Implementation of the Oriented-Eddy Collision Turbulence Model in OpenFoam

Blair Perot
Mike Martell
Chris Zusi

Department of Mechanical and Industrial Engineering

UMassAmherst

Goal: Develop a New Turbulence Model

Navy Requirements

- Easy for others to test.
- Easy for others to adopt.
- Easy to test real world applications.

Our Requirements

- Solve coupled tensor PDE’s.
- Mixed BCs.
- Handle arbitrary numbers of equations.

Why OpenFoam?

```
tmp<fvScalarMatrix> kEqn
(
      fvm::ddt(kINT)
    - fvm::laplacian(dEff(), kINT)
    + fvm::SuSp((alpha*nu()*qsq + tauR), kINT)
    ==
    - fvc::div(phi_, kINT)
    + (Ptmp && (RijStarINT*kINT))
    - A
    + M
);
```

- **Advantages over commercial code**
  - Free, open source, parallel
  - Users can inspect, alter, expand on the source code.
- **Advantages over in house code development**
  - Many numerical methods, operators, utilities already implemented and tested
- **Large, user-driven support community**
  - Interact with other OpenFOAM users.
  - Get help from CFD experts.

Why OpenFoam?

- **Advant**
  - Free,
  - Users code.
- **Advant**
  - Many alread

```
tmp<fvScalarMatrix> kEqn
(
      fvm::ddt(kINT)
    - fvm::laplacian(dEff(), kINT)
    + fvm::SuSp((alpha*nu()*qsq + tauR), kINT)
    ==
    - fvc::div(phi_, kINT)
    + (Ptmp && (RijStarINT*kINT))
    - A
    + M
);
```

- **Large, user-driven support community**
  - Interact with other OpenFOAM users.
  - Get help from CFD experts.

Why OpenFoam?

```
tmp<fvScalarMatrix> kEqn
(
      fvm::ddt(kINT)
    - fvm::laplacian(dEff(), kINT)
    + fvm::SuSp((alpha*nu()*qsq + tauR), kINT)
    ==
    - fvc::div(phi_, kINT)
    + (Ptmp && (RijStarINT*kINT))
    - A
    + M
);
```

- **Advantages over commercial code**
  - Free, open source, parallel
  - Users can inspect, alter, expand on the source code
- **Advant**
  - Many alrea
- **Large,**
  - Inter
  - Get h

Prior Experience:

In House Codes

- UNS3D: Moving unstructured staggered mesh code for two-phase incompressible flows
- Stag++: Cartesian staggered mesh for DNS/LES of incompressible turbulence.

Fluent

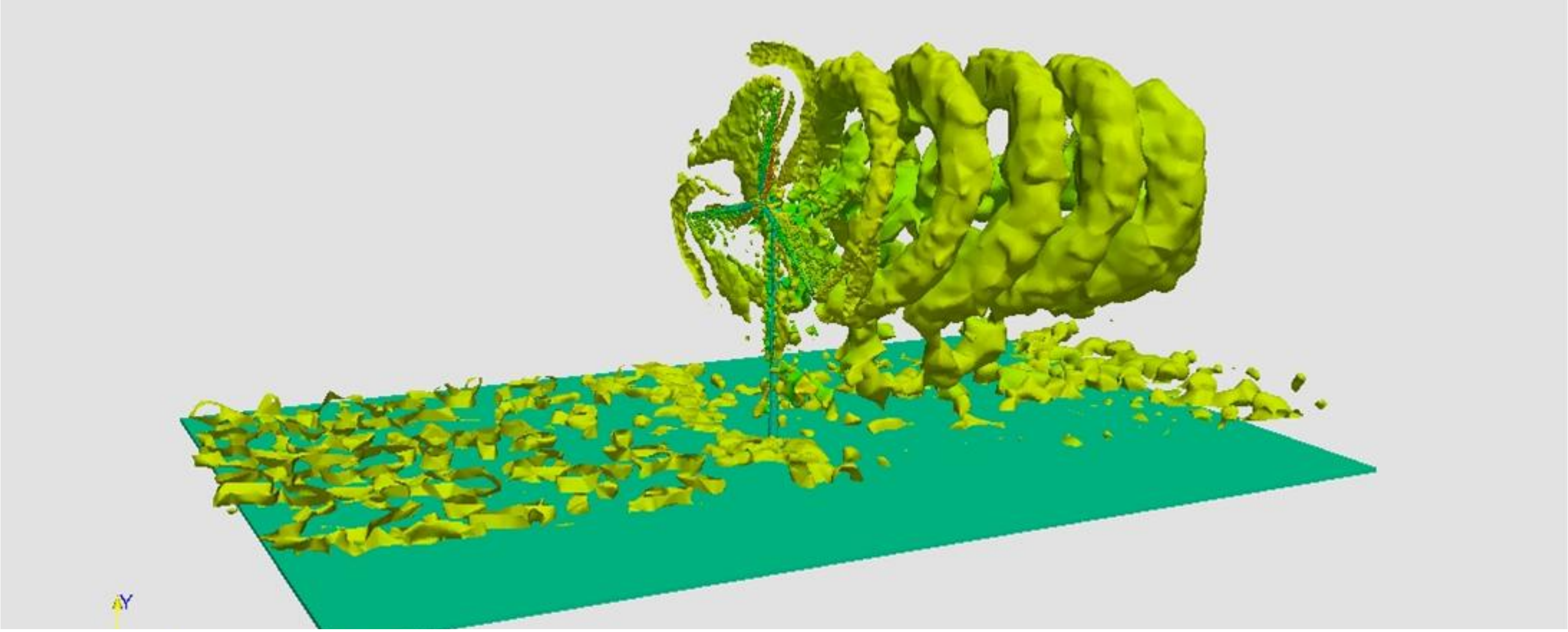
- User defined subroutines for RANS modeling.

