User Defined Function

Содержание

Для чего нужны UDFs?

UDFs

функции пользователя на C или C++, которые можно подгружать в ANSYS Fluent для расширения стандартных возможностей.

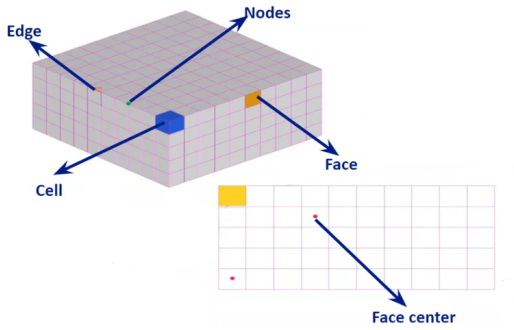
- Граничные условия пользователя.
- Свойства веществ, которых нет в базах данных Fluent (зависимости от температуры и т.п.).
- Дополнительные источниковые члены.
- Решение дополнительных уравнений переноса.
- Расчёт величин после итерационной процедуры.
- Расширение существующих CFD моделей.

Структура UDFs файлов

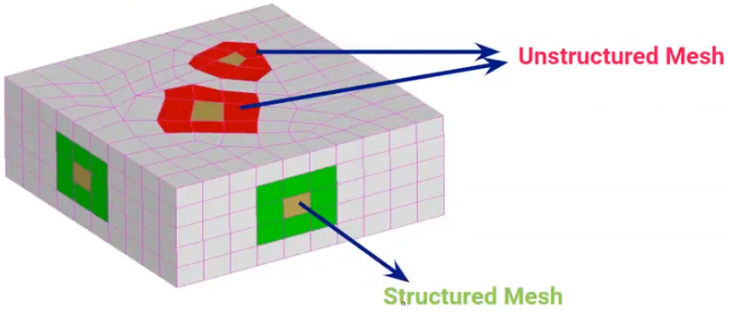
- Файлы с UDFs имеют расширение ***.c** или ***.cpp**. На пример: *boundary.c*.
- Каждый файл имеет заголовок: *#include "udf.h"*.
- Дополнительные заголовки могут содержать *#include "sg.h"*, *#include "mem.h"*, *#include "flow.h"*, *#include "unsteady.h"* в зависимости от задачи.
- Каждый файл может содержать одну и более функций.
- Функции подразделяются на два типа:
  - **Интерпретируемые** (используются для небольших функций без доступа к динамическим данным).
  - **Компилируемые** (Используются для доступа к памяти при решении). Создаются ***.DLL** библиотеки, которые подгружаются по мере необходимости.

Основная терминология

- **Cell** - контрольный объём в рассматриваемой области;
- **Node** - узловые точки расчётной сетки;
- **Cell center** - центр ячейки, в котором хранятся значения переменных;
- **Edge** - граница поверхности;
- **Face** - граница ячейки;
- **Face center** - хранит информацию о поверхности.

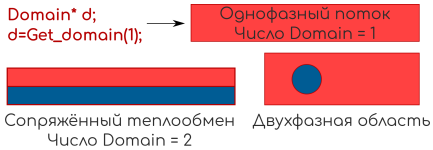


Структурированная и неструктурированная сетки



Структура данных во Fluent

- **Thread** - коллекция определённых сущностей (границы, поверхности, набор ячеек).
- **Domain** - наивысший уровень структуры данных. Хранит в себе все данные, связанные с набором узлов, поверхностей и ячеек среза в сетке.



- Во многофазных потоках домены существуют для каждой фазы, для взаимодействия между фазами, а так же для смеси.

Структура данных во Fluent

- Внутри **Domain** данные организованы по **Threads**.
- **Node thread** - тип данных связанный с узлами сетки (группирование по узлам).
- **Face thread** - данные сгруппированные по поверхностям.
- **face_t** - целочисленный тип данных, который идентифицирует отдельную поверхность в срезе.
  - срез граничных поверхностей
  - срез внутренних поверхностей.
- Все поверхности, которые формируют особенную границу группируются вместе в один *boundary face thread*.
- **Cell thread** и **cell_t** - аналогично для ячеек.

