Table of Contents

Initial values

The following code shows five different udfs for inlet velocity, temperature, turbulent kinematic energy, turbulent dissipation and turbulent diffusion.

```
#include "udf.h"

DEFINE_PROFILE(inlet_vel_10, thread, position)
{

  real p[ND_ND]; /* this will hold the position vector */
  real y, w;
  face_t f;

  w = 0.1; /* inlet width in m */

  begin_f_loop(f,thread)
  {
    F_CENTROID(p, f, thread);
    y = p[1]; /*  coordinate, here p[one] means y-position, as in
 0,1,2...
    is x,y,z */

 F_PROFILE(f, thread, position) = (8. * 10. / 7.) * pow(((0.5 * w -...
    y) / (0.5 * w)), 1. / 7.);
  }
  end_f_loop(f, thread)
}


#include "udf.h"

DEFINE_PROFILE(inlet_x_e, thread, position)
{

  real p[ND_ND]; /* this will hold the position vector */
  real y, w, cl, a, re, lu, le, vt, mu, k, o;
  face_t f;

  w = 0.1; /* inlet width in m */
  mu = 1.84324 * pow(10., -5.); /* dynamic viscosity */
  begin_f_loop(f,thread)
  {
    F_CENTROID(p, f, thread);
 y = p[1] - 0.5 * w; /* non-dimensional x coordinate */

  k = 0.002*pow(5.,2.)-((y/(0.5*w))*0.04913523734);
  cl = k * pow(0.09,-3./4.);
  a = 2.*cl;
  re = (1.16*y*sqrt(k))/mu;
  le = y * cl * (1-exp(-re/a));
        lu = y * cl * (1-exp(-re/70.));
  vt = 0.09 * lu * sqrt(k);
        o = k/vt;

  F_PROFILE(f, thread, position) = pow(k,3./2.)/le;
  }
  end_f_loop(f, thread)
}

/* k_profile_test*/

#include "udf.h"

DEFINE_PROFILE(inlet_x_k, thread, position)
{

  real p[ND_ND]; /* this will hold the position vector */
  real y, w, m, r, d;
  face_t f;

  w = 0.1; /* inlet width in m */
  m = 1.84324 * pow(10., -5.); /* dynamic viscosity */
  r = 1.16; /* fluid density */
  d = 16.32654493;

  begin_f_loop(f,thread)
  {
    F_CENTROID(p, f, thread);
 y = p[1]; /* non-dimensional x coordinate */

 F_PROFILE(f, thread, position) = 0.002*pow(5.,2.)-(y/(0.5*w))*...
    0.04913523734;
  }
  end_f_loop(f, thread)
}

#include "udf.h"

DEFINE_PROFILE(inlet_x_o, thread, position)
{

  real p[ND_ND]; /* this will hold the position vector */
  real y, w, cl, a, re, lu, le, vt, mu, k, e;
  face_t f;
  w = 0.1; /* inlet width in m */
  mu = 1.84324 * pow(10., -5.); /* dynamic viscosity */

  begin_f_loop(f,thread)
  {
    F_CENTROID(p, f, thread);
 y = p[1] - 0.5 * w; /* non-dimensional x coordinate */

  k = 0.002*pow(5.,2.)-((y/(0.5*w))*0.04913523734);
  cl = k * pow(0.09,-3./4.);
  a = 2.*cl;
  re = (1.16*y*sqrt(k))/mu;
  le = y * cl * (1-exp(-re/a));
    lu = y * cl * (1-exp(-re/70.));
  vt = 0.09 * lu * sqrt(k);
    e = pow(k,3./2.)/le;

  F_PROFILE(f, thread, position) = k / vt;

  }
  end_f_loop(f, thread)
}

/* Tandberg temperature profile*/

#include "udf.h"

DEFINE_PROFILE(inlet_x_temperature, thread, position)
{

 real p[ND_ND]; /* this will hold the position vector */
 real y, w;
 face_t f;

 w = 0.1; /* inlet width in m */

 begin_f_loop(f, thread)
 {
  F_CENTROID(p, f, thread);
  y = p[1]; /* non-dimensional x coordinate, here x[one] means...
        y-position, as in 0,1,2 is x,y,z */

  F_PROFILE(f, thread, position) = (8. * (100 - 50) / 7.) *...
        pow(((0.5 * w - y) / (0.5 * w)), 1. / 7.) + 323.;
  /*Message("hei, a= %f", 0.002 * (8. * (100 - 50) / 7.) *...
        pow(((0.5 * w - y) / (0.5 * w)), 1. / 7.) + 323.);*/

 }
 end_f_loop(f, thread)

}
```