OpenFoam

- Wind turbine blade simulation. Rotating imbedded mesh.

Wind Turbine Calculations with OpenFoam

Spin indicator
= second invariant of the strain tensor

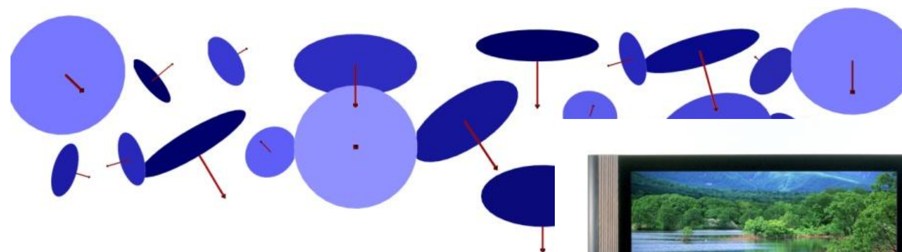


- Runs on 8-16 CPUs
- No major issues

Eddy Collision Model Overview

Assumption:

Turbulent Flow = Flow of a Colloidal suspension of disk-like spinning objects.

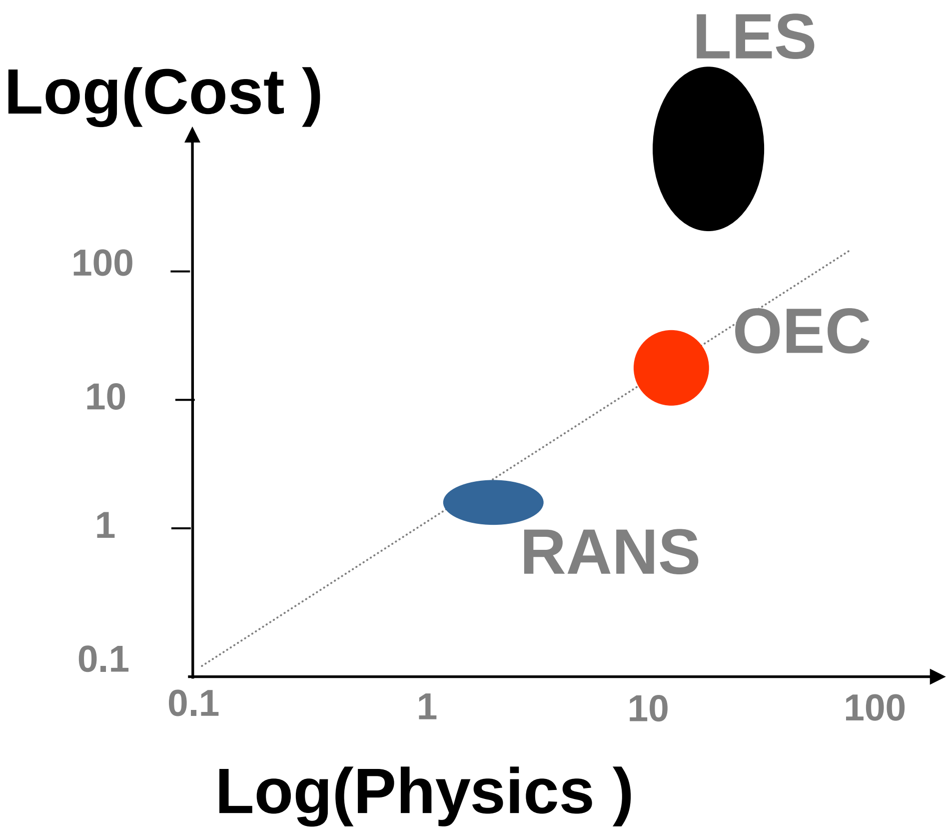


Pouring Spherical objects results in RANS eqns.

