governing Equations*

The behavior of a fluid can be described using equations formulated from the laws of conservation, specifically the conservation of mass, momentum, and energy. This study will focus primarily on the conservation of momentum, as the behaviors of interest are governed by this law. Mass conservation is expressed in the continuity equation:

$$\frac{d\rho}{dt} + \nabla \cdot (\rho \boldsymbol{U}) = 0$$

Where ρ is the density of the fluid and $\boldsymbol{U}$ is the velocity of the fluid. The continuity equation arises from the fact that the mass flow rates in and out of a system must be equal in a steady-state situation. If the fluid is incompressible, having constant density, then continuity equation shows that the divergence of the fluid is zero. The fluid considered in this study is assumed to be incompressible.

The law of conservation of momentum can be modified for the case of incompressible Newtonian fluids, which describes most fluids of interest, to form the Navier-Stokes equation:

$$(\frac{\partial \boldsymbol{U}}{\partial t} + \boldsymbol{U} \cdot \nabla \boldsymbol{U}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{U} + f$$

Here, p is the pressure, ν is the kinematic viscosity, and f represents body forces acting on the fluid. The Navier-Stokes equation compares the acceleration of the fluid, represented by the unsteady and convective terms on the left side of the equation, to the pressure gradient and viscous effects, which make up the stress divergence, along with the body forces on the right side of the equation.

A few key assumptions are made in addition to that of incompressible flow. The first is that the fluid, while composed of many individual particles, acts as a continuum and properties such as temperature, pressure, and velocity are continuous throughout the fluid. Additionally, the fluid is constrained to be stationary where it contacts a surface.

Even with these assumptions and conditions, it is often prohibitively computationally expensive to find a numerical solution to the Navier-Stokes equation. This difficulty arises from the development of turbulence. Turbulence is characterized by chaotic and varied pressure and velocity changes that develop as kinetic energy is transferred throughout a fluid. These changes often appear as vortices or eddies of varying length. Kinetic energy is transferred from larger to smaller eddies until the scale decreases to a point where viscous effects transform the rotational energy to heat. Many different models are employed to simulate the effect of these smaller eddies without actually resolving them, thereby decreasing the computational cost. This study will attempt to resolve the finest turbulent length scales without using any models in a process known as Direct Numerical Simulation (DNS).

2.2 *TKE Budgets*

It is important to understand how turbulence is formed and how it transfers energy throughout the fluid. One of the most useful descriptions of this process is the turbulent kinetic energy balance equation:

$$0 = -\frac{\bar{D}}{\bar{D}t}\langle u_i u_k \rangle - \frac{\partial}{\partial x_k}\langle u_i u_j u_k \rangle + \nu \nabla^2 \langle u_i u_j \rangle + P_{ij} + \Pi_{ij} - \varepsilon_{ij}$$

The notation presented here is taken from Turbulent Flows by Stephen B. Pope [5]. In order, the terms are known as the mean convection, turbulent convection, viscous diffusion, production, pressure effects, and dissipation. The variable *u* represents the fluctuating velocity components, *v* is the viscosity, and the subscripts *i* and *j* are directional components. The subscript *k* iterates through each of the directional components such that u_k is equal to $u_x + u_y + u_z$. Each term is a tensor with two directional components denoted by *i* and *j*.

The mean convection is considered to be zero for fully developed steady-state flows. The turbulent convection term represents the transfer of kinetic energy by turbulent motion. The viscous diffusion term describes the dispersal of energy by viscous forces. The production term is expanded to yield:

$$P_{ij} = -\langle u_i u_k \rangle \frac{\partial \langle U_j \rangle}{\partial x_k} - \langle u_j u_k \rangle \frac{\partial \langle U_i \rangle}{\partial x_k}$$

Here, *U*, is the fluid velocity and the term represents the transfer of energy from the fluid's mean motion to the turbulent eddies. The pressure term represents the transfer of energy by pressure forces and can be divided into two parts: the pressure transport, T_{kij}, and the pressure strain, R_{ij}:

$$\Pi_{ij} = R_{ij} - \frac{\partial}{\partial x_k} T_{kij}$$

where $R_{ij} = \langle \frac{p}{\rho} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \rangle$ and $T_{ijk} = \frac{1}{\rho} \langle u_i p \rangle \delta_{jk} + \frac{1}{\rho} \langle u_j p \rangle \delta_{ik}$

Here, *p*, represents the pressure. The final term is the dissipation of turbulent kinetic energy and is represented by the tensor quantity:

$$\varepsilon_{ij} = 2\nu \langle \frac{\partial u_i}{\partial x_k} \frac{\partial u_j}{\partial x_k} \rangle$$

These terms quantify the behavior of turbulence within a specified domain and computing them allows for a detailed analysis of this important flow parameter.

2.3 *Solver Details*

Nek5000 simulates unsteady incompressible Navier-Stokes equations using the spectral element method of spatial discretization with either implicit or explicit time-stepping. A detailed description can be found in the Nek5000 notes and Fischer [6] and Deville et al. [7]. Information regarding the validation of the code and related studies can be found in Obabko et al. [8] and Merzari et al. [9]. In simple terms, the spectral element method differs from the more common finite difference and finite element methods by dividing the domain into quad or hex elements and approximating the governing equations using a trigonometric series. This approach allows for much higher order approximations and, consequently, much higher numerical accuracy.

Nek is also capable of performing LES using the filter developed by Fischer and Mullen [10]. This method explicitly filters the solution after every time step using the filter operator F_α defined as:

$$F\alpha = \alpha I_{N-1} + (1 - \alpha)$$

Where I is the identity operator and I_N is the interpolation parameter at N+1 GLL nodes. This filter preserves the spectral convergence and goes to zero exponentially as N goes to infinity.

3 MODULE DEVELOPMENT AND VERIFICATION

3.1 *Derivation*

The first goal of this project was to develop a module for Nek that could compute the averaged tensor quantities that are used to calculate the budget terms. The first step towards accomplishing this goal was the derivation of the budget terms as a function of the average velocities. This was accomplished by making the substitution $u = U$ - $<U>$ and solving for each individual tensor. This is known as the Reynolds Decomposition and represents the velocity fluctuations as the difference between the instantaneous and mean velocities. For example:

$$\varepsilon_{uv} = 2\nu \langle \frac{\partial u}{\partial x_k}\frac{\partial v}{\partial x_k}\rangle = 2\nu \langle \frac{\partial (U - \langle U\rangle)}{\partial x_k} \times \frac{\partial (V - \langle V\rangle)}{\partial x_k}\rangle$$

$$= 2\nu \langle \frac{\partial U}{\partial x_k}\frac{\partial V}{\partial x_k} - \frac{\partial U}{\partial x_k}\frac{\partial \langle V\rangle}{\partial x_k} - \frac{\partial \langle U\rangle}{\partial x_k}\frac{\partial V}{\partial x_k} + \frac{\partial \langle U\rangle}{\partial x_k}\frac{\partial \langle V\rangle}{\partial x_k}\rangle$$

$$= 2\nu \left[\langle \frac{\partial U}{\partial x_k}\frac{\partial V}{\partial x_k}\rangle - \frac{\partial \langle U\rangle}{\partial x_k}\frac{\partial \langle V\rangle}{\partial x_k}\right]$$

$$= 2\nu \left[\langle \frac{\partial U}{\partial x}\frac{\partial V}{\partial x}\rangle + \langle \frac{\partial U}{\partial y}\frac{\partial V}{\partial y}\rangle + \langle \frac{\partial U}{\partial z}\frac{\partial V}{\partial z}\rangle - \frac{\partial \langle U\rangle}{\partial x_k}\frac{\partial \langle V\rangle}{\partial x_k} - \frac{\partial \langle U\rangle}{\partial x_y}\frac{\partial \langle V\rangle}{\partial x_y} - \frac{\partial \langle U\rangle}{\partial x_z}\frac{\partial \langle V\rangle}{\partial x_z}\right]$$

Some terms are more complex, but because the instantaneous and average velocity components are easily accessible in Nek, this separation and expansion process is not difficult to implement and allows for the average quantities to be calculated with a minimum of round-off error. In total, 60 quantities were averaged at every time step. With the average quantities in place, a series of subroutines were created to calculate each budget term. Due to symmetry, a number of tensor terms can be ignored leaving 30 individual terms.

3.2 *Primary Routines*

The full code can be found in Appendix A. Selected routines are explained below.

3.2.1 Budgets_avg

The budgets_avg routine calls a series of averaging subroutines that compute the time average of a given variable at each time step. The computed averages include standard averages, averages that are used to compute the root mean squared, or RMS, values in post-processing, averaged of velocity gradients, and multiplicative combinations of the above. These averaged quantities are then outputted in Nek's .fld format.

3.2.2 Budget_terms

The budget_terms routine loads the .fld files produced by the averaging routine using the load_avgs subroutine. The load_avgs subroutine syntax works as follows:

```
- call load_avgs(urms,vrms,wrms,namef,numf)
```

This opens a series of .fld files from one to a number, *numf* , each with a name, *namef*, pulls the three variables *urms*, *vrms*, and *wrms*, and averages the content of each .fld file for each variable. This step is necessary because the simulation is too large to be completed in one run and must be divided into multiple steps resulting in multiple .fld files.

Once the averaged quantities have been loaded, a series of subroutines are called to compute the individual turbulent budget terms. An example of this process being used to calculate the uu production term is presented below:

```
- call ke_prod(prd_uu,uavg,vavg,wavg,uavg,uavg,urms,uvms,wums,
$                urms,uvms,wums)
```

Where the ke_prod subroutine does the following:

```
- subroutine ke_prod(prd,u,v,w,a,b,aums,avms,awms,bums,bvms,bwms)

      call gradm1(adx,ady,adz,a)
      call gradm1(bdx,bdy,bdz,b)

do e=1,lelt
do k=1,lz1
do j=1,ly1
do i=1,lx1
prd(i,j,k,e) = (a(i,j,k,e)*u(i,j,k,e)-aums(i,j,k,e))*bdx(i,j,k,e)+
$               (a(i,j,k,e)*v(i,j,k,e)-avms(i,j,k,e))*bdy(i,j,k,e)+
$               (a(i,j,k,e)*w(i,j,k,e)-awms(i,j,k,e))*bdz(i,j,k,e)+
$               (b(i,j,k,e)*u(i,j,k,e)-bums(i,j,k,e))*adx(i,j,k,e)+
$               (b(i,j,k,e)*v(i,j,k,e)-bvms(i,j,k,e))*ady(i,j,k,e)+
$               (b(i,j,k,e)*w(i,j,k,e)-bwms(i,j,k,e))*adz(i,j,k,e)
enddo
enddo
enddo
enddo

return
end
```

The variable declarations have been omitted for brevity's sake. The variables are passed based on position within the two calling statements. The gradients of the average u-velocity are taken and combined with the other previously computed quantities to produce the turbulent production. This is the standard form for each of the budget term subroutines.

After each of the budget terms has been calculated, the results are output as .fld files. These files can be loaded in a visualization program such as VisIt or used as a restart parameter in Nek. The terms themselves are fed into the interp_var subroutine whose syntax follows:

```
- call interp_var(vel1,xyz1,n_interp,uavg,vavg,wavg)
```

This subroutine outputs a variable, *vel1*, which contains the values produced by interpolating *uavg*, *vavg*, and *wavg* at a number of points, *n_interp*, with locations specified by *xyz1*. This is useful for producing profiles of the desired variables over different spaces within the domain. The actual interpolation routine is included in the Nek distribution and the interp_var subroutine presented here is a modified way of initiating the process with more generalized calling statements. Finally, the interpolation results are written to text files in the .csv format so they can be easily loaded in spreadsheet form.

3.3 *Verification*

The results were compared to the data in the University of Tokyo DNS database [11], available online at http://thtlab.jp/DNS/dns_database.html. Their data was taken from DNS analysis of two-dimensional flow between two parallel plates. They used a domain with dimensions of $2\pi \cdot$ delta, $2 \cdot$ delta, and $5\pi \cdot$ delta in the x, y, and z directions respectively. Here, delta is the half-channel height. They discretized their domain using 128x128 Fourier series in the x and z directions and a 96th order Chebyshev polynomial in the y direction. We matched this geometry and subdivided it into 12 elements in each direction. 13th order polynomials were used to discretize these elements.

Periodic boundary conditions were used in the x and z directions so, statistically, the flow should not have any meaningful variations in those directions. Therefore, the flow was averaged over both directions leaving it only as a function of y. This drastically decreased the amount of time averaging required for statistical convergence.

The Reynolds number used in the simulation was 4560 based on the channel height of two and an average u-velocity of one. Filtering was turned off allowing for the resolution of the smallest turbulent scales. The flow was allowed to develop for 300 seconds using a polynomial order (lx1) of five. The order was raised to 13 and run for another 100 seconds before the budget subroutine was called and the required averages taken over a period of 75 seconds, equating to just over 4.5 run through times. As expected, the spatial averaging yielded much better results in much less time.

Figures 3-12 compare the terms calculated by Kasagi to those obtained through this module and are shown on the following pages.