Почему используется C?

- Ansys Fluent написан на языке C. Язык обладает высокой скоростью вычислений, низкоуровневыми возможностями работы с функциями и памятью, высокоуровневыми операциями работы с графикой.
- Это позволяет расширить функционал базовой программы с помощью ”shared object”в Unix и ”Dynamic Linked Library”в Windows.

Основы языка C

Основы языка С

```
return_type function_name(parameter lists)
{
body of the function;
return output;
/* comment */
}
```

- Типы данных:
  - *int* - целочисленный
  - *long* - целочисленный с расширенным диапазоном
  - *float* - число с плавающей запятой
  - *double* - число с плавающей запятой с расширенным диапазоном
  - *char* - один байт памяти - символ

Основы языка С

- Константы
  - #define g 9.81
- Переменные
  - real volume;
- Локальные переменные

```
void function ()
{
int a,b; /* you can use a and b within braces only */
a=5;
b=25;
}
```

Основы языка С

- Глобальные переменные

```
int a,b=10;
void function ()
{
int c;
c=a+b
}
```

Основы языка С

- Массивы

```
int a[10], b[10][10];
a[0]=1;
b[5][5]=4;
```

- Указатели - переменные, указывающие на другую переменную.

```
a=2;
int *ip; /* declares the pointers */
ip = &a; /* assigned the adress of variable a to pointer ip */
*ip = 4; /* changing the value of a using its memory address */
```

Основы языка С
Логические операторы

```
if(logical-expression)
{statements}
else
{statements}
if(x<0.)
{
y=x/50.;
}
else
{
x=-x;
y=x/25.;
}
```

Основы языка С
Циклы

```
int i,j,n=10;
for (i=1; i==n; i++)
{
j=i*i
printf(”%d %d\n”,i,j)
}
```

Основы языка С

Арифметические и логические операции

- = присваивание
- + сложение
- – вычитание
- * умножение
- / деление
- % деление по модулю
- ++ инкремент
- –– декремент

- < меньше чем
- <= меньше или равно
- > больше
- >= больше или равно
- == равно
- ! = не равно

Данные в UDF

Обработка данных в UDF

Данные сетки и типы данных решателя

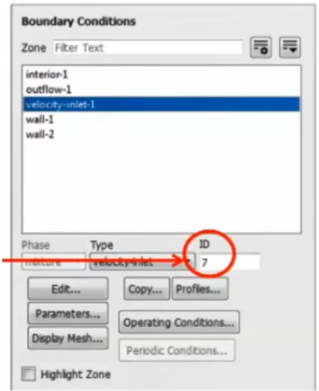
- Некоторые данные сетки и решателя пишутся с заглавной буквы.

Тип	Пример	Описание
Domain	*d;	d - указатель на Domain
Thread	*t;	t - указатель на срез
cell_t	c;	c - индекс ячейки
face_t	f;	f - индекс поверхности
Node	*node	node - указатель на структуру Node

Обработка данных в UDF

- Срезы связаны с ID.

```
int zone_ID = 7;
Thread *thread_name;
tread_name = Lookup_Thread(domain, zone_ID)
```



- Для некоторых макросов: После того как UDF привязывается к определённой поверхности, решатель ”по-умолчанию”привязывает указатель на срез.
- DEFINE_PROFILE(name, thread, position)

```
#include ”udf.h”
DEFINE_PROFILE(pressure_profile,t,i)
{ real x[ND_ND]; /* this will hold the position vector */
real y;
face_t f;
begin_f_loop(f,t)
{
F_CENTROID(x,f,t);
y=x[1];
F_PROFILE(f,t,i)=1.1e5-y*y/(0.0745*0.0745)*0.1e5;
}
end_f_loop(f,t)
}
```