Pouring a concentrated Disk suspension results in OEC model

Review of OEC Model Properties

- Cost roughly about 10x k/e.
- Four model constants.
- Realizable, Material Frame Indifferent, Galilean Invariant, Exact in linear limit, etc.



PDE Formulation

$$\frac{Dq_i}{Dt} = -q_k \bar{u}_{k,i} - \frac{1}{3}\left(\alpha\nu\overline{q^2} + \frac{1}{\tau_R}\right)q_i - (A_i + B_i) + \frac{1}{3}\left[(\nu+\nu_t)q_{i,k}\right]_{,k} + W_i$$

Eddy Size and Orientation (vector)

$$\frac{DR_{ij}}{Dt} = \left[\bar{u}_{i,k} + \left(\frac{q_i q_l}{q^2} - \delta_{il}\right)2\bar{u}^*_{l,k}\right]R_{kj} + \left[\bar{u}_{j,k} + \left(\frac{q_j q_l}{q^2} - \delta_{jl}\right)2\bar{u}^*_{l,k}\right]R_{ki} - \left(\alpha\nu\overline{q^2} + \frac{1}{\tau_R}\right)R_{ij}$$
$$- A_{ij} + M_{ij} + \left[(\nu+\nu_t)R_{ij,k}\right]_{,k} - D(\nu+\nu_t)\left[\frac{R_{ij}}{K}\right]_{,k}(K)_{,k} - E(\nu+\nu_t)\frac{(K)_{,k}}{K}\frac{(K)_{,k}}{K}R_{ij} + W_{ij}$$

Eddy Velocity Fluctuation (tensor)

- **Global Variables (sum many eddies – 20-50)**

$$\tilde{R}_{ij} = \tfrac{1}{N}\sum R_{ij} \qquad \nu_T = \sqrt{\sum \tfrac{1}{2} R_{ii}{}^2 / \sum R_{ii} q^{2}}$$

Open Foam Formulation

$$\frac{Dq_i}{Dt} = -q_k \bar{u}_{k,i} - \frac{1}{3}\left(\alpha \nu \overline{q^2} + \frac{1}{\tau_R}\right) q_i - (A_i + B_i) + \frac{1}{3}\left[(\nu + \nu_t) q_{i,k}\right]_{,k} + W_i$$

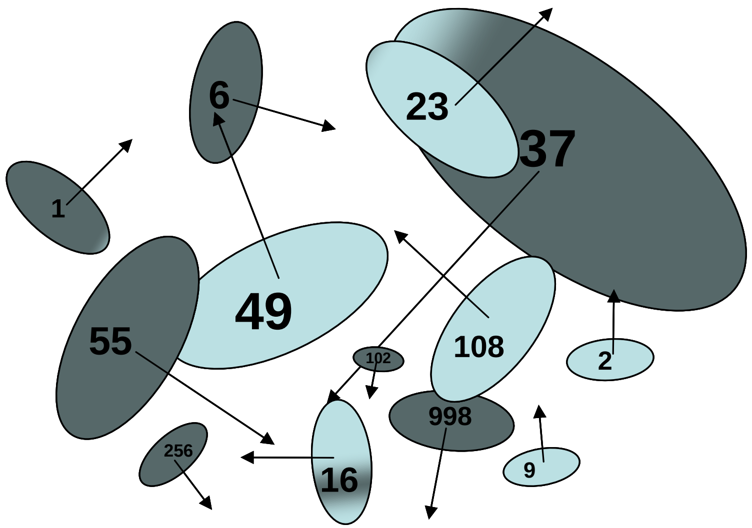
```
fvm::ddt(qiINT)
- (1.0/3.0)*fvm::laplacian(dEff(), qiINT)
+ (1.0/3.0)*fvm::SuSp(((alpha*nu()*qsq + tauR)), qiINT)
==
- fvc::div(phi_, qiINT)
- ( qiINT & fvc::grad(U) )
- ( Ai + Bi )
```

Implicit terms on the left-hand side.

Explicit terms on the right-hand side.

First Challenge: Multiple Eddies

- **The Number of Eddies for each physical location is arbitrary.**
- **10 is minimal necessary.**
- **100 is usually very good.**



Pointer lists are employed to keep track of eddies

Rij_[eddy][cell].xx() ← **Component (11 in this case)**

Pointer list with an entry for every eddy

Location in mesh

Multiple Eddies: OpenFoam Solution

```
forAll(initOrientations_,i) {
      ...
      solveqR(i, qi_[i], ...);
      ...
 }

void OEC::solveqR(int i, volVectorField qiINT, ...) {
      ...
      tmp<fvVectorMatrix> qEqn
       (
             fvm::ddt(qiINT) = ...
       );
       ...
       solve(qEqn, mesh_.solver("q"));
       ...
}
```

This system allows us to write generalized functions which handle any number of eddy vectors.