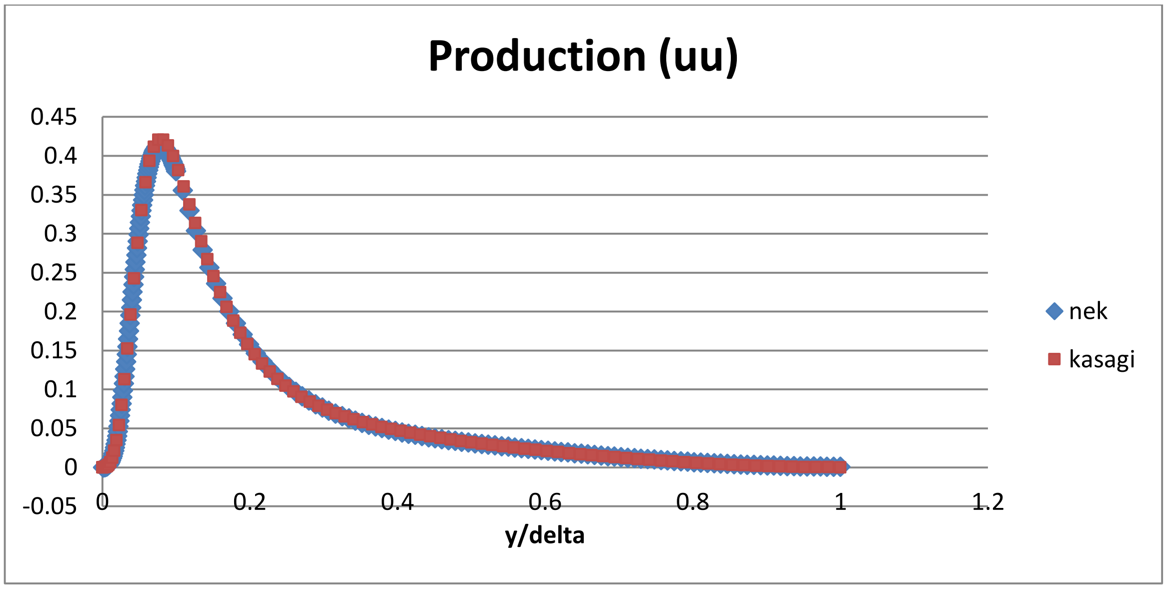


Figure 3. Turbulent Production – uu Tensor Component

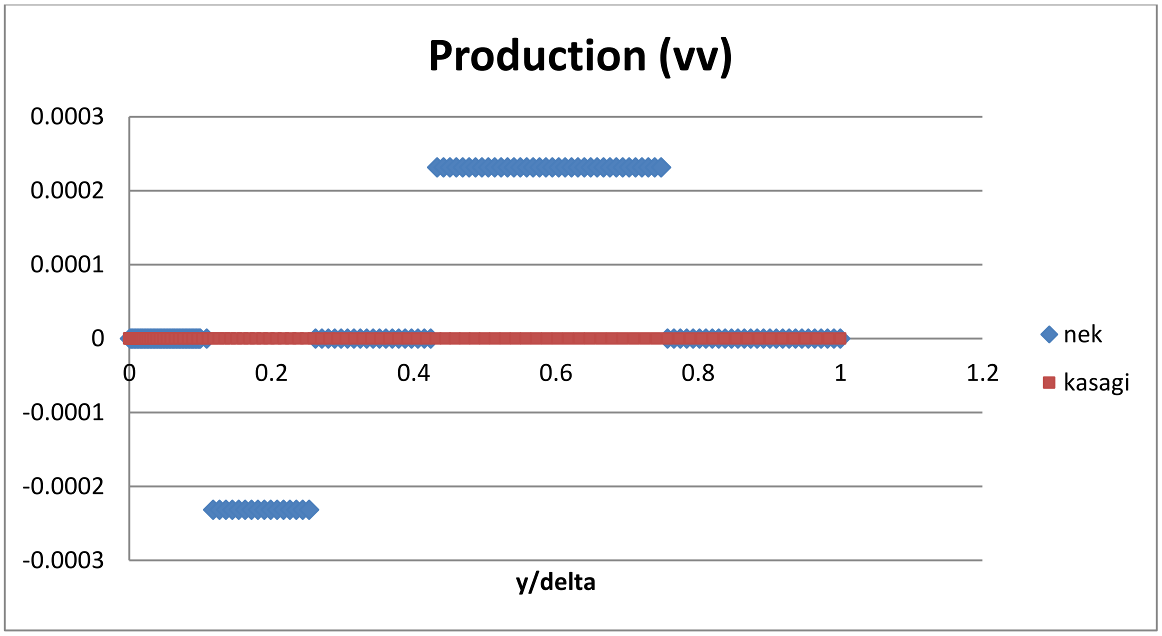


Figure 4. Turbulent Production – vv Tensor Component

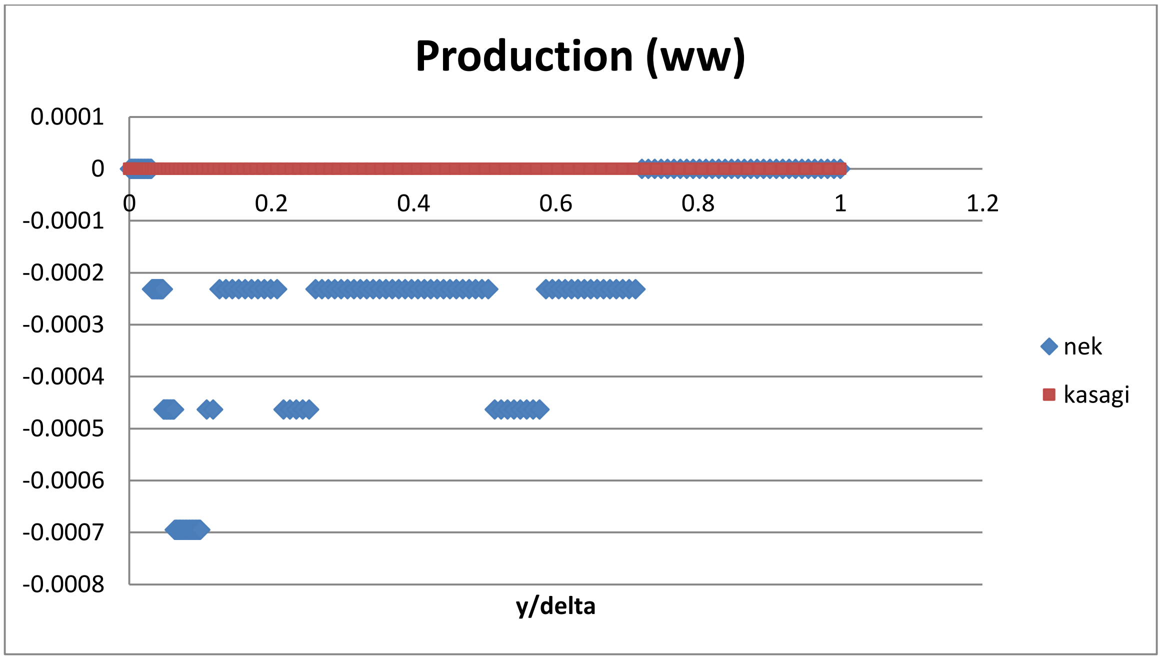


Figure 5. Turbulent Production – ww Tensor Component

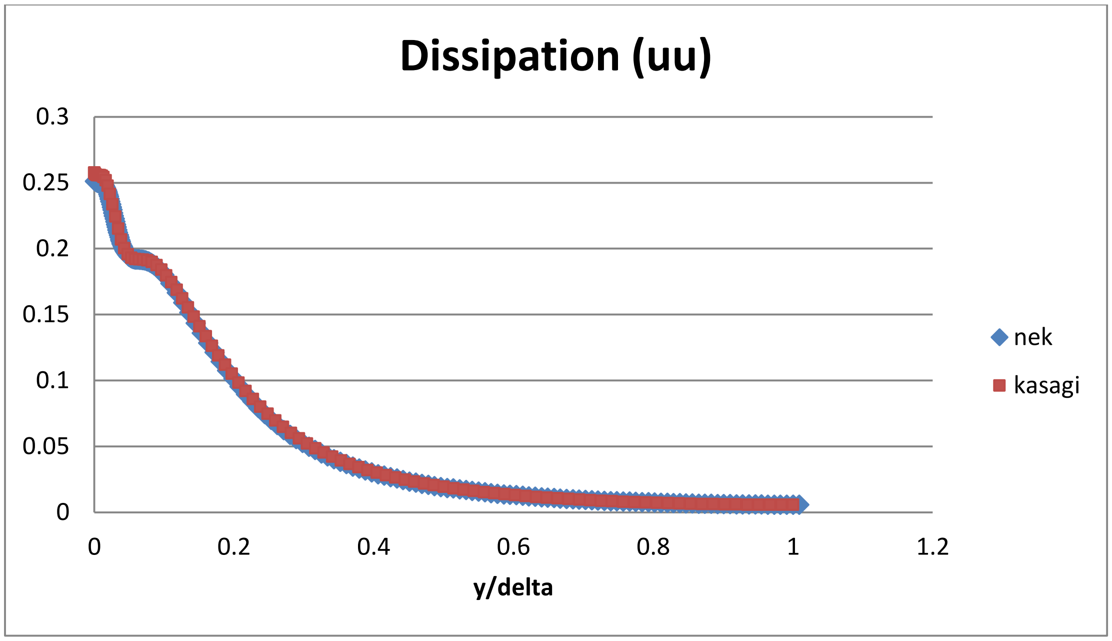


Figure 6. Turbulent Dissipation – uu Tensor Component

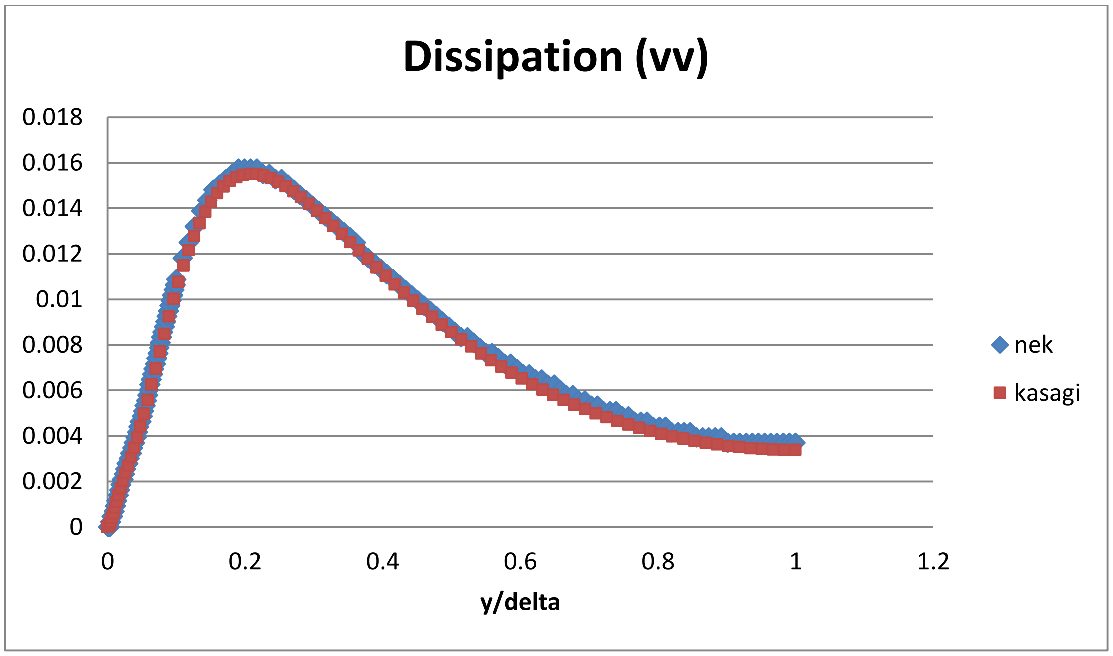


Figure 7. Turbulent Dissipation – vv Tensor Component

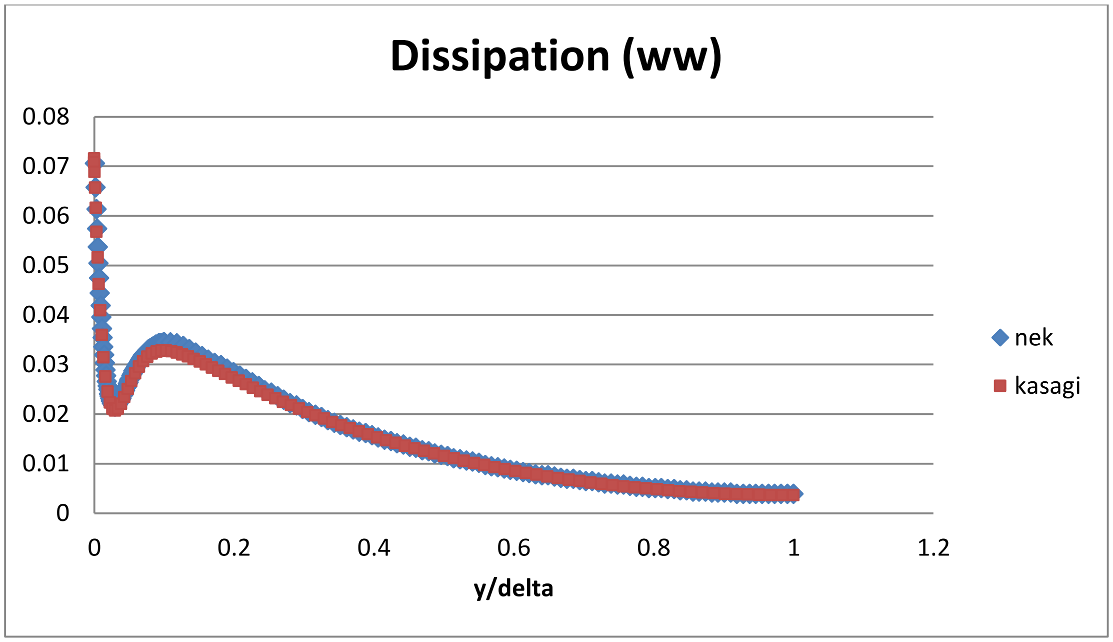


Figure 8. Turbulent Dissipation – ww Tensor Component

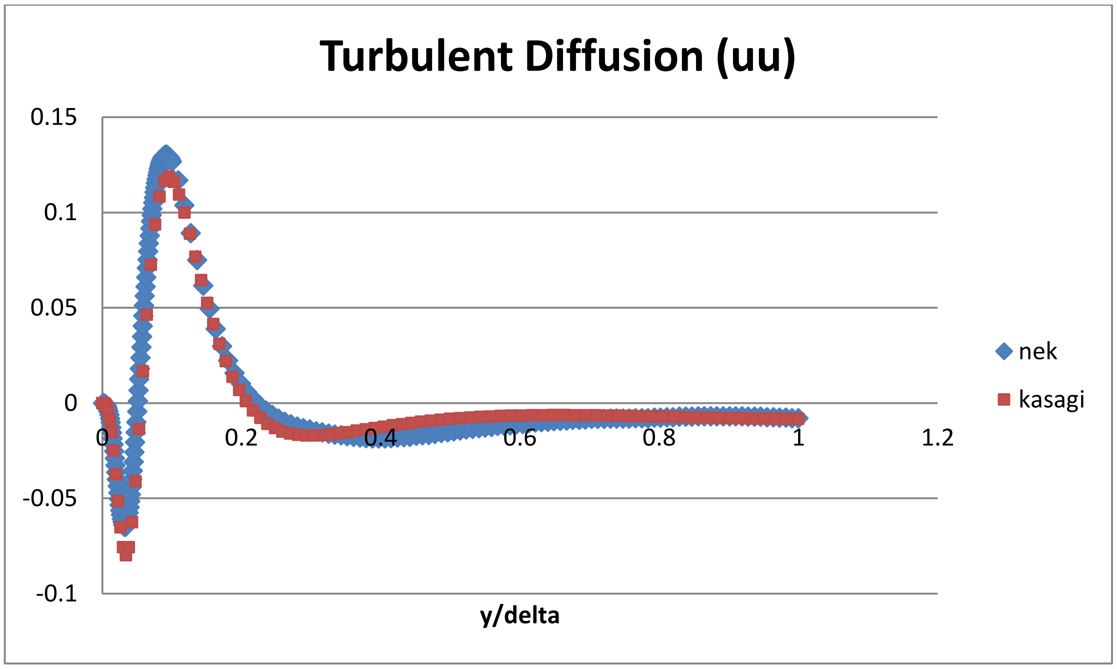


Figure 9. Turbulent Diffusion – uu Tensor Component

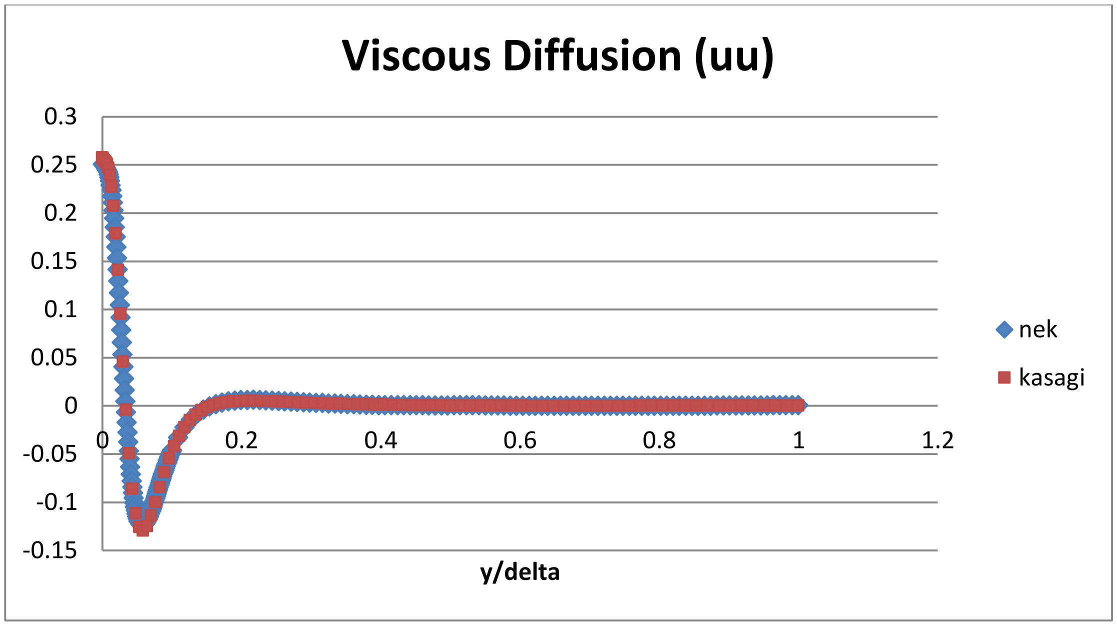


Figure 10. Viscous Diffusion – uu Tensor Component

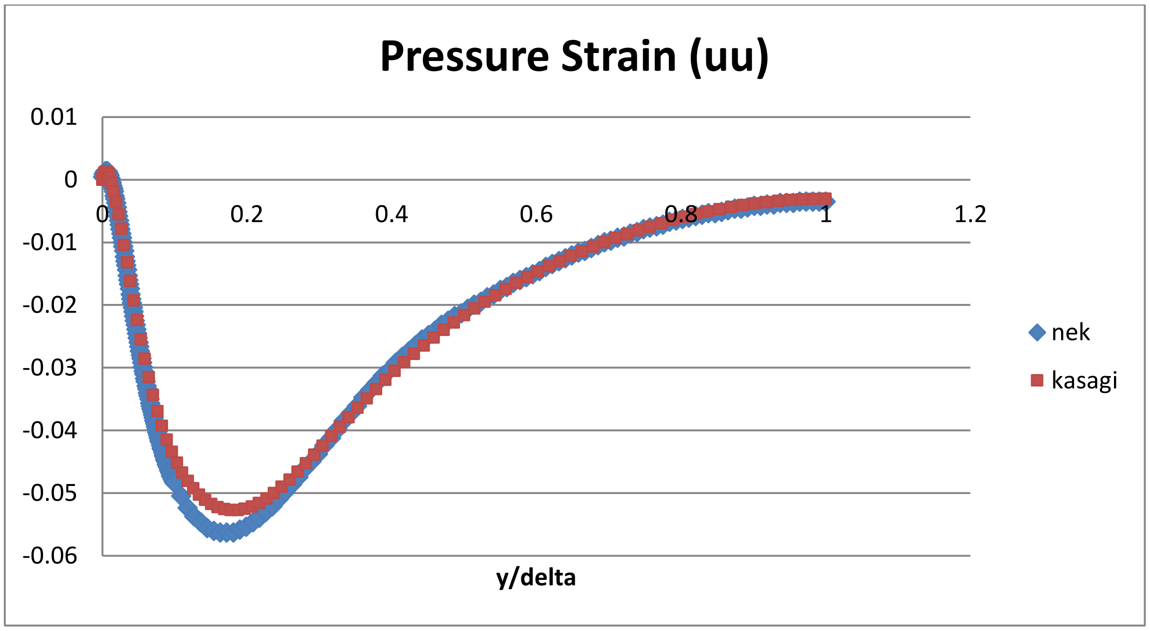


Figure 11. Pressure Diffusion – uu Tensor Component

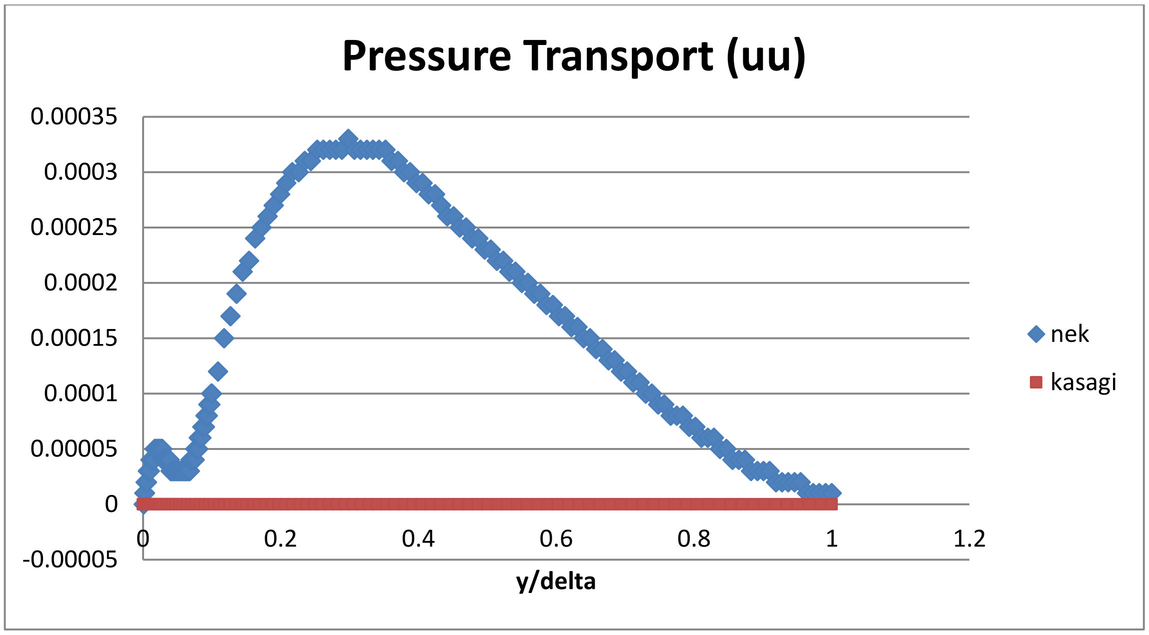


Figure 12. Pressure Transport – uu Tensor Component

In each figure, the horizontal axis is normalized by the channel half-height, which is one in this case. Most of interesting behavior occurs in or near the boundary layer. Most terms are in very good agreement with Kasagi's data. However, the vv and ww production terms as well as the pressure transport term , Figures 4,5, and 12 respectively, wildly differ. This is because these terms are expected to be on the order of 10^{-10} or smaller and random statistical noise generated in the simulation is being displayed giving a nonsense result. Nek gives very small values for these terms, but they are not as small as the essentially zero values produced by Kasagi. The turbulent diffusion differs slightly, possibly because the triple average term required more time to statistically converge.