Capture boundary condition

The UDF introduces a memory variable for storing detected trapped particles onto wall cells along the chosen boundary.

```
/* Capture boundary condition for DPM with UDM location*/

#include "udf.h"
#include "mem.h"
#define NUM_UDM 1; /* number of UDMs */
DEFINE_DPM_BC(bc_capture, p, t, f, f_normal, dim)
{
    int hit=0;
        if  (p->type==DPM_TYPE_INERT)
        {
            if (((NNULLP(t)) && (THREAD_TYPE(t) == THREAD_F_WALL)))
            {
                Thread * tread_cell=P_CELL_THREAD(p);
                cell_t c=P_CELL(p);
                C_UDMI(c,tread_cell,0) += 1.0;

                    return PATH_ABORT;
            }

        }
}
```

UDM reset

This UDF resets data stored in the user defined memory location when activated.

```
#include "udf.h"
#include "mem.h"
#define NUM_UDM 1;  /* number of UDMs*/

DEFINE_ON_DEMAND(udm_reset)
{
    int i = 0;
    Domain *d;
    Thread *t;
    cell_t c;

    d=Get_Domain(1);

    Message("Setting all UDM on the domain equal to zero\n");

    thread_loop_c(t,d) /* Loop over all threads in the domain*/
    {
        /*Loop over all cells in the thread*/
        begin_c_loop(c,t)
        {
            C_UDMI(c,t,i) = 0.0;
        }
        end_c_loop(c,t)
    }
    Message("All UDM are reset. DONE\n");

}
```

Brownian force

The following code computes a body force representing brownian diffusion on particles as an extension of the discrete phase model (DPM).

```
#include "udf.h"
#include "mem.h"
#include "dpm.h"
#include "surf.h"
#include "random.h"
#define Kb 1.38e-23 /*Stephan-Boltzmann constant*/
#define pi 3.1415926

DEFINE_DPM_BODY_FORCE(brownian_force, p , i)
{
    real T,Dp,labda,Cd,P,Cc,a,b,d,labda_ref;
    real z,mu_f,rho_f,rho_p,p_dt,ABS_U_V,Re,ueff,udiff,F;
    cell_t c=P_CELL(p);
    Thread *t=P_CELL_THREAD(p);

    /* Extract needed variables from Fluent*/
    labda_ref=0.0664*pow(10.0,-6.0);
    T=C_T(c,t);
    P=C_P(c,t);
    Dp=P_DIAM(p);
    mu_f=C_MU_L(c,t);
 rho_f = C_R(c, t);
    rho_p=P_RHO(p);
    p_dt=P_DT(p);

    ABS_U_V = sqrt(pow(C_U(c,t)-P_VEL(p)[0],2)+pow(C_V(c,t)-...
    P_VEL(p)[1],2)+pow(C_W(c,t)-P_VEL(p)[3],2));

    /* |U-V|, changed [] for x and y */

    Re = rho_f*Dp*ABS_U_V/mu_f;


    a=(101.0e3)/(P+101.0e3);
    b=T/293.0;
    d=(1.0+110.0/293.0)/(1.0+110.0/T);
    labda=labda_ref*a*b*d;
    Cc=1.0+labda/Dp*(2.34+1.05*exp(-0.39*Dp/labda));

    /*Shiller and Neumann correlation from Crowe et al. 2012*/

    Cd=24.0/(Cc*Re)*(1.0+0.15*pow(Re,0.687));
    /* Implementation of Brownian diffusion model */

    z = cheap_gauss_random(); /* generates random number*/

    a = 2.0*Kb*T*Cc;
    b = 3.0*mu_f*pi*Dp*p_dt;
    ueff = sqrt(a/b);
    udiff = z*ueff;

    a = 3.0*Cd*rho_f*ABS_U_V*udiff;
    b = 4*Dp*rho_p*Cc;
    F = a/b;

    return(F); /* Return variable into DPM model*/
}
```

Initial particles

This UDF imparts the particles injected with DPM with the velocity vectors equal to that of the fluid at which cell the particles are injected from.

```
#include "udf.h"
#include "surf.h"
#include "mem.h"

DEFINE_DPM_INJECTION_INIT(init_particles, I)
{
    Particle *p;
    loop (p,I->p)
    {
        cell_t c=P_CELL(p);
        Thread *t=P_CELL_THREAD(p);
        P_VEL(p)[0]=C_U(c,t); /* Velocity in x direction [0]*/
        P_VEL(p)[1]=C_V(c,t); /* Vel. in y dir. */
        P_VEL(p)[2]=C_W(c,t); /* Vel. in z dir. */
```

Published with MATLAB® R2018a