The model can be implemented on a per-eddy basis.

Averaging all of the entities in a given pointer list is also easy, which is good because all we really care about is the average R, K, etc.

Second Challenge: Tensors

- No gradient of a rank 2 (and higher) tensor

$$-D\left(\nu+\nu_t\right)\left[\frac{R_{ij}}{K}\right]_{,k}(K)_{,k}$$

```
...
fvc::grad(R)
...
```

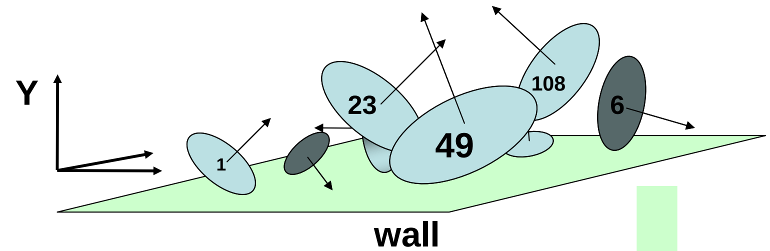
Solution:

Break it into vectors

```
forAll (mesh_.C(), cell)        // internal cells
 {
      Rx[cell].x() = Rtmp[cell].xx();
      Rx[cell].y() = Rtmp[cell].xy();
      Rx[cell].z() = Rtmp[cell].xz();
      ...
      gradKgradR[cell].xx() = ( gradK[cell].x()*gradRx[cell].xx()
                              + gradK[cell].y()*gradRx[cell].xy()
                              + gradK[cell].z()*gradRx[cell].xz() );
      gradKgradR[cell].yy() = ( gradK[cell].x()*gradRy[cell].yx()
                              + gradK[cell].y()*gradRy[cell].yy()
                              + gradK[cell].z()*gradRy[cell].yz() );
      gradKgradR[cell].zz() = ( gradK[cell].x()*gradRz[cell].zx()
                              + gradK[cell].y()*gradRz[cell].zy()
                              + gradK[cell].z()*gradRz[cell].zz() );
```

Third Challenge: Boundary Conditions

- On wall, the boundary conditions are mixed.



This is for an xz-wall.

We currently can not do walls that are not aligned with the tensor coordinate directions.

$$R_{ij}\Big|_{slip-wall} \rightarrow \begin{bmatrix} \dfrac{\partial R_{11}}{\partial y} = 0 & R_{12} = 0 & \dfrac{\partial R_{13}}{\partial y} \\ & R_{22} = 0 & R_{23} = 0 \\ & & \dfrac{\partial R_{33}}{\partial y} \end{bmatrix}$$

$$q_i\Big|_{wall} \rightarrow \begin{bmatrix} q_1 = 0 \\ \dfrac{\partial q_2}{\partial y} = 0 \\ q_3 = 0 \end{bmatrix}$$

Last Challenge: Stable Time Marching

- Many source terms have no fvm:: (implicit) implementation.
- Some explicit source terms can be unstable with Explicit Euler time advancement.

```
tmp<fvVectorMatrix> qEqn
(
      fvm::ddt(qiINT)
    - (1.0/3.0)*fvm::laplacian(dEff(), qiINT)
    + (1.0/3.0)*fvm::SuSp(((alpha*nu()*qsq + tauR)), qiINT)
    ==
    - fvc::div(phi_, qiINT)
    - ( qiINT & GU_ )
    - ( Ai + Bi )
    + ( (qiTMPINT - qiINT) / mesh_.time().deltaT() )     // RK3 correction term
);

//...

solve(qEqn, mesh_.solver("q"));
```

Solution:

- Write a RK3 solver.
- Modify equations with explicit time derivative
- Use FOAM's `.storeOldTime()` to save the old time values.

Last Challenge: Stable Time Marching

```
tmp<fvVectorMatrix> qEqn
(
      fvm::ddt(qiINT)
    - (1.0/3.0)*fvm::laplacian(dEff(), qiINT)
    + (1.0/3.0)*fvm::SuSp(((alpha*nu()*qsq + tauR)), qiINT)
   ==
    - fvc::div(phi_, qiINT)
    - ( qiINT & GU_ )
    - ( Ai + Bi )
    + ( (qiTMPINT - qiINT) / mesh_.time().deltaT() )     // RK3 correction term
);

//...

solve(qEqn, mesh_.solver("q"));
```

- Use FOAM's `.storeOldTime()` to save the old time values.

Last Observation:

- Gradient of a vector

$$a_{i,j} = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \frac{\partial a_2}{\partial x_1} \\ \frac{\partial a_1}{\partial x_2} & \frac{\partial a_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{2,1} \\ a_{1,2} & a_{2,2} \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} = \partial_i a_j$$

Summary:

- OEC is implemented in OpenFoam.