4 SIMULATION PARAMETERS

With the budgets module verified, the focus shifted to analyzing the velocity distribution and development of turbulence within the pebble bed lattice.

4.1 *Mesh Characteristics and Generation*

The spherical pebbles are arranged in a face-centered cubic (FCC) distribution across a domain ranging from negative to positive one in each direction. This domain is shown in Figure 13, on the following page, along with the primary plane from which interpolations were taken.

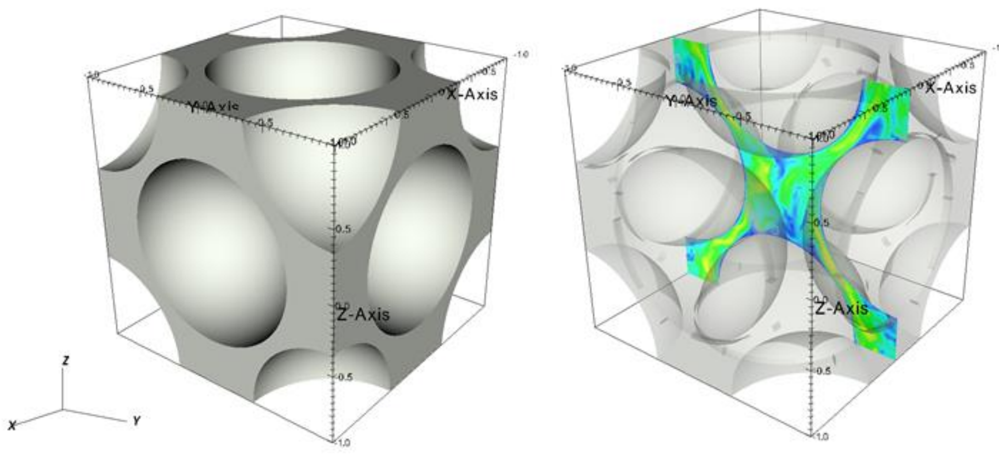


Figure 13. Simulation Domain and Interpolation Plane

Four half-spheres are located on the x and y boundaries while eight quarter spheres are found in the corners of the domain. The spheres have a diameter of 1.305 with a gap of 0.1088 between them. The flow has a volume-averaged magnitude of one in the z direction. Figure 14 highlights the line over which the following graphs were interpolated.

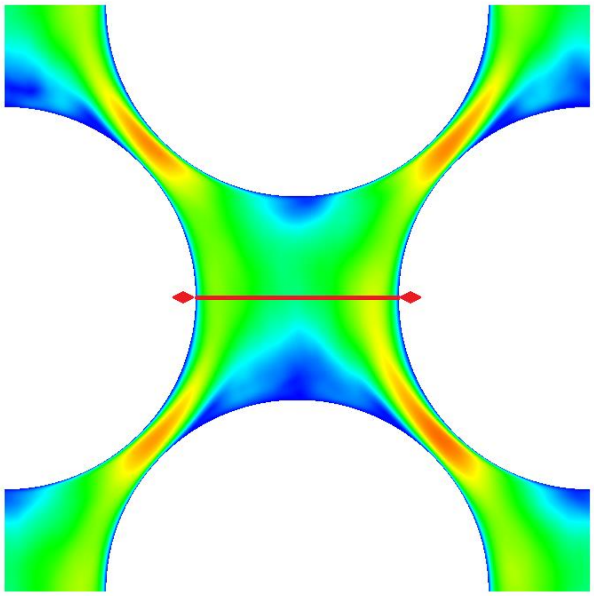


Figure 14. Interpolation Line

The mesh used in this simulation was generated by Paul Fischer in Prenek, a module included in the Nek5000 distribution, using the FCC option in the three-dimensional meshing section. There are 28672 fully hexahedral elements in total with periodic boundary conditions on each side. A depiction of the central region of the mesh can be found in Appendix B. Two simulation runs were conducted: one with a polynomial order of seven, resulting in just under 15 million total divisions; and one with an order of 11, yielding slightly less than 50 million divisions. Heat transfer was not simulated and no filtering or turbulence modeling was used.

4.2 *Simulation Details and Process*

Initially, both DNS runs were run on 2048 processors each for roughly 25 flow-through times (FTT) to allow the flow to develop. One FTT represents the time required for fluid to pass through the entirety of the domain. The polynomial order was increased, filtering turned off, and each production run began. Data on fluid velocity, both instantaneous and averaged, along with the RMS terms were saved at specified time steps for every point in the domain. For the 7th-order case, these time steps occurred at intervals of 10 FTT, while the 11th-order case saved outputs every 1.25 FTT. Additionally, the velocity components at five selected points spaced throughout the domain, the volume-averaged velocity components, and the volume-averaged kinetic energy were computed each time step and saved in .csv format.

After 15 runs, 65 FTT for 7th order and ~19 for 11th order, we began computing and saving the average quantities required to calculate the terms in turbulent kinetic energy budget. These terms are the production, dissipation, turbulent convection, viscous diffusion, pressure strain, and pressure transport, and each is composed of averaged tensor quantities of velocity components and their gradients. In total, 230 and 57.5 FTT worth of data was collected for the 7th and 11th-order simulations, respectively.

The DNS were conducted with a Reynolds number of 3898 based on the sphere diameter of 1.305, an inlet flow rate of 1.2628, a density of one, and viscosity of 4.2293E-4. These values are all non-dimensionalized and were picked to produce a volume-averaged flow rate of one, which can be held constant in nek5000, and so that the only parameter that needed to be changed to adjust the Reynolds number was the viscosity.

In addition to the DNS, a series of LES were conducted at varying Reynolds numbers to compare how such variations affected the development of turbulence. One simulation was conducted with Re = 2445, another with Re = 4551, and the final simulation used Re = 5867. The Reynolds numbers were adjusted by varying the viscosity. These LES were conducted with a polynomial order of eight and the filtering parameter set to 5%. The number of processors and run-time characteristics were the same as those in the 7th-order DNS. Data was collected for 100 FTT for each LES. Later studies determined that similar results could be attained with only 50 FTT. While not quite as smooth, the 50-FTT results matched the behavior of the 100-FTT results well allowing future studies to make comparisons using less processing time.

5 RESULTS

The results presented in this section are interpolated across the large central gap between pebbles on the yz-plane. The profiles of the velocity magnitude taken from the DNS are presented in Figure 15:

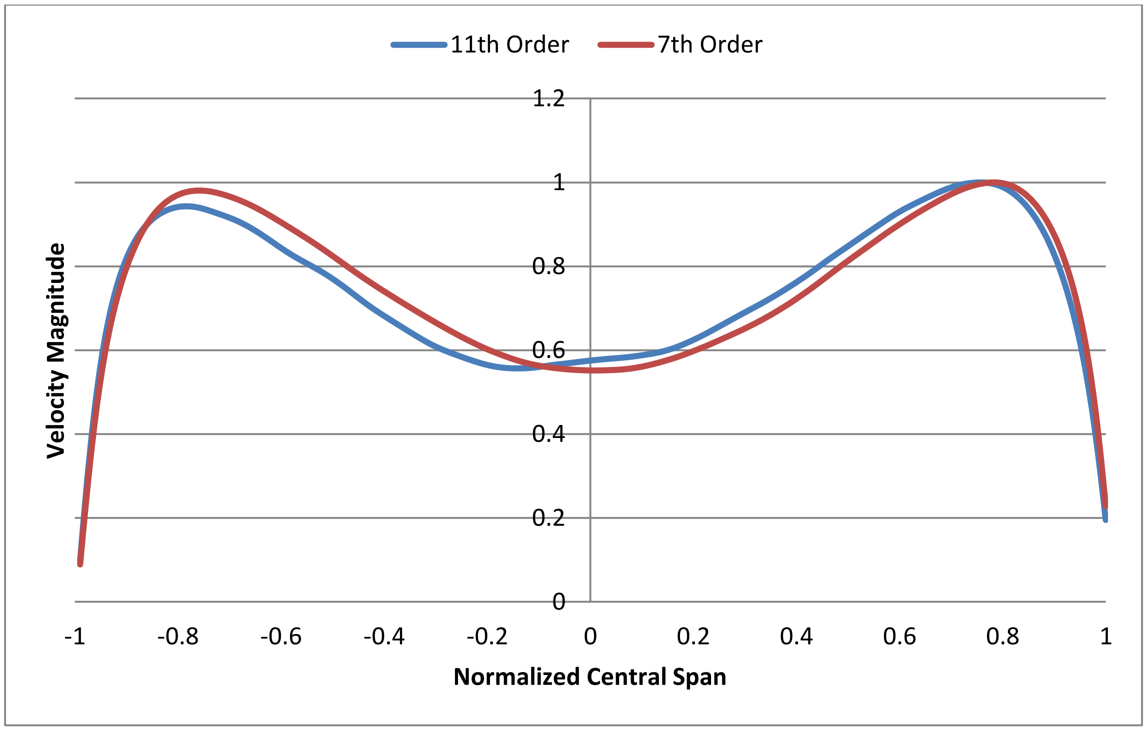


Figure 15. Velocity Magnitude Profiles from DNS

The horizontal axis has been normalized using the maximum width of the central gap and the vertical axis normalized using the maximum observed velocity for each interpolation. The 7th-order profile is much smoother due to the greater amount of averaging time. One should also note that the 11th-order result displays a small amount of asymmetry in the magnitude of the two peaks and the location of the central minimum. We have estimated the Kolmogorov scale at the center of the domain to be ~0.03 pebble diameters while the maximum distance between points in the 11th-order simulation is ~0.007 pebble diameters and 0.013 for the 7th-order simulation.

Figure 16 explores the development of this asymmetry by depicting velocity profiles computed in the LES runs as well as the quasi-DNS results published by NRG.

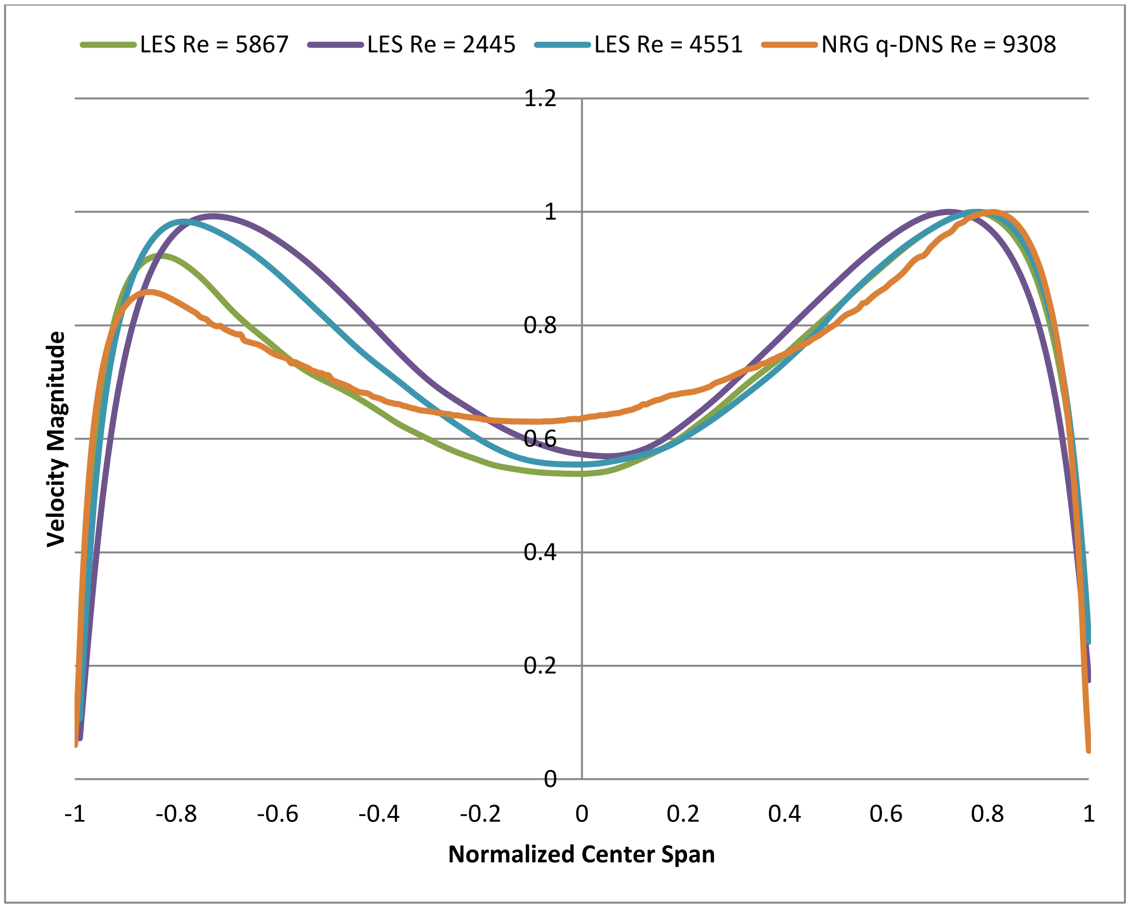
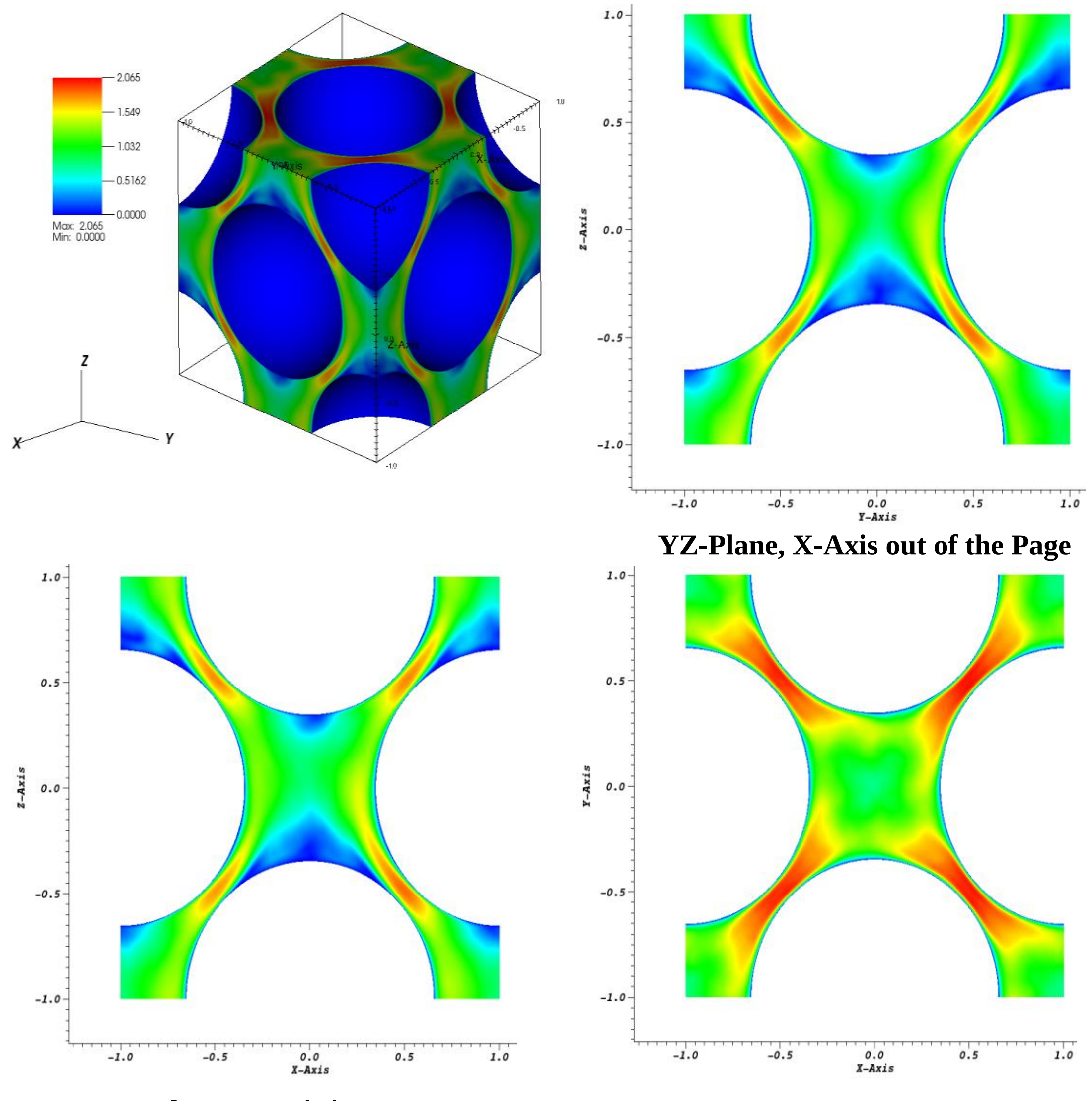


Figure 16. LES Velocity Magnitude Comparison

The Re=2445 profile is nearly symmetric across the gap, but as the Re increases, the flow pattern becomes more asymmetric. This trend supports the data produced by NRG.

This asymmetry may be explained in the following set of images in Figure 17, showing the velocity fields at various planes throughout the domain. These figures were taken from the 7th-order DNS, using an intermediate Re of 4551.



YZ-Plane, X-Axis out of the Page

XZ-Plane, Y-Axis into Page

XY-Plane, Z-Axis out of Page

Figure 17. Velocity Distributions Throughout Domain

The same slight asymmetries can be seen in these sections. The section from the xy-plane gives the clearest view of what appears to be a rotational flow pattern developing around the z-axis, the direction of flow. This rotational pattern is highlighted in the series of sections presented in Figure 18:

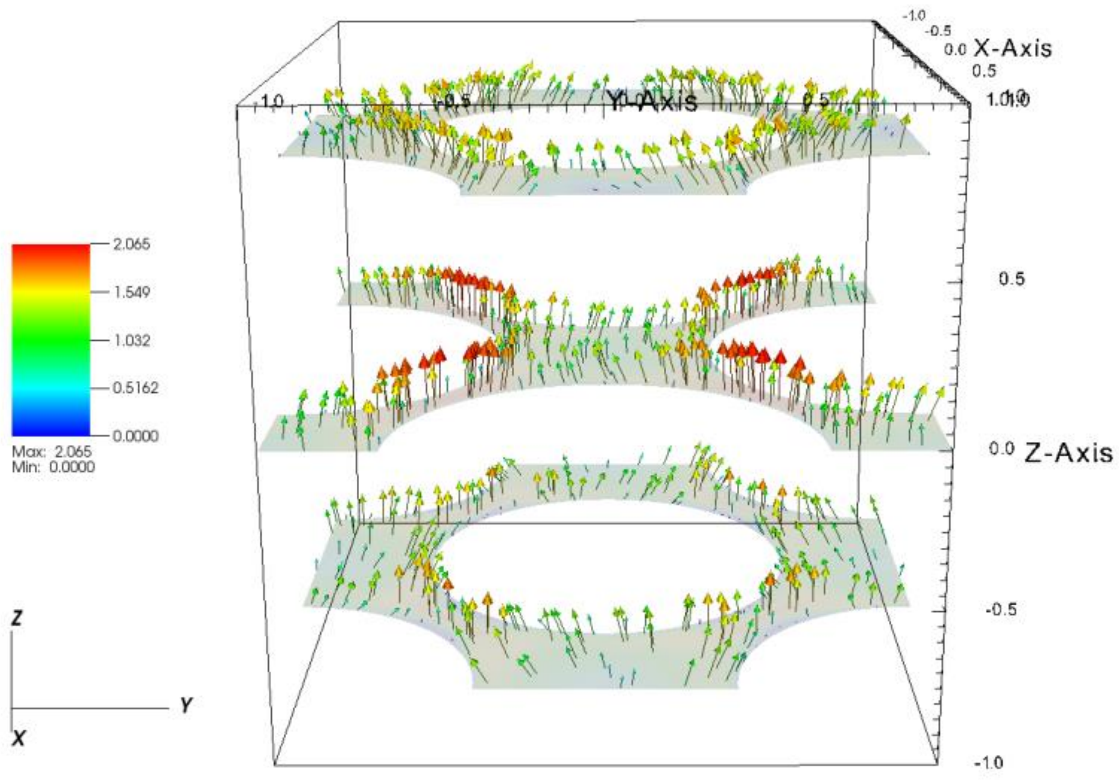


Figure 18. Velocity Vectors on Selected Planes

These velocity vector plots show how the flow passes around the entrance pebble in a relatively straight manner, but is rotating in the center gap and continues to rotate as it passes around the exit pebble. Rotational patterns were also experimentally observed in a similar geometry by Hassan and Dominguez-Ontinveros (2008). They measured little to no rotation in low-Re flows and full vortex formation in higher-Re flows.

As DNS attempts to resolve turbulence at its smallest scales, it is possible to calculate the turbulent kinetic energy budgets to a high degree of precision. The two studies, with varying levels of resolution as determined by the spectral polynomial order in each element, have provided insight into how fine a mesh is needed to capture the minute turbulent details. Unfortunately, during the computation of the turbulent terms, it became clear that the 7th-order results were under resolved resulting in their omission. Each profile is interpolated across the central gap of the YZ-plane. The horizontal axis is normalized using the maximum gap width so that the edges of the channel occur at positive and negative one. The profiles represent the sum of the uu, vv, and ww tensor components divided by two. The pressure terms displayed odd behavior and have been omitted until further verification can be done.

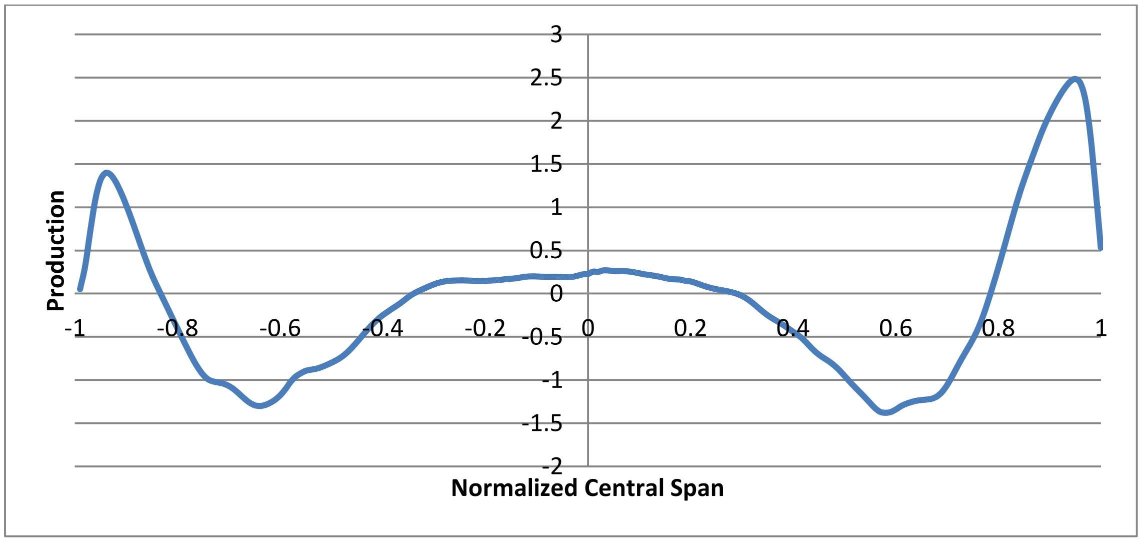


Figure 19. Turbulent Production

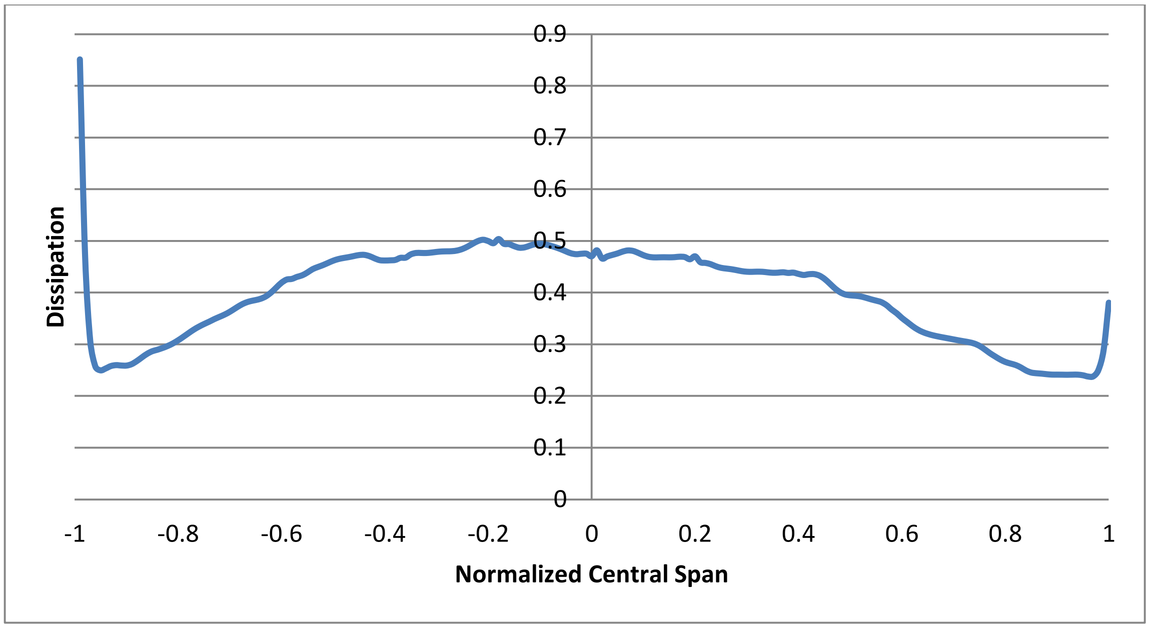


Figure 20. Turbulent Dissipation

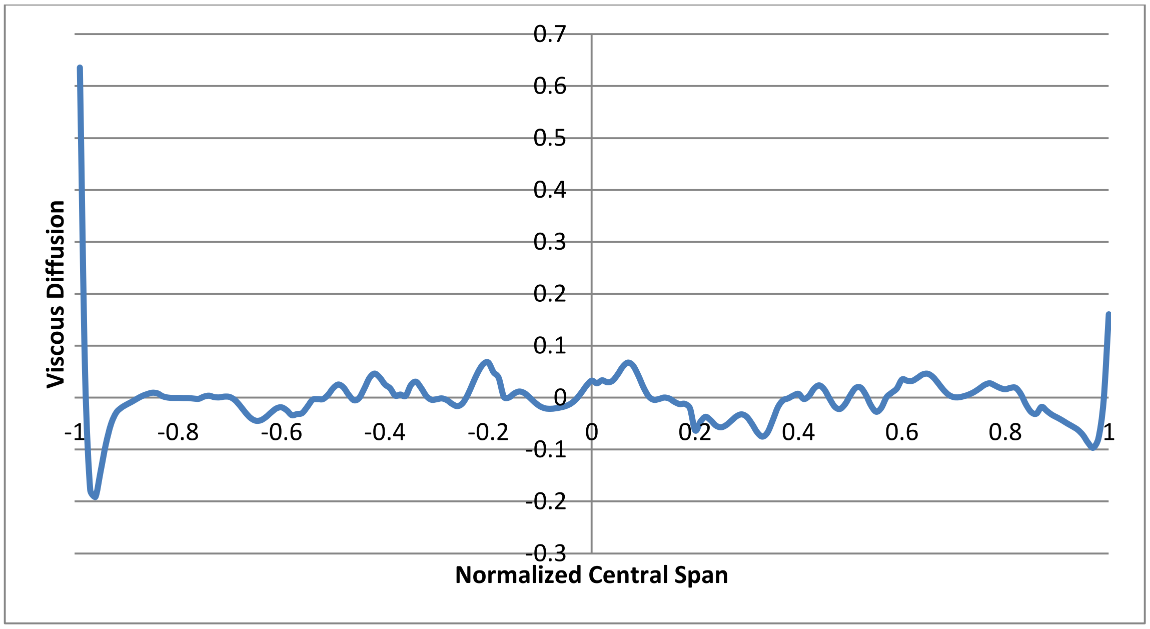


Figure 21. Viscous Diffusion

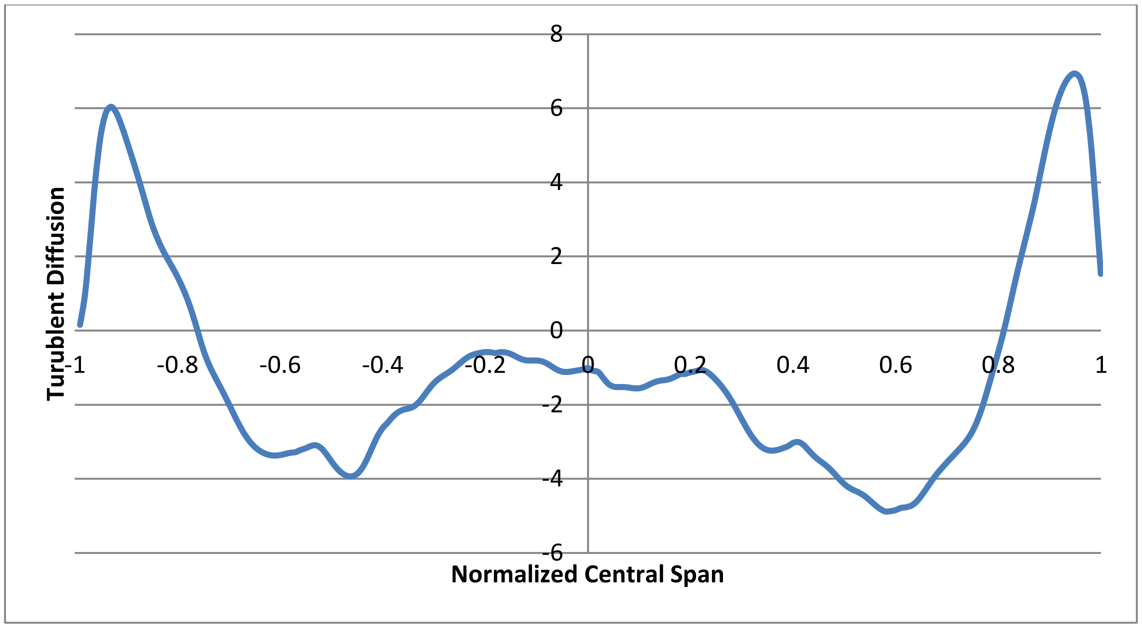


Figure 22. Turbulent Diffusion

Figure 19 depicts the production of turbulence within the center of the pebble bed. This profile bears some similarities to those seen in channel flow. As expected, the peak production occurs near the boundary layer. Figure 20 shows the turbulent dissipation. Again, the results make sense if the central gap is viewed as a type of channel. Peak dissipation occurs at the walls then rapidly decreases. Figure 21 depicts the viscous diffusion, which is essentially zero throughout most of the domain except for sharp peaks located near the walls where viscous forces come to dominate. Figure 22 presents the turbulent diffusion which appears to be the dominant effect within the central gap. Unlike channel flow, the two prominent peaks are both positive with the largest peak being closer to the wall. It is possible that the negative peak closest to the wall was missed by the interpolation routine, but the reasons for the differences are, as of yet, unknown.

Various statistical analyses were performed in an attempt to ensure the validity of the above data. The first identified 5 points within the domain, collected a long time history of the instantaneous velocities at these points, and computed the correlation coefficients between them. Point 1 was located at the center of the domain at (0,0,0) while Points 2-5 were located on the upper surface at (0,0.8,1), (0,-0.8,1), (0.8,0,1), and (-0.8,0,1), respectively.

The instantaneous velocity magnitude was recorded at each of these points for 100000 time steps. The correlation coefficient between each of the points was than calculated. The highest coefficient for the 7th-order simulation was 0.0521 with most falling around 0.01 or smaller. The 11th-order simulation had maximum correlation at 0.0862 with the rest falling between 0.055 and 0.01. These low values argue that the flow within the domain is uncorrelated on a point-to-point basis.

6 CONCLUSIONS

The primary goal of this study was to create a DNS data reference for fluid flow in a pebble bed. One of our main points of interest was determining why other groups were reporting asymmetric velocity profiles given the symmetric geometry and boundary conditions. We found that asymmetries in the velocity profile did arise and the magnitude of these asymmetries was linked to the flow's Reynolds number. These effects appear to be the result of a rotational flow pattern that is most pronounced in the central gap. The experimental data of Hassan supports the formation of vortices within these inter-pebble gaps. The trends we observed appear to match those found in the NRG study and future simulations will be run in an attempt to match their results directly.

The turbulent kinetic energy budgets show some similarities to those developed for channel flow, but further testing will have to be done to ensure their validity. The 7th-order velocity results appear to be useful and can be achieved in far less time than that required to complete the 11th-order simulation. However, the 7th-order turbulent budget terms appear to miss much of the information captured by the 11th-order simulation and, as a result, are not useful. Future studies will analyze the data in other parts of the domain focusing on the narrow inter-pebble gaps and the predicted stagnation zones.

REFERENCES

[1] LJ. Lommers, F. Shahrokhi, JA. Mayer III, FH. Southworth, AREVA Near-Term HTR Concept, *HTR 2010- Prague - 18-20 October 2010*

[2] C. H. Rycroft, T. Lind, S. Güntay, and A. Dehbi, Granular flows in pebble bed reactors: dust generation and scaling, proceedings of ICAPP 2012, Chicago, June 24–28, 2012

[3] A. Shams, F. Roelofs, EMJ. Komen, E. Baglietto, 2013. "Quasi-direct numerical simulation of a pebble bed configuration. Part I: Flow (velocity) field analysis ," *Nuclear Engineering and Design,* Vol. **263**, pp. 473–489

[4] Y. Hassan, EE. Dominguez-Ontiveros, 2008. "Flow visualization in a pebble bed reactor experiment using PIV and refractive index matching techniques," *Nuclear Engineering and Design,* Vol. **238**, pp. *3080-3085*

[5] SB. Pope, 2000. "Turbulent Flows", Cambridge: Cambridge University Press

[6] P. F. Fischer, 1997**. "**An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations," *J. of Comp. Phys.* Vol. **133**, pp. 84–101.

[7] M. O. Deville, P. F. Fischer, and E. H. Mund, 2002. "Higher Order Methods for Incompressible Fluid Flow," Cambridge: Cambridge Univ. Press.

[8] AV Obabko, PF Fischer, TJ Tautges, S Karabasov, VM Goloviznin, MA Zaytsev, VV Chudanov, VA Pervichko, AE Aksenova, 2011, "CFD validation in OECD/NEA t-junction benchmark", *Report ANL/NE-11/25*

[9] E Merzari, WD Pointer, P Fischer, 2013, "Numerical simulation and proper orthogonal decomposition of the flow in a counter-flow t-junction", *Journal of Fluids Engineering*, **135** (9), 091304

[10] P. Fischer and J. Mullen, 2001. Filter-based stabilization of spectral element methods. *Comptes rendus de l'Academie des sciences, Serie I Analyse numerique*, Vol. **332**, pp. 265–270.

[11] N. Kasagi, K. Horiuti, Y. Miyake, T. Miyauchi and Y. Nagano, 1990, "Establishment of the Direct Numerical Simulation Data Bases of Turbulent Transport Phenomena", 1990 - http://thtlab.jp

APPENDIX A: CODE

Averaging Routine:

```
      subroutine budgets_avg
c
c     This routine is based on avg_all and computes the terms of the
c     turbulent kinetic energy equation. These terms are dumped in a
c     series of .fld files and can be interpolated over a set of user-
c     defined points with the results outputted in .csv files.
c
c     E denotes the expected value operator and X,Y two
c     real valued random variables.
c
c     variances and covariances can be computed in a post-processing
step:
c
c        var(X)   := E(X^X) - E(X)*E(X)
c        cov(X,Y) := E(X*Y) - E(X)*E(Y)
c
c     Note: The E-operator is linear, in the sense that the expected
c           value is given by E(X) = 1/N * sum[ E(X)_i ], where E(X)_i
c           is the expected value of the sub-ensemble i (i=1...N).
c
      include 'SIZE'
      include 'TOTAL'
      include 'AVG'

      logical ifverbose
      integer icalld,i,j,k,e
      save    icalld
      data    icalld  /0/
      real    vis,rho

      parameter(interp=1)
      parameter(ninp=400)
      parameter(vis=4.386E-4)
      parameter(rho=1)


      common /ugradtens/
     $ dudx(lx1,ly1,lz1,lelt),
     $ dudy(lx1,ly1,lz1,lelt),
     $ dudz(lx1,ly1,lz1,lelt),
     $ dvdx(lx1,ly1,lz1,lelt),
     $ dvdy(lx1,ly1,lz1,lelt),
     $ dvdz(lx1,ly1,lz1,lelt),
     $ dwdx(lx1,ly1,lz1,lelt),
     $ dwdy(lx1,ly1,lz1,lelt),
     $ dwdz(lx1,ly1,lz1,lelt)

      common /dissipation/
     $ udxavg(lx1,ly1,lz1,lelt),udyavg(lx1,ly1,lz1,lelt),
     $ udzavg(lx1,ly1,lz1,lelt),vdxavg(lx1,ly1,lz1,lelt),
     $ vdyavg(lx1,ly1,lz1,lelt),vdzavg(lx1,ly1,lz1,lelt),
     $ wdxavg(lx1,ly1,lz1,lelt),wdyavg(lx1,ly1,lz1,lelt),
     $ wdzavg(lx1,ly1,lz1,lelt),udxrms(lx1,ly1,lz1,lelt),
     $ udyrms(lx1,ly1,lz1,lelt),udzrms(lx1,ly1,lz1,lelt),
     $ vdxrms(lx1,ly1,lz1,lelt),vdyrms(lx1,ly1,lz1,lelt),
     $ vdzrms(lx1,ly1,lz1,lelt),wdxrms(lx1,ly1,lz1,lelt),
     $ wdyrms(lx1,ly1,lz1,lelt),wdzrms(lx1,ly1,lz1,lelt),
     $ uvdxms(lx1,ly1,lz1,lelt),uvdyms(lx1,ly1,lz1,lelt),
     $ uvdzms(lx1,ly1,lz1,lelt),uwdxms(lx1,ly1,lz1,lelt),
     $ uwdyms(lx1,ly1,lz1,lelt),uwdzms(lx1,ly1,lz1,lelt),
     $ vwdxms(lx1,ly1,lz1,lelt),vwdyms(lx1,ly1,lz1,lelt),
     $ vwdzms(lx1,ly1,lz1,lelt),eps_uu(lx1,ly1,lz1,lelt),
     $ eps_vv(lx1,ly1,lz1,lelt),eps_ww(lx1,ly1,lz1,lelt),
     $ eps_uv(lx1,ly1,lz1,lelt),eps_uw(lx1,ly1,lz1,lelt),
     $ eps_vw(lx1,ly1,lz1,lelt)

      common /production/
     $ prd_uu(lx1,ly1,lz1,lelt),prd_vv(lx1,ly1,lz1,lelt),
     $ prd_ww(lx1,ly1,lz1,lelt),prd_uv(lx1,ly1,lz1,lelt),
     $ prd_uw(lx1,ly1,lz1,lelt),prd_vw(lx1,ly1,lz1,lelt)

      common /pdiffusion/
     $ upavg(lx1,ly1,lz1,lelt),vpavg(lx1,ly1,lz1,lelt),
     $ wpavg(lx1,ly1,lz1,lelt),pdf_uu(lx1,ly1,lz1,lelt),
     $ pdf_vv(lx1,ly1,lz1,lelt),pdf_ww(lx1,ly1,lz1,lelt),
     $ pdf_uv(lx1,ly1,lz1,lelt),pdf_uw(lx1,ly1,lz1,lelt),
     $ pdf_vw(lx1,ly1,lz1,lelt)


      common /vdiffusion/
     $ vdf_uu(lx1,ly1,lz1,lelt),vdf_vv(lx1,ly1,lz1,lelt),
     $ vdf_ww(lx1,ly1,lz1,lelt),vdf_uv(lx1,ly1,lz1,lelt),
     $ vdf_uw(lx1,ly1,lz1,lelt),vdf_vw(lx1,ly1,lz1,lelt)

      common /tdiffusion/
     $ uuuavg(lx1,ly1,lz1,lelt),uuvavg(lx1,ly1,lz1,lelt),
     $ uuwavg(lx1,ly1,lz1,lelt),uvvavg(lx1,ly1,lz1,lelt),
     $ uwwavg(lx1,ly1,lz1,lelt),uvwavg(lx1,ly1,lz1,lelt),
     $ vvvavg(lx1,ly1,lz1,lelt),vvwavg(lx1,ly1,lz1,lelt),
     $ vwwavg(lx1,ly1,lz1,lelt),wwwavg(lx1,ly1,lz1,lelt),
     $ tdf_uu(lx1,ly1,lz1,lelt),tdf_vv(lx1,ly1,lz1,lelt),
     $ tdf_ww(lx1,ly1,lz1,lelt),tdf_uv(lx1,ly1,lz1,lelt),
     $ tdf_uw(lx1,ly1,lz1,lelt),tdf_vw(lx1,ly1,lz1,lelt)

       common /pstrain/
     $ pudxavg(lx1,ly1,lz1,lelt),pudyavg(lx1,ly1,lz1,lelt),
     $ pudzavg(lx1,ly1,lz1,lelt),pvdxavg(lx1,ly1,lz1,lelt),
     $ pvdyavg(lx1,ly1,lz1,lelt),pvdzavg(lx1,ly1,lz1,lelt),
     $ pwdxavg(lx1,ly1,lz1,lelt),pwdyavg(lx1,ly1,lz1,lelt),
          $ pwdzavg(lx1,ly1,lz1,lelt),pst_uu(lx1,ly1,lz1,lelt),
          $ pst_vv(lx1,ly1,lz1,lelt),pst_ww(lx1,ly1,lz1,lelt),
          $ pst_uv(lx1,ly1,lz1,lelt),pst_uw(lx1,ly1,lz1,lelt),
          $ pst_vw(lx1,ly1,lz1,lelt)

            common /plane/
          $ eps_uu_avg(lx1,ly1,lz1,lelt),eps_vv_avg(lx1,ly1,lz1,lelt),
          $ eps_ww_avg(lx1,ly1,lz1,lelt),eps_uv_avg(lx1,ly1,lz1,lelt),
          $ eps_uw_avg(lx1,ly1,lz1,lelt),eps_vw_avg(lx1,ly1,lz1,lelt),
          $ prd_uu_avg(lx1,ly1,lz1,lelt),prd_vv_avg(lx1,ly1,lz1,lelt),
          $ prd_ww_avg(lx1,ly1,lz1,lelt),prd_uv_avg(lx1,ly1,lz1,lelt),
          $ prd_uw_avg(lx1,ly1,lz1,lelt),prd_vw_avg(lx1,ly1,lz1,lelt),
          $ tdf_uu_avg(lx1,ly1,lz1,lelt),tdf_vv_avg(lx1,ly1,lz1,lelt),
          $ tdf_ww_avg(lx1,ly1,lz1,lelt),tdf_uv_avg(lx1,ly1,lz1,lelt),
          $ tdf_uw_avg(lx1,ly1,lz1,lelt),tdf_vw_avg(lx1,ly1,lz1,lelt),
          $ pdf_uu_avg(lx1,ly1,lz1,lelt),pdf_vv_avg(lx1,ly1,lz1,lelt),
          $ pdf_ww_avg(lx1,ly1,lz1,lelt),pdf_uv_avg(lx1,ly1,lz1,lelt),
          $ pdf_uw_avg(lx1,ly1,lz1,lelt),pdf_vw_avg(lx1,ly1,lz1,lelt),
          $ vdf_uu_avg(lx1,ly1,lz1,lelt),vdf_vv_avg(lx1,ly1,lz1,lelt),
          $ vdf_ww_avg(lx1,ly1,lz1,lelt),vdf_uv_avg(lx1,ly1,lz1,lelt),
          $ vdf_uw_avg(lx1,ly1,lz1,lelt),vdf_vw_avg(lx1,ly1,lz1,lelt),
          $ pst_uu_avg(lx1,ly1,lz1,lelt),pst_vv_avg(lx1,ly1,lz1,lelt),
          $ pst_ww_avg(lx1,ly1,lz1,lelt),pst_uv_avg(lx1,ly1,lz1,lelt),
          $ pst_uw_avg(lx1,ly1,lz1,lelt),pst_vw_avg(lx1,ly1,lz1,lelt),
          $ u_pl_avg(lx1,ly1,lz1,lelt),udy_pl_avg(lx1,ly1,lz1,lelt),
          $ v_pl_avg(lx1,ly1,lz1,lelt),vdy_pl_avg(lx1,ly1,lz1,lelt),
          $ w_pl_avg(lx1,ly1,lz1,lelt),wdy_pl_avg(lx1,ly1,lz1,lelt)


             common /scratch_interp/
          $  xyz1(3,ninp),vel1(3,ninp),dvel(3,ninp),
          $  eps1(3,ninp),eps2(3,ninp),prd1(3,ninp),
          $  prd2(3,ninp),tdf1(3,ninp),tdf2(3,ninp),
          $  pdf1(3,ninp),pdf2(3,ninp),pst1(3,ninp),
          $  pst2(3,ninp),vdf1(3,ninp),vdf2(3,ninp)


           if (ax1.ne.lx1 .or. ay1.ne.ly1 .or. az1.ne.lz1) then
              if(nid.eq.0) write(6,*)
          $     'ABORT: wrong size of ax1,ay1,az1 in avg_all(), check
     SIZEu!'
              call exitt
           endif
           if (ax2.ne.lx2 .or. ay2.ne.ay2 .or. az2.ne.lz2) then
              if(nid.eq.0) write(6,*)
          $     'ABORT: wrong size of ax2,ay2,az2 in avg_all(), check
     SIZEu!'
              call exitt
           endif

           ntot  = nx1*ny1*nz1*nelv
           ntott = nx1*ny1*nz1*nelt
           nto2  = nx2*ny2*nz2*nelv
```

```
! initialization
if (icalld.eq.0) then
   icalld = icalld + 1
   atime  = 0.
   timel  = time

   call rzero(uavg,ntot)
   call rzero(vavg,ntot)
   call rzero(wavg,ntot)
   call rzero(pavg,nto2)
   call rzero(udxavg,ntot)
   call rzero(udyavg,ntot)
   call rzero(udzavg,ntot)
   call rzero(vdxavg,ntot)
   call rzero(vdyavg,ntot)
   call rzero(vdzavg,ntot)
   call rzero(wdxavg,ntot)
   call rzero(wdyavg,ntot)
   call rzero(wdzavg,ntot)
   call rzero(udxrms,ntot)
   call rzero(udyrms,ntot)
   call rzero(udzrms,ntot)
   call rzero(vdxrms,ntot)
   call rzero(vdyrms,ntot)
   call rzero(vdzrms,ntot)
   call rzero(wdxrms,ntot)
   call rzero(wdyrms,ntot)
   call rzero(wdzrms,ntot)
   call rzero(upavg,ntot)
   call rzero(vpavg,ntot)
   call rzero(wpavg,ntot)
   call rzero(uuuavg,ntot)
   call rzero(uuvavg,ntot)
   call rzero(uuwavg,ntot)
   call rzero(uvvavg,ntot)
   call rzero(uvwavg,ntot)
   call rzero(uwwavg,ntot)
   call rzero(vvvavg,ntot)
   call rzero(vvwavg,ntot)
   call rzero(vwwavg,ntot)
   call rzero(wwwavg,ntot)
   do i = 1,ldimt
      call rzero(tavg(1,1,1,1,i),ntott)
   enddo
   call rzero(urms,ntot)
   call rzero(vrms,ntot)
   call rzero(wrms,ntot)
   call rzero(prms,nto2)
   do i = 1,ldimt
      call rzero(trms(1,1,1,1,i),ntott)
   enddo
   call rzero(vwms,ntot)
   call rzero(wums,ntot)
   call rzero(uvms,ntot)
```

```
      endif

      dtime = time  - timel
      atime = atime + dtime

      ! dump freq
      iastep = param(68)
      if  (iastep.eq.0) iastep=param(15)   ! same as iostep
      if  (iastep.eq.0) iastep=500

      ifverbose=.false.
      if (istep.le.10) ifverbose=.true.
      if  (mod(istep,iastep).eq.0) ifverbose=.true.

      if (atime.ne.0..and.dtime.ne.0.) then
         if(nid.eq.0) write(6,*) 'Compute statistics ...'
         beta  = dtime/atime
         alpha = 1.-beta

c---------------------------------------------------------------------
c     Required Variables
c---------------------------------------------------------------------

         ! compute averages E(X)
         call avg1    (uavg,vx,alpha,beta,ntot ,'um  ',ifverbose)
         call avg1    (vavg,vy,alpha,beta,ntot ,'vm  ',ifverbose)
         call avg1    (wavg,vz,alpha,beta,ntot ,'wm  ',ifverbose)
         call avg1    (pavg,pr,alpha,beta,nto2 ,'prm ',ifverbose)
         call avg1    (tavg,t ,alpha,beta,ntott,'tm  ',ifverbose)
         call gradm1(dudx,dudy,dudz,vx)
         call gradm1(dvdx,dvdy,dvdz,vy)
         call gradm1(dwdx,dwdy,dwdz,vz)

         do i = 2,ldimt
            call avg1 (tavg(1,1,1,1,i),t(1,1,1,1,i),alpha,beta,
     &                 ntott,'psav',ifverbose)
         enddo

         ! compute averages E(X^2)
         call avg2    (urms,vx,alpha,beta,ntot ,'ums ',ifverbose)
         call avg2    (vrms,vy,alpha,beta,ntot ,'vms ',ifverbose)
         call avg2    (wrms,vz,alpha,beta,ntot ,'wms ',ifverbose)
         call avg2    (prms,pr,alpha,beta,nto2 ,'prms',ifverbose)
         call avg2    (trms,t ,alpha,beta,ntott,'tms ',ifverbose)
         do i = 2,ldimt
            call avg2 (trms(1,1,1,1,i),t(1,1,1,1,i),alpha,beta,
     &                 ntott,'psms',ifverbose)
         enddo

         ! compute averages E(X*Y) (for now just for the velocities)
         call avg3    (uvms,vx,vy,alpha,beta,ntot,'uvm ',ifverbose)
         call avg3    (vwms,vy,vz,alpha,beta,ntot,'vwm ',ifverbose)
         call avg3    (wums,vz,vx,alpha,beta,ntot,'wum ',ifverbose)
```

```
c -----  Dissipation  ----------------------------------------------
         call avg1(udxavg,dudx,alpha,beta,ntot ,'dudxm  ',ifverbose)
         call avg1(udyavg,dudy,alpha,beta,ntot ,'dudym  ',ifverbose)
         call avg1(udzavg,dudz,alpha,beta,ntot ,'dudzm  ',ifverbose)
         call avg1(vdxavg,dvdx,alpha,beta,ntot ,'dvdxm  ',ifverbose)
         call avg1(vdyavg,dvdy,alpha,beta,ntot ,'dvdym  ',ifverbose)
         call avg1(vdzavg,dvdz,alpha,beta,ntot ,'dvdzm  ',ifverbose)
         call avg1(wdxavg,dwdx,alpha,beta,ntot ,'dwdxm  ',ifverbose)
         call avg1(wdyavg,dwdy,alpha,beta,ntot ,'dwdym  ',ifverbose)
         call avg1(wdzavg,dwdz,alpha,beta,ntot ,'dwdzm  ',ifverbose)
         call avg2(udxrms,dudx,alpha,beta,ntot ,'dudxms ',ifverbose)
         call avg2(udyrms,dudy,alpha,beta,ntot ,'dudyms ',ifverbose)
         call avg2(udzrms,dudz,alpha,beta,ntot ,'dudzms ',ifverbose)
         call avg2(vdxrms,dvdx,alpha,beta,ntot ,'dvdxms ',ifverbose)
         call avg2(vdyrms,dvdy,alpha,beta,ntot ,'dvdyms ',ifverbose)
         call avg2(vdzrms,dvdz,alpha,beta,ntot ,'dvdzms ',ifverbose)
         call avg2(wdxrms,dwdx,alpha,beta,ntot ,'dwdxms ',ifverbose)
         call avg2(wdyrms,dwdy,alpha,beta,ntot ,'dwdyms ',ifverbose)
         call avg2(wdzrms,dwdz,alpha,beta,ntot ,'dwdzms ',ifverbose)
         call avg3(uvdxms,dudx,dvdx,alpha,beta,ntot,'uvxms
',ifverbose)
         call avg3(uvdyms,dudy,dvdy,alpha,beta,ntot,'uvyms
',ifverbose)
         call avg3(uvdzms,dudz,dvdz,alpha,beta,ntot,'uvzms
',ifverbose)
         call avg3(uwdxms,dudx,dwdx,alpha,beta,ntot,'uwxms
',ifverbose)
         call avg3(uwdyms,dudy,dwdy,alpha,beta,ntot,'uwyms
',ifverbose)
         call avg3(uwdzms,dudz,dwdz,alpha,beta,ntot,'uwzms
',ifverbose)
         call avg3(vwdxms,dvdx,dwdx,alpha,beta,ntot,'vwxms
',ifverbose)
         call avg3(vwdyms,dvdy,dwdy,alpha,beta,ntot,'vwyms
',ifverbose)
         call avg3(vwdzms,dvdz,dwdz,alpha,beta,ntot,'vwzms
',ifverbose)
c -----  Pressure Diffusion  ------------------------------------------
         call avg3(upavg,pr,vx,alpha,beta,nto2,'upavg ',ifverbose)
         call avg3(vpavg,pr,vy,alpha,beta,nto2,'vpavg ',ifverbose)
         call avg3(wpavg,pr,vz,alpha,beta,nto2,'wpavg ',ifverbose)
c -----  Turbulent Diffusion  -----------------------------------------
         call avg4(uuuavg,vx,vx,vx,alpha,beta,ntot,'uuuavg',ifverbose)
         call avg4(uuvavg,vx,vx,vy,alpha,beta,ntot,'uuvavg',ifverbose)
         call avg4(uuwavg,vx,vx,vz,alpha,beta,ntot,'uuwavg',ifverbose)
         call avg4(uvvavg,vx,vy,vy,alpha,beta,ntot,'uvvavg',ifverbose)
         call avg4(uwwavg,vx,vz,vz,alpha,beta,ntot,'uwwavg',ifverbose)
         call avg4(uvwavg,vx,vy,vz,alpha,beta,ntot,'uvwavg',ifverbose)
         call avg4(vvvavg,vy,vy,vy,alpha,beta,ntot,'vvvavg',ifverbose)
         call avg4(vvwavg,vy,vy,vz,alpha,beta,ntot,'vvwavg',ifverbose)
         call avg4(vwwavg,vy,vz,vz,alpha,beta,ntot,'vwwavg',ifverbose)
         call avg4(wwwavg,vz,vz,vz,alpha,beta,ntot,'wwwavg',ifverbose)
c -----  Pressure Strain  ---------------------------------------------
```

```
         call avg3(pudxavg,pr,dudx,alpha,beta,ntot,'pudxavg',ifverbose)
         call avg3(pudyavg,pr,dudy,alpha,beta,ntot,'pudyavg',ifverbose)
         call avg3(pudzavg,pr,dudz,alpha,beta,ntot,'pudzavg',ifverbose)
         call avg3(pvdxavg,pr,dvdx,alpha,beta,ntot,'pvdxavg',ifverbose)
         call avg3(pvdyavg,pr,dvdy,alpha,beta,ntot,'pvdyavg',ifverbose)
         call avg3(pvdzavg,pr,dvdz,alpha,beta,ntot,'pvdzavg',ifverbose)
         call avg3(pwdxavg,pr,dwdx,alpha,beta,ntot,'pwdxavg',ifverbose)
         call avg3(pwdyavg,pr,dwdy,alpha,beta,ntot,'pwdyavg',ifverbose)
         call avg3(pwdzavg,pr,dwdz,alpha,beta,ntot,'pwdzavg',ifverbose)
      endif


c -----  Output .fld files for visualization  -------------------------
----
         call outpost2(uavg,vavg,wavg,pavg,tavg,ldimt,'avg')
         call outpost2(urms,vrms,wrms,pavg,tavg,ldimt,'rms')
         call outpost2(uvms,wums,vwms,pavg,tavg,ldimt,'rm2')
         call outpost2(udxavg,udyavg,udzavg,pavg,tavg,ldimt,'uda')
         call outpost2(vdxavg,vdyavg,vdzavg,pavg,tavg,ldimt,'vda')
         call outpost2(wdxavg,wdyavg,wdzavg,pavg,tavg,ldimt,'wda')
         call outpost2(udxrms,udyrms,udzrms,pavg,tavg,ldimt,'udr')
         call outpost2(vdxrms,vdyrms,vdzrms,pavg,tavg,ldimt,'vdr')
         call outpost2(wdxrms,wdyrms,wdzrms,pavg,tavg,ldimt,'wdr')
         call outpost2(uvdxms,uvdyms,uvdzms,pavg,tavg,ldimt,'uvd')
         call outpost2(uwdxms,uwdyms,uwdzms,pavg,tavg,ldimt,'uwd')
         call outpost2(vwdxms,vwdyms,vwdzms,pavg,tavg,ldimt,'vwd')
         call outpost2(upavg,vpavg,wpavg,pavg,tavg,ldimt,'vpa')
         call outpost2(uuuavg,uuvavg,uuwavg,pavg,tavg,ldimt,'uuu')
         call outpost2(uvvavg,uvwavg,uwwavg,pavg,tavg,ldimt,'uvv')
         call outpost2(vvvavg,vvwavg,vwwavg,pavg,tavg,ldimt,'vvv')
         call outpost2(wwwavg,wwwavg,wwwavg,pavg,tavg,ldimt,'www')
         call outpost2(pudxavg,pudyavg,pudzavg,pavg,tavg,ldimt,'pud')
         call outpost2(pvdxavg,pvdyavg,pvdzavg,pavg,tavg,ldimt,'pvd')
         call outpost2(pwdxavg,pwdyavg,pwdzavg,pavg,tavg,ldimt,'pwd')


      return
      end


c--------------------------------------------------------------------
-
      subroutine avg4(avg,f,g,h,alpha,beta,n,name,ifverbose)
      include 'SIZE'
      include 'TSTEP'
c
      real avg(n),f(n),g(n),h(n)
      character*4 name
      logical ifverbose
c
      do k=1,n
         avg(k) = alpha*avg(k) + beta*f(k)*g(k)*h(k)
      enddo
c
```

```
      if (ifverbose) then
         avgmax = glmax(avg,n)
         avgmin = glmin(avg,n)
         if (nid.eq.0) write(6,1) istep,time,avgmin,avgmax
     $                           ,alpha,beta,name
    1    format(i9,1p5e13.5,1x,a4,' av4mnx')
      endif
c
      return
      end
```

Term Calculation Routine:

```
      subroutine budget_terms
c
c     This subroutine takes the outputs from budget_avg and computes
the
c     tensor components of each turbulent kinetic energy term before
c     outputting them in both tabular .csv format and graphical .fld
c     format
c
c     E denotes the expected value operator and X,Y two
c     real valued random variables.
c
```

```
c     variances and covariances can be computed in a post-processing
step:
c
c        var(X)   := E(X^X) - E(X)*E(X)
c        cov(X,Y) := E(X*Y) - E(X)*E(Y)
c
c     Note: The E-operator is linear, in the sense that the expected
c           value is given by E(X) = 1/N * sum[ E(X)_i ], where E(X)_i
c           is the expected value of the sub-ensemble i (i=1...N).
c
      include 'SIZE'
      include 'TOTAL'
      include 'AVG'

      logical ifverbose
      integer icalld,i,j,k,e,ii
      save    icalld
      data    icalld  /0/
      real    vis,rho
      character namef*7

      parameter(interp=1)
      parameter(ninp=400)
      parameter(vis=4.2293E-4)
      parameter(rho=1)


      common /ugradtens/
     $ dudx(lx1,ly1,lz1,lelt),
     $ dudy(lx1,ly1,lz1,lelt),
     $ dudz(lx1,ly1,lz1,lelt),
     $ dvdx(lx1,ly1,lz1,lelt),
     $ dvdy(lx1,ly1,lz1,lelt),
     $ dvdz(lx1,ly1,lz1,lelt),
     $ dwdx(lx1,ly1,lz1,lelt),
     $ dwdy(lx1,ly1,lz1,lelt),
     $ dwdz(lx1,ly1,lz1,lelt)

      common /dissipation/
     $ udxavg(lx1,ly1,lz1,lelt),udyavg(lx1,ly1,lz1,lelt),
     $ udzavg(lx1,ly1,lz1,lelt),vdxavg(lx1,ly1,lz1,lelt),
     $ vdyavg(lx1,ly1,lz1,lelt),vdzavg(lx1,ly1,lz1,lelt),
     $ wdxavg(lx1,ly1,lz1,lelt),wdyavg(lx1,ly1,lz1,lelt),
     $ wdzavg(lx1,ly1,lz1,lelt),udxrms(lx1,ly1,lz1,lelt),
     $ udyrms(lx1,ly1,lz1,lelt),udzrms(lx1,ly1,lz1,lelt),
     $ vdxrms(lx1,ly1,lz1,lelt),vdyrms(lx1,ly1,lz1,lelt),
     $ vdzrms(lx1,ly1,lz1,lelt),wdxrms(lx1,ly1,lz1,lelt),
     $ wdyrms(lx1,ly1,lz1,lelt),wdzrms(lx1,ly1,lz1,lelt),
     $ uvdxms(lx1,ly1,lz1,lelt),uvdyms(lx1,ly1,lz1,lelt),
     $ uvdzms(lx1,ly1,lz1,lelt),uwdxms(lx1,ly1,lz1,lelt),
     $ uwdyms(lx1,ly1,lz1,lelt),uwdzms(lx1,ly1,lz1,lelt),
     $ vwdxms(lx1,ly1,lz1,lelt),vwdyms(lx1,ly1,lz1,lelt),
     $ vwdzms(lx1,ly1,lz1,lelt),eps_uu(lx1,ly1,lz1,lelt),
     $ eps_vv(lx1,ly1,lz1,lelt),eps_ww(lx1,ly1,lz1,lelt),
     $ eps_uv(lx1,ly1,lz1,lelt),eps_uw(lx1,ly1,lz1,lelt),
     $ eps_vw(lx1,ly1,lz1,lelt)

      common /production/
     $ prd_uu(lx1,ly1,lz1,lelt),prd_vv(lx1,ly1,lz1,lelt),
     $ prd_ww(lx1,ly1,lz1,lelt),prd_uv(lx1,ly1,lz1,lelt),
     $ prd_uw(lx1,ly1,lz1,lelt),prd_vw(lx1,ly1,lz1,lelt)

      common /pdiffusion/
     $ upavg(lx1,ly1,lz1,lelt),vpavg(lx1,ly1,lz1,lelt),
     $ wpavg(lx1,ly1,lz1,lelt),pdf_uu(lx1,ly1,lz1,lelt),
     $ pdf_vv(lx1,ly1,lz1,lelt),pdf_ww(lx1,ly1,lz1,lelt),
     $ pdf_uv(lx1,ly1,lz1,lelt),pdf_uw(lx1,ly1,lz1,lelt),
     $ pdf_vw(lx1,ly1,lz1,lelt)


      common /vdiffusion/
     $ vdf_uu(lx1,ly1,lz1,lelt),vdf_vv(lx1,ly1,lz1,lelt),
     $ vdf_ww(lx1,ly1,lz1,lelt),vdf_uv(lx1,ly1,lz1,lelt),
     $ vdf_uw(lx1,ly1,lz1,lelt),vdf_vw(lx1,ly1,lz1,lelt)

      common /tdiffusion/
     $ uuuavg(lx1,ly1,lz1,lelt),uuvavg(lx1,ly1,lz1,lelt),
     $ uuwavg(lx1,ly1,lz1,lelt),uvvavg(lx1,ly1,lz1,lelt),
     $ uwwavg(lx1,ly1,lz1,lelt),uvwavg(lx1,ly1,lz1,lelt),
     $ vvvavg(lx1,ly1,lz1,lelt),vvwavg(lx1,ly1,lz1,lelt),
     $ vwwavg(lx1,ly1,lz1,lelt),wwwavg(lx1,ly1,lz1,lelt),
     $ tdf_uu(lx1,ly1,lz1,lelt),tdf_vv(lx1,ly1,lz1,lelt),
     $ tdf_ww(lx1,ly1,lz1,lelt),tdf_uv(lx1,ly1,lz1,lelt),
     $ tdf_uw(lx1,ly1,lz1,lelt),tdf_vw(lx1,ly1,lz1,lelt)

       common /pstrain/
     $ pudxavg(lx1,ly1,lz1,lelt),pudyavg(lx1,ly1,lz1,lelt),
     $ pudzavg(lx1,ly1,lz1,lelt),pvdxavg(lx1,ly1,lz1,lelt),
     $ pvdyavg(lx1,ly1,lz1,lelt),pvdzavg(lx1,ly1,lz1,lelt),
     $ pwdxavg(lx1,ly1,lz1,lelt),pwdyavg(lx1,ly1,lz1,lelt),
     $ pwdzavg(lx1,ly1,lz1,lelt),pst_uu(lx1,ly1,lz1,lelt),
     $ pst_vv(lx1,ly1,lz1,lelt),pst_ww(lx1,ly1,lz1,lelt),
     $ pst_uv(lx1,ly1,lz1,lelt),pst_uw(lx1,ly1,lz1,lelt),
     $ pst_vw(lx1,ly1,lz1,lelt)

       common /plane/
     $ eps_uu_avg(lx1,ly1,lz1,lelt),eps_vv_avg(lx1,ly1,lz1,lelt),
     $ eps_ww_avg(lx1,ly1,lz1,lelt),eps_uv_avg(lx1,ly1,lz1,lelt),
     $ eps_uw_avg(lx1,ly1,lz1,lelt),eps_vw_avg(lx1,ly1,lz1,lelt),
     $ prd_uu_avg(lx1,ly1,lz1,lelt),prd_vv_avg(lx1,ly1,lz1,lelt),
     $ prd_ww_avg(lx1,ly1,lz1,lelt),prd_uv_avg(lx1,ly1,lz1,lelt),
     $ prd_uw_avg(lx1,ly1,lz1,lelt),prd_vw_avg(lx1,ly1,lz1,lelt),
     $ tdf_uu_avg(lx1,ly1,lz1,lelt),tdf_vv_avg(lx1,ly1,lz1,lelt),
     $ tdf_ww_avg(lx1,ly1,lz1,lelt),tdf_uv_avg(lx1,ly1,lz1,lelt),
     $ tdf_uw_avg(lx1,ly1,lz1,lelt),tdf_vw_avg(lx1,ly1,lz1,lelt),
     $ pdf_uu_avg(lx1,ly1,lz1,lelt),pdf_vv_avg(lx1,ly1,lz1,lelt),
     $ pdf_ww_avg(lx1,ly1,lz1,lelt),pdf_uv_avg(lx1,ly1,lz1,lelt),
     $ pdf_uw_avg(lx1,ly1,lz1,lelt),pdf_vw_avg(lx1,ly1,lz1,lelt),
          $ vdf_uu_avg(lx1,ly1,lz1,lelt),vdf_vv_avg(lx1,ly1,lz1,lelt),
          $ vdf_ww_avg(lx1,ly1,lz1,lelt),vdf_uv_avg(lx1,ly1,lz1,lelt),
          $ vdf_uw_avg(lx1,ly1,lz1,lelt),vdf_vw_avg(lx1,ly1,lz1,lelt),
          $ pst_uu_avg(lx1,ly1,lz1,lelt),pst_vv_avg(lx1,ly1,lz1,lelt),
          $ pst_ww_avg(lx1,ly1,lz1,lelt),pst_uv_avg(lx1,ly1,lz1,lelt),
          $ pst_uw_avg(lx1,ly1,lz1,lelt),pst_vw_avg(lx1,ly1,lz1,lelt),
          $ u_pl_avg(lx1,ly1,lz1,lelt),udy_pl_avg(lx1,ly1,lz1,lelt),
          $ v_pl_avg(lx1,ly1,lz1,lelt),vdy_pl_avg(lx1,ly1,lz1,lelt),
          $ w_pl_avg(lx1,ly1,lz1,lelt),wdy_pl_avg(lx1,ly1,lz1,lelt)


             common /scratch_interp/
          $  xyz1(3,ninp),vel1(3,ninp),dvel(3,ninp),
          $  eps1(3,ninp),eps2(3,ninp),prd1(3,ninp),
          $  prd2(3,ninp),tdf1(3,ninp),tdf2(3,ninp),
          $  pdf1(3,ninp),pdf2(3,ninp),pst1(3,ninp),
          $  pst2(3,ninp),vdf1(3,ninp),vdf2(3,ninp)


           if (ax1.ne.lx1 .or. ay1.ne.ly1 .or. az1.ne.lz1) then
              if(nid.eq.0) write(6,*)
          $     'ABORT: wrong size of ax1,ay1,az1 in avg_all(), check
     SIZEu!'
              call exitt
           endif
           if (ax2.ne.lx2 .or. ay2.ne.ay2 .or. az2.ne.lz2) then
              if(nid.eq.0) write(6,*)
          $     'ABORT: wrong size of ax2,ay2,az2 in avg_all(), check
     SIZEu!'
              call exitt
           endif

           ntot  = nx1*ny1*nz1*nelv
           ntott = nx1*ny1*nz1*nelt
           nto2  = nx2*ny2*nz2*nelv

           if (icalld.eq.0) then
              icalld = icalld + 1
              atime  = 0.
              timel  = time

              call rzero(uavg,ntot)
              call rzero(vavg,ntot)
              call rzero(wavg,ntot)
              call rzero(pavg,nto2)
              call rzero(udxavg,ntot)
              call rzero(udyavg,ntot)
              call rzero(udzavg,ntot)
              call rzero(vdxavg,ntot)
              call rzero(vdyavg,ntot)
              call rzero(vdzavg,ntot)
              call rzero(wdxavg,ntot)
              call rzero(wdyavg,ntot)
              call rzero(wdzavg,ntot)
```

```
   call rzero(udxrms,ntot)
   call rzero(udyrms,ntot)
   call rzero(udzrms,ntot)
   call rzero(vdxrms,ntot)
   call rzero(vdyrms,ntot)
   call rzero(vdzrms,ntot)
   call rzero(wdxrms,ntot)
   call rzero(wdyrms,ntot)
   call rzero(wdzrms,ntot)
   call rzero(uvdxms,ntot)
   call rzero(uvdyms,ntot)
   call rzero(uvdzms,ntot)
   call rzero(uwdxms,ntot)
   call rzero(uwdyms,ntot)
   call rzero(uwdzms,ntot)
   call rzero(vwdxms,ntot)
   call rzero(vwdyms,ntot)
   call rzero(vwdzms,ntot)
   call rzero(upavg,ntot)
   call rzero(vpavg,ntot)
   call rzero(wpavg,ntot)
   call rzero(uuuavg,ntot)
   call rzero(uuvavg,ntot)
   call rzero(uuwavg,ntot)
   call rzero(uvvavg,ntot)
   call rzero(uvwavg,ntot)
   call rzero(uwwavg,ntot)
   call rzero(vvvavg,ntot)
   call rzero(vvwavg,ntot)
   call rzero(vwwavg,ntot)
   call rzero(wwwavg,ntot)
   call rzero(pudxavg,ntot)
   call rzero(pudyavg,ntot)
   call rzero(pudzavg,ntot)
   call rzero(pvdxavg,ntot)
   call rzero(pvdyavg,ntot)
   call rzero(pvdzavg,ntot)
   call rzero(pwdxavg,ntot)
   call rzero(pwdyavg,ntot)
   call rzero(pwdzavg,ntot)
   do i = 1,ldimt
      call rzero(tavg(1,1,1,1,i),ntott)
   enddo
   call rzero(urms,ntot)
   call rzero(vrms,ntot)
   call rzero(wrms,ntot)
   call rzero(prms,nto2)
   do i = 1,ldimt
      call rzero(trms(1,1,1,1,i),ntott)
   enddo
   call rzero(vwms,ntot)
   call rzero(wums,ntot)
   call rzero(uvms,ntot)
endif
```

```
c ------------------------------------------------------------------
c     Load Averaged Quantities
c ------------------------------------------------------------------

      numf = 10

c -----    Velocity Averages    ------------------
      namef = 'avgpbr8'
      call load_avgs(uavg,vavg,wavg,namef,numf)

c -----    Velocity RMS    ------------------
      namef = 'rmspbr8'
      call load_avgs(urms,vrms,wrms,namef,numf)

c -----    Velocity RM2    ------------------
      namef = 'rm2pbr8'
      call load_avgs(uvms,wums,vwms,namef,numf)

c -----    Grad u Average    ------------------
      namef = 'udapbr8'
      call load_avgs(udxavg,udyavg,udzavg,namef,numf)

c -----    Grad v Average    ------------------
      namef = 'vdapbr8'
      call load_avgs(vdxavg,vdyavg,vdzavg,namef,numf)

c -----    Grad w Avergae    ------------------
      namef = 'wdapbr8'
      call load_avgs(wdxavg,wdyavg,wdzavg,namef,numf)

c -----    Grad u RMS    ------------------
      namef = 'udrpbr8'
      call load_avgs(udxrms,udyrms,udzrms,namef,numf)

c -----    Grad v RMS    ------------------
      namef = 'vdrpbr8'
      call load_avgs(vdxrms,vdyrms,vdzrms,namef,numf)

c -----    Grad w RMS    ------------------
      namef = 'wdrpbr8'
      call load_avgs(wdxrms,wdyrms,wdzrms,namef,numf)

c -----    Grad uv RMS    ------------------
      namef = 'uvdpbr8'
      call load_avgs(uvdxms,uvdyms,uvdzms,namef,numf)

c -----    Grad uw RMS    ------------------
      namef = 'uwdpbr8'
      call load_avgs(uwdxms,uwdyms,uwdzms,namef,numf)

c -----    Grad vw RMS    ------------------
      namef = 'vwdpbr8'
      call load_avgs(vwdxms,vwdyms,vwdzms,namef,numf)
```

```
c -----   Pressure*Velocity Averages    ------------------
      namef = 'vpapbr8'
      call load_avgs(upavg,vpavg,wpavg,namef,numf)

c -----    Velocity*3 Averages    ------------------
      namef = 'uuupbr8'
      call load_avgs(uuuavg,uuvavg,uuwavg,namef,numf)

c -----    Velocity*3 Averages    ------------------
      namef = 'uvvpbr8'
      call load_avgs(uvvavg,uvwavg,uwwavg,namef,numf)

c -----    Velocity*3 Averages    ------------------
      namef = 'vvvpbr8'
      call load_avgs(vvvavg,vvwavg,vwwavg,namef,numf)

c -----    Velocity*3 Averages    ------------------
      namef = 'wwwpbr8'
      call load_avgs(wwwavg,wwwavg,wwwavg,namef,numf)

c -----    Grad P*u    ------------------
      namef = 'pudpbr8'
      call load_avgs(pudxavg,pudyavg,pudzavg,namef,numf)

c -----    Grad P*v    ------------------
      namef = 'pvdpbr8'
      call load_avgs(pvdxavg,pvdyavg,pvdzavg,namef,numf)

c -----    Grad P*w    ------------------
      namef = 'pwdpbr8'
      call load_avgs(pwdxavg,pwdyavg,pwdzavg,namef,numf)
```

```
c -----------------------------------------------------------------------
c      Calculate KE budget terms
c -----------------------------------------------------------------------

      if ((mod(istep,iastep).eq.0.and.istep.gt.1) .or.lastep.eq.1) then
c ---  Dissipation  ----------------------------------------------------
c         uu
         call ke_diss(eps_uu,udxrms,udyrms,udzrms,udxavg,udyavg,
     $                     udzavg,udxavg,udyavg,udzavg)
C         vv
         call ke_diss(eps_vv,vdxrms,vdyrms,vdzrms,vdxavg,vdyavg,
     $                     vdzavg,vdxavg,vdyavg,vdzavg)
c        ww
         call ke_diss(eps_ww,wdxrms,wdyrms,wdzrms,wdxavg,wdyavg,
     $                     wdzavg,wdxavg,wdyavg,wdzavg)
c        uv
         call ke_diss(eps_uv,uvdxms,uvdyms,uvdzms,udxavg,udyavg,
     $                     udzavg,vdxavg,vdyavg,vdzavg)
c        uw
         call ke_diss(eps_uw,uwdxms,uwdyms,uwdzms,udxavg,udyavg,
     $                     udzavg,wdxavg,wdyavg,wdzavg)
c        vw
         call ke_diss(eps_vw,vwdxms,vwdyms,vwdzms,vdxavg,vdyavg,
     $                     vdzavg,wdxavg,wdyavg,wdzavg)

c ---  Production  ----------------------------------------------------

c        uu
         call ke_prod(prd_uu,uavg,vavg,wavg,uavg,uavg,urms,uvms,wums,
     $                urms,uvms,wums)
c        vv
         call ke_prod(prd_vv,uavg,vavg,wavg,vavg,vavg,uvms,vrms,vwms,
     $                uvms,vrms,vwms)
c        ww
         call ke_prod(prd_ww,uavg,vavg,wavg,wavg,wavg,wums,vwms,wrms,
     $                wums,vwms,wrms)
c        uv
         call ke_prod(prd_uv,uavg,vavg,wavg,uavg,vavg,urms,uvms,wums,
     $                uvms,vrms,vwms)
c        uw
         call ke_prod(prd_uw,uavg,vavg,wavg,uavg,wavg,urms,uvms,wums,
     $                wums,vwms,wrms)
c        vw
         call ke_prod(prd_vw,uavg,vavg,wavg,vavg,wavg,uvms,vrms,vwms,
     $                wums,vwms,wrms)

c ---  Pressure Diffusion  ------------------------------------------

c        uu
         call ke_pdif(pdf_uu,pavg,uavg,uavg,upavg,upavg)
c        vv
         call ke_pdif(pdf_vv,pavg,vavg,vavg,vpavg,vpavg)
c        ww
         call ke_pdif(pdf_ww,pavg,wavg,wavg,wpavg,wpavg)
c        uv
         call ke_pdif(pdf_uv,pavg,uavg,vavg,upavg,vpavg)
c        uw
         call ke_pdif(pdf_uw,pavg,uavg,wavg,upavg,wpavg)
c        vw
         call ke_pdif(pdf_vw,pavg,vavg,wavg,vpavg,wpavg)

c ---  Pressure Strain  ---------------------------------------------

c        uu
         call ke_pstr(pst_uu,pavg,udxavg,udxavg,pudxavg,pudxavg)
```

```
c        vv
         call ke_pstr(pst_vv,pavg,vdyavg,vdyavg,pvdyavg,pvdyavg)
c        ww
         call ke_pstr(pst_ww,pavg,wdzavg,wdzavg,pwdzavg,pwdzavg)
c        uv
         call ke_pstr(pst_uv,pavg,udyavg,vdxavg,pudyavg,pvdxavg)
c        uw
         call ke_pstr(pst_uw,pavg,udzavg,wdxavg,pudzavg,pwdxavg)
c        vw
         call ke_pstr(pst_vw,pavg,vdzavg,wdyavg,pvdzavg,pwdyavg)

c ---  Viscous Diffusion  ------------------------------------------

c        uu
         call ke_vdif(vdf_uu,uavg,uavg,urms)
c        vv
         call ke_vdif(vdf_vv,vavg,vavg,vrms)
c        ww
         call ke_vdif(vdf_ww,wavg,wavg,wrms)
c        uv
         call ke_vdif(vdf_uv,uavg,vavg,uvms)
c        uw
         call ke_vdif(vdf_uw,uavg,wavg,wums)
c        vw
         call ke_vdif(vdf_vw,vavg,wavg,vwms)

c ---  Turbulent Diffusion  -----------------------------------------

c        uu
         call ke_tdif(tdf_uu,uavg,vavg,wavg,uavg,uavg,urms,uuuavg,
     $                uuvavg,uuwavg,urms,uvms,wums,urms,uvms,wums)
c        vv
         call ke_tdif(tdf_vv,uavg,vavg,wavg,vavg,vavg,vrms,uvvavg,
     $                vvvavg,vvwavg,uvms,vrms,vwms,uvms,vrms,vwms)
c        ww
         call ke_tdif(tdf_ww,uavg,vavg,wavg,wavg,wavg,wrms,uwwavg,
     $                vwwavg,wwwavg,wums,vwms,wrms,wums,vwms,wrms)
c        uv
         call ke_tdif(tdf_uv,uavg,vavg,wavg,uavg,vavg,uvms,uuvavg,
     $                uvvavg,uvwavg,urms,uvms,wums,uvms,vrms,vwms)
c        uw
         call ke_tdif(tdf_uw,uavg,vavg,wavg,uavg,wavg,wums,uuwavg,
     $                uvwavg,uwwavg,urms,uvms,wums,wums,vwms,wrms)
c        vw
         call ke_tdif(tdf_vw,uavg,vavg,wavg,vavg,wavg,vwms,uvwavg,
     $                vvwavg,vwwavg,uvms,vrms,vwms,wums,vwms,wrms)

c -----------------------------------------------------------------
c      Prepare Outputs
c -----------------------------------------------------------------


c -----  Output .fld files for visualization  -------------------------
----
```

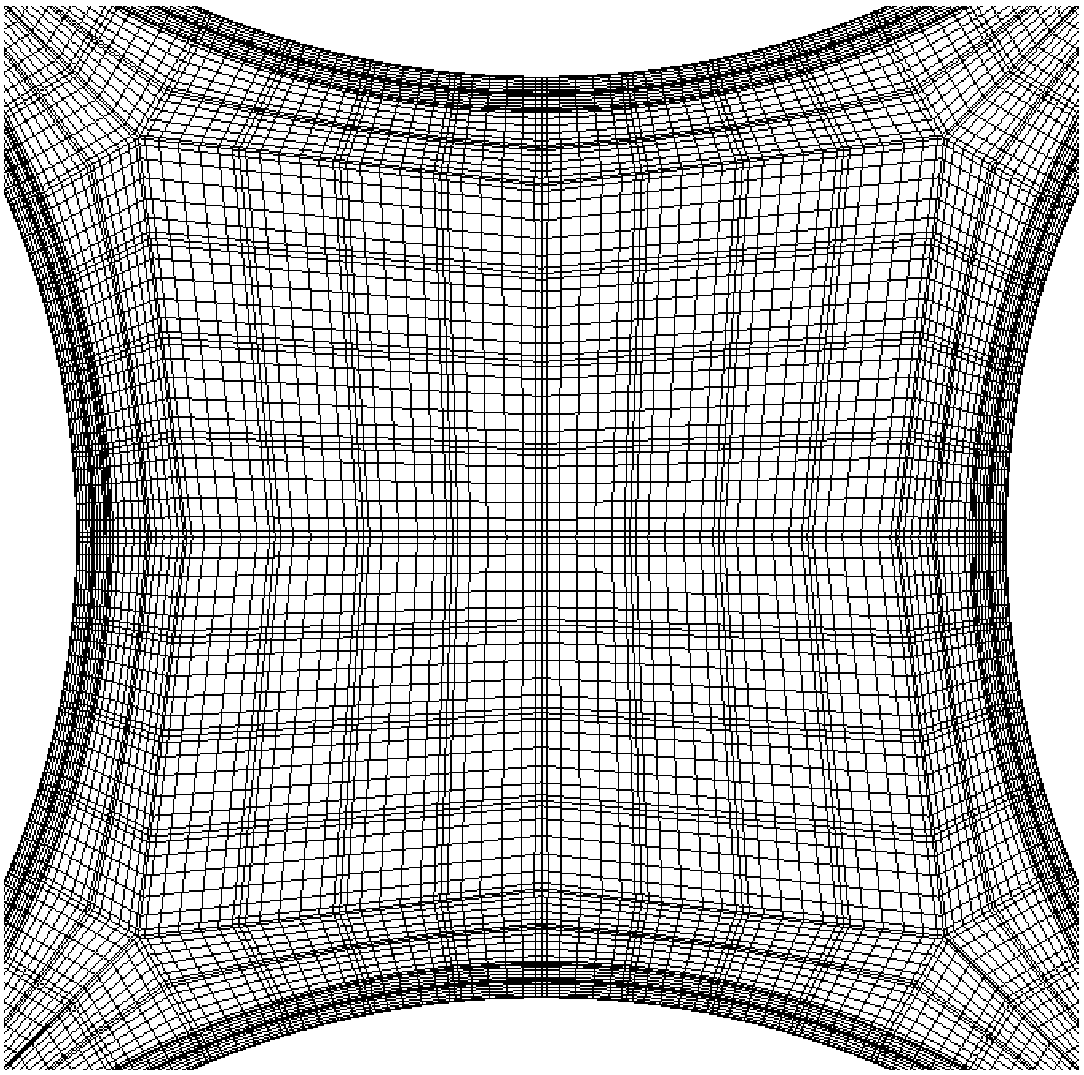
```
c         call outpost2(udyavg,vdyavg,wdyavg,pavg,tavg,ldimt,'vdy')
         call outpost2(eps_uu,eps_vv,eps_ww,pavg,tavg,ldimt,'ep1')
         call outpost2(eps_uv,eps_uw,eps_vw,pavg,tavg,ldimt,'ep2')
         call outpost2(prd_uu,prd_vv,prd_ww,pavg,tavg,ldimt,'pr1')
         call outpost2(prd_uv,prd_uw,prd_vw,pavg,tavg,ldimt,'pr2')
         call outpost2(tdf_uu,tdf_vv,tdf_ww,pavg,tavg,ldimt,'td1')
         call outpost2(tdf_uv,tdf_uw,tdf_vw,pavg,tavg,ldimt,'td2')
         call outpost2(pdf_uu,pdf_vv,pdf_ww,pavg,tavg,ldimt,'pd1')
         call outpost2(pdf_uv,pdf_uw,pdf_vw,pavg,tavg,ldimt,'pd2')
         call outpost2(pst_uu,pst_vv,pst_ww,pavg,tavg,ldimt,'ps1')
         call outpost2(pst_uv,pst_uw,pst_vw,pavg,tavg,ldimt,'ps2')
         call outpost2(vdf_uu,vdf_vv,vdf_ww,pavg,tavg,ldimt,'vd1')
         call outpost2(vdf_uv,vdf_uw,vdf_vw,pavg,tavg,ldimt,'vd2')

c  ----  Interpolate desired variables and write to csv files   -------
--

         if (interp.eq.1) then

         n_interp=ninp
         do j=1,ninp
            xyz1(1,j) = 8.0
            xyz1(3,j) = 5.0
            if (j.LE.100) then
               xyz1(2,j) = 0+0.001*j
            elseif ((j.GT.100) .and. (j.LT.301)) then
               xyz1(2,j) = 0.1+0.009*(j-100)
            else
               xyz1(2,j) = 1.9+0.001*(j-300)
            endif
         enddo

         call interp_var(vel1,xyz1,n_interp,uavg,vavg,wavg)
         call interp_var(dvel,xyz1,n_interp,udyavg,vdyavg,wdyavg)
         call interp_var(eps1,xyz1,n_interp,eps_uu,eps_vv,eps_ww)
         call interp_var(eps2,xyz1,n_interp,eps_uv,eps_uw,eps_vw)
         call interp_var(prd1,xyz1,n_interp,prd_uu,prd_vv,prd_ww)
         call interp_var(prd2,xyz1,n_interp,prd_uv,prd_uw,prd_vw)
         call interp_var(tdf1,xyz1,n_interp,tdf_uu,tdf_vv,tdf_ww)
         call interp_var(tdf2,xyz1,n_interp,tdf_uv,tdf_uw,tdf_vw)
         call interp_var(pdf1,xyz1,n_interp,pdf_uu,pdf_vv,pdf_ww)
         call interp_var(pdf2,xyz1,n_interp,pdf_uv,pdf_uw,pdf_vw)
         call interp_var(pst1,xyz1,n_interp,pst_uu,pst_vv,pst_ww)
         call interp_var(pst2,xyz1,n_interp,pst_uv,pst_uw,pst_vw)
         call interp_var(vdf1,xyz1,n_interp,vdf_uu,vdf_vv,vdf_ww)
         call interp_var(vdf2,xyz1,n_interp,vdf_uv,vdf_uw,vdf_vw)


open(197,file='data_eps.csv',form='formatted',status='replace')

         do j=1,ninp
            write(197,10)j,xyz1(2,j),eps1(1,j),eps1(2,j),eps1(3,j),
     $                    eps2(1,j),eps2(2,j),eps2(3,j)
         enddo
```

```
  10     format(I4,',',F8.5,',',F8.5,',',F8.5,',',F8.5,',',F8.5,',',
     $           F8.5,',',F8.5)
         close(197)


open(198,file='data_prd.csv',form='formatted',status='replace')

         do j=1,ninp
           write(198,10)j,xyz1(2,j),prd1(1,j),prd1(2,j),prd1(3,j),
     $                    prd2(1,j),prd2(2,j),prd2(3,j)
         enddo
         close(198)


open(199,file='data_tdf.csv',form='formatted',status='replace')

         do j=1,ninp
           write(199,10)j,xyz1(2,j),tdf1(1,j),tdf1(2,j),tdf1(3,j),
     $                    tdf2(1,j),tdf2(2,j),tdf2(3,j)
         enddo
         close(199)


open(200,file='data_pdf.csv',form='formatted',status='replace')

         do j=1,ninp
           write(200,10)j,xyz1(2,j),pdf1(1,j),pdf1(2,j),pdf1(3,j),
     $                    pdf2(1,j),pdf2(2,j),pdf2(3,j)
         enddo
         close(200)


open(201,file='data_pst.csv',form='formatted',status='replace')

         do j=1,ninp
           write(201,10)j,xyz1(2,j),pst1(1,j),pst1(2,j),pst1(3,j),
     $                    pst2(1,j),pst2(2,j),pst2(3,j)
         enddo
         close(201)


open(202,file='data_vdf.csv',form='formatted',status='replace')

         do j=1,ninp
           write(202,10)j,xyz1(2,j),vdf1(1,j),vdf1(2,j),vdf1(3,j),
     $                    vdf2(1,j),vdf2(2,j),vdf2(3,j)
         enddo
         close(202)


open(203,file='data_vel.csv',form='formatted',status='replace')

         do j=1,ninp
           write(203,10)j,xyz1(2,j),vel1(1,j),vel1(2,j),vel1(3,j),
     $                   dvel(1,j),dvel(2,j),dvel(3,j)
         enddo
         close(203)

         endif

      endif

      return
      end


c-----------------------------------------------------------------------
-
      subroutine interp_var(intvar,xyz,n,wrk1,wrk2,wrk3)
c
c     General version of interp_v, interpolates desired variables
c     wrk(1-3) at points xyz
c
c     intpts to get rid off " WARNING: point on boundary or ..."

      include 'SIZE'
      include 'TOTAL'

      real intvar(3,n),xyz(ldim,n),wrk1(lx1,ly1,lz1,lelt),
     $      wrk2(lx1,ly1,lz1,lelt),wrk3(lx1,ly1,lz1,lelt)
      logical ifjac,ifpts

      parameter(nmax=lpart,nfldmax=3)
      common /rv_intp/ pts(ldim*nmax)
      common /iv_intp/ ihandle
      common /outtmp/ wrk(lx1*ly1*lz1*lelt,nfldmax)


      integer icalld,e
      save    icalld
      data    icalld /0/

      nxyz  = nx1*ny1*nz1
      ntot  = nxyz*nelt
      nflds = 3 !velocity

      if (n.gt.nmax) call exitti ('ABORT: interp_v() n > nmax!$',n)

      if (nelgt.ne.nelgv) call exitti
     $   ('ABORT: interp_v() nelgt.ne.nelgv not yet supported!$',nelgv)

      do i=1,n                          ! ? not moving -> save?
         pts(i)     = xyz(1,i)
         pts(i + n) = xyz(2,i)
         if (if3d)  pts(i + n*2) = xyz(3,i)
      enddo

      if (icalld.eq.0) then             ! interpolation setup
```

```
        icalld = 1
        tolin  = 1.e-8
        call intpts_setup(tolin,ihandle)
      endif

      ! pack working array

      do i=1,ntot
       wrk(i,1)=wrk1(i,1,1,1)
       wrk(i,2)=wrk2(i,1,1,1)
       wrk(i,3)=wrk3(i,1,1,1)
      enddo

      ! interpolate
      ifjac  = .true.           ! output transpose (of Jacobian)
      ifpts  = .true.            ! find points
      call intpts(wrk,nflds,pts,n,intvar,ifjac,ifpts,ihandle)      !
copy array instead?

      return
      end

c  -----------------------------------------------------------------------

      subroutine laplacian(gradd,u)

      include 'SIZE'

c  Computes the laplacian of variable u using gradm1

      real gradd(lx1,ly1,lz1,lelt),u(lx1,ly1,lz1,lelt),
     $     udx(lx1,ly1,lz1,lelt),udy(lx1,ly1,lz1,lelt),
     $     udz(lx1,ly1,lz1,lelt),udxx(lx1,ly1,lz1,lelt),
     $     udxy(lx1,ly1,lz1,lelt),udxz(lx1,ly1,lz1,lelt),
     $     udyx(lx1,ly1,lz1,lelt),udyy(lx1,ly1,lz1,lelt),
     $     udyz(lx1,ly1,lz1,lelt),udzx(lx1,ly1,lz1,lelt),
     $     udzy(lx1,ly1,lz1,lelt),udzz(lx1,ly1,lz1,lelt)

      integer i,j,k,e


      call gradm1(udx,udy,udz,u)
      call gradm1(udxx,udxy,udxz,udx)
      call gradm1(udyx,udyy,udyz,udy)
      call gradm1(udzx,udzy,udzz,udz)

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      gradd(i,j,k,e) = udxx(i,j,k,e)+udyy(i,j,k,e)+udzz(i,j,k,e)
      enddo
      enddo
      enddo
      enddo

      return
      end

c ----------------------------------------------------------------------
      subroutine ke_diss(eps,abx,aby,abz,adx,ady,adz,bdx,bdy,bdz)

      include 'SIZE'

      integer i,j,k,e

      real eps(lx1,ly1,lz1,lelt),abx(lx1,ly1,lz1,lelt),
     $     aby(lx1,ly1,lz1,lelt),abz(lx1,ly1,lz1,lelt),
     $     adx(lx1,ly1,lz1,lelt),ady(lx1,ly1,lz1,lelt),
     $     adz(lx1,ly1,lz1,lelt),bdx(lx1,ly1,lz1,lelt),
     $     bdy(lx1,ly1,lz1,lelt),bdz(lx1,ly1,lz1,lelt),
     $     vis

      vis = 4.2293E-4

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      eps(i,j,k,e) = 2*vis*(abx(i,j,k,e)+aby(i,j,k,e)+abz(i,j,k,e)-
     $               adx(i,j,k,e)*bdx(i,j,k,e)-
     $               ady(i,j,k,e)*bdy(i,j,k,e)-
     $               adz(i,j,k,e)*bdz(i,j,k,e))
      enddo
      enddo
      enddo
      enddo

      return
      end

c -------------------------------------------------------------------------
      subroutine ke_prod(prd,u,v,w,a,b,aums,avms,awms,bums,bvms,bwms)

      include 'SIZE'

      integer i,j,k,e

      real u(lx1,ly1,lz1,lelt),v(lx1,ly1,lz1,lelt),w(lx1,ly1,lz1,lelt),
     $     a(lx1,ly1,lz1,lelt),b(lx1,ly1,lz1,lelt),
     $     aums(lx1,ly1,lz1,lelt),avms(lx1,ly1,lz1,lelt),
     $     awms(lx1,ly1,lz1,lelt),bums(lx1,ly1,lz1,lelt),
     $     bvms(lx1,ly1,lz1,lelt),bwms(lx1,ly1,lz1,lelt),
     $     adx(lx1,ly1,lz1,lelt),ady(lx1,ly1,lz1,lelt),
     $     adz(lx1,ly1,lz1,lelt),bdx(lx1,ly1,lz1,lelt),
     $     bdy(lx1,ly1,lz1,lelt),bdz(lx1,ly1,lz1,lelt),
     $     prd(lx1,ly1,lz1,lelt)
```

```
      call gradm1(adx,ady,adz,a)
      call gradm1(bdx,bdy,bdz,b)

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      prd(i,j,k,e) = (a(i,j,k,e)*u(i,j,k,e)-
aums(i,j,k,e))*bdx(i,j,k,e)+
     $               (a(i,j,k,e)*v(i,j,k,e)-
avms(i,j,k,e))*bdy(i,j,k,e)+
     $               (a(i,j,k,e)*w(i,j,k,e)-
awms(i,j,k,e))*bdz(i,j,k,e)+
     $               (b(i,j,k,e)*u(i,j,k,e)-
bums(i,j,k,e))*adx(i,j,k,e)+
     $               (b(i,j,k,e)*v(i,j,k,e)-
bvms(i,j,k,e))*ady(i,j,k,e)+
     $               (b(i,j,k,e)*w(i,j,k,e)-bwms(i,j,k,e))*adz(i,j,k,e)
      enddo
      enddo
      enddo
      enddo

      return
      end

c -----------------------------------------------------------------------
      subroutine ke_pdif(pdf,p,a,b,pa,pb)

      include 'SIZE'

      integer i,j,k,e

      real rho

      real pdf(lx1,ly1,lz1,lelt),p(lx1,ly1,lz1,lelt),
     $     a(lx1,ly1,lz1,lelt),b(lx1,ly1,lz1,lelt),
     $     pa(lx1,ly1,lz1,lelt),pb(lx1,ly1,lz1,lelt)

      rho = 1

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      pdf(i,j,k,e) = 1/rho*(pa(i,j,k,e)-p(i,j,k,e)*a(i,j,k,e)+
     $                      pb(i,j,k,e)-p(i,j,k,e)*b(i,j,k,e))
      enddo
      enddo
      enddo
      enddo

      return
      end


c -----------------------------------------------------------------------
      subroutine ke_pstr(pst,p,adb,bda,padb,pbda)

      include 'SIZE'

      integer i,j,k,e
      real rho
      real pst(lx1,ly1,lz1,lelt),p(lx1,ly1,lz1,lelt),
     $     adb(lx1,ly1,lz1,lelt),bda(lx1,ly1,lz1,lelt),
     $     padb(lx1,ly1,lz1,lelt),pbda(lx1,ly1,lz1,lelt)

      rho = 1

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      pst(i,j,k,e) = 1/rho*(padb(i,j,k,e)-p(i,j,k,e)*adb(i,j,k,e)+
     $                     pbda(i,j,k,e)-p(i,j,k,e)*bda(i,j,k,e))
      enddo
      enddo
      enddo
      enddo

      return
      end

c -----------------------------------------------------------------------
      subroutine ke_vdif(vdf,a,b,abms)

      include 'SIZE'

      integer i,j,k,e
      real vis
      real vdf(lx1,ly1,lz1,lelt),a(lx1,ly1,lz1,lelt),
     $     b(lx1,ly1,lz1,lelt),abms(lx1,ly1,lz1,lelt),
     $     vdf_temp(lx1,ly1,lz1,lelt)

      vis = 4.2293E-4

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      vdf_temp(i,j,k,e) = vis*(abms(i,j,k,e)-a(i,j,k,e)*b(i,j,k,e))
      enddo
      enddo
      enddo
      enddo

      call laplacian(vdf,vdf_temp)
      return
      end

c ----------------------------------------------------------------------
-
      subroutine
ke_tdif(tdf,u,v,w,a,b,ba,abu,abv,abw,au,av,aw,bu,bv,bw)

      include 'SIZE'
      integer i,j,k,e

      real tdf(lx1,ly1,lz1,lelt),a(lx1,ly1,lz1,lelt),
     $     b(lx1,ly1,lz1,lelt),u(lx1,ly1,lz1,lelt),
     $     v(lx1,ly1,lz1,lelt),w(lx1,ly1,lz1,lelt),
     $     au(lx1,ly1,lz1,lelt),av(lx1,ly1,lz1,lelt),
     $     aw(lx1,ly1,lz1,lelt),bu(lx1,ly1,lz1,lelt),
     $     bv(lx1,ly1,lz1,lelt),bw(lx1,ly1,lz1,lelt),
     $     abu(lx1,ly1,lz1,lelt),abv(lx1,ly1,lz1,lelt),
     $     abw(lx1,ly1,lz1,lelt),ba(lx1,ly1,lz1,lelt)

      common /tdif_calc/
     $     tdf1(lx1,ly1,lz1,lelt),tdf2(lx1,ly1,lz1,lelt),
     $     tdf3(lx1,ly1,lz1,lelt),tdf1dx(lx1,ly1,lz1,lelt),
     $     tdf1dy(lx1,ly1,lz1,lelt),tdf1dz(lx1,ly1,lz1,lelt),
     $     tdf2dx(lx1,ly1,lz1,lelt),tdf2dy(lx1,ly1,lz1,lelt),
     $     tdf2dz(lx1,ly1,lz1,lelt),tdf3dx(lx1,ly1,lz1,lelt),
     $     tdf3dy(lx1,ly1,lz1,lelt),tdf3dz(lx1,ly1,lz1,lelt)

      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      tdf1(i,j,k,e) = abu(i,j,k,e)-
     $                au(i,j,k,e)*b(i,j,k,e)-
     $                a(i,j,k,e)*bu(i,j,k,e)-
     $                ba(i,j,k,e)*u(i,j,k,e)+
     $                2*a(i,j,k,e)*b(i,j,k,e)*u(i,j,k,e)

      tdf2(i,j,k,e) = abv(i,j,k,e)-
     $                av(i,j,k,e)*b(i,j,k,e)-
     $                a(i,j,k,e)*bv(i,j,k,e)-
     $                ba(i,j,k,e)*v(i,j,k,e)+
     $                2*a(i,j,k,e)*b(i,j,k,e)*v(i,j,k,e)

      tdf3(i,j,k,e) = abw(i,j,k,e)-
     $                aw(i,j,k,e)*b(i,j,k,e)-
     $                a(i,j,k,e)*bw(i,j,k,e)-
     $                ba(i,j,k,e)*w(i,j,k,e)+
     $                2*a(i,j,k,e)*b(i,j,k,e)*w(i,j,k,e)

      enddo
      enddo
      enddo
      enddo
      call gradm1(tdf1dx,tdf1dy,tdf1dz,tdf1)
      call gradm1(tdf2dx,tdf2dy,tdf2dz,tdf2)
      call gradm1(tdf3dx,tdf3dy,tdf3dz,tdf3)
      do e=1,lelt
      do k=1,lz1
      do j=1,ly1
      do i=1,lx1
      tdf(i,j,k,e) = tdf1dx(i,j,k,e)+tdf2dy(i,j,k,e)+tdf3dz(i,j,k,e)
      enddo
      enddo
      enddo
      enddo

      return
      end

c ---------------------------------------------------------------------
--

      subroutine load_avgs(avgq1,avgq2,avgq3,namef,numf)

      include 'SIZE'
      include 'ZPER'
      include 'TOTAL'


      real avgq1(lx1,ly1,lz1,lelt),avgq2(lx1,ly1,lz1,lelt),
     $     avgq3(lx1,ly1,lz1,lelt)

      character namef*7

      integer i, j, k, e, ii, numf

      ntot = lx1*ly1*lz1*lelt

      call rzero(avgq1,ntot)
      call rzero(avgq2,ntot)
      call rzero(avgq3,ntot)

      do ii=1,numf

      call blank(initc(1),80)

      write(initc(1),7)ii,namef

   7       format('/intrepid-
fs0/users/wardpaul/persistent/test_pbr/run',
     &     i2.2,'/',A7,'.fld01',' U')

      call setics


      do i=1,lx1
      do j=1,ly1
      do k=1,lz1
      do e=1,lelt
         avgq1(i,j,k,e)=avgq1(i,j,k,e)+vx(i,j,k,e)*(1.0/numf)
         avgq2(i,j,k,e)=avgq2(i,j,k,e)+vy(i,j,k,e)*(1.0/numf)
         avgq3(i,j,k,e)=avgq3(i,j,k,e)+vz(i,j,k,e)*(1.0/numf)
      enddo
      enddo
      enddo
      enddo

      enddo

      return
      end
```

APPENDIX B: MESH SAMPLE



Mesh Sample from Y-Z Plane