Макросы для написания UDF

- Функции, которые определены в библиотеке udf.h и которые начинаются с DEFINE_:
  - DEFINE_ADJUST
  - DEFINE_DELTAT
  - DEFINE_EXECUTE_AT_END
  - DEFINE_EXECUTE_AT_EXIT
  - DEFINE_EXECUTE_FROM_GUI
  - DEFINE_EXECUTE_AT_LOADING
  - DEFINE_EXECUTE_AFTER_CASE/DATA
  - DEFINE_INIT
  - DEFINE_ON_DEMAND
  - DEFINE_OUTPUT_PARAMETER
  - DEFINE_RW_FILE

Циклы в макросах

```
thread_loop_f(t,d)
{
/* Loop over all face threads in domain d */
}

thread_loop_c(t,d)
{
/* Loop over all cell threads in domain d */
}
```

Циклы в макросах

```
face_t f;
begin_f_loop(f,t)
{
/* Loop over all face threads in a given thread */
}
end_f_loop(f,t)

cell_t c;
begin_c_loop(c,t)
{
/* Loop over all cell threads in a given thread */
}
end_c_loop(c,t)
```

Доступные данные и переменные

Macro	Argument	Returns
C_VOLUME(c,t)	cell_t c, Thread *t	объём ячейки
C_R(c,t)	cell_t c, Thread *t	плотность
C_P(c,t)	cell_t c, Thread *t	давление
C_U(c,t)	cell_t c, Thread *t	компонента скорости U
C_V(c,t)	cell_t c, Thread *t	V
C_W(c,t)	cell_t c, Thread *t	W
C_T(c,t)	cell_t c, Thread *t	температура
C_H(c,t)	cell_t c, Thread *t	энтальпия

Доступные данные и переменные

Macro	Argument	Returns
C_K(c,t)	cell_t c, Thread *t	кинетическая энергия турб.
C_NUT(c,t)	cell_t c, Thread *t	турб. вязкость для Spalart-Allmaras
C_D(c,t)	cell_t c, Thread *t	скорость диссипации k
C_O(c,t)	cell_t c, Thread *t	удельная скорость диссипации
C_YI(c,t,i)	cell_t c, Thread *t, int I	массовая доля компонента
C_CENTROID(x,c,t)	realx[ND_ND], cell_t c, Thread *t	x (центр ячейки)

- real x[ND_ND]

Доступные данные и переменные(Градиенты)

- C_R_G(c,t). Суффикс _G означает, что это градиент.
- C_R_G(c,t)[0]. Градиент вдоль оси X.
- Градиенты ячейки определены в mem.h
- C_R_RG(c,t). Реконструированный градиент.
- C_R_M1(c,t). Градиент с предыдущего шага.
- C_R_M2(c,t). Градиент с пред предыдущего шага.

Макросы

Макрос ND

- Константа ND_ND равна 2 и 3 для 2D и 3D соответственно. Используется для создания матриц.

```
real A[ND_ND][ND_ND];

for (i=0; i<ND_ND;++i)
{
for (j=0; j<ND_ND; ++j)
{
A[i][j]=f(i,j);
}
}
```

Макрос ND

- Утилита ND_SUM рассчитывает сумму ND_ND аргументов

```
ND_SUM(x,y,z)
2D: x+y;
3D: x+y+z;
```

- Утилита ND_SET генерирует ND_ND операторов присваивания

```
real u[ND_ND][ND_ND], v[ND_ND][ND_ND], w[ND_ND][ND_ND];
ND_SET(u,v,w, C_U(c,t), C_V(c,t), C_W(c,t));
u=C_U(c,t);
v=C_V(c,t);
if 3D: w=C_W(c,t);
```

Макрос NV

- Макрос NV аналогичен макросу ND, но работает с векторами, которые являются массивами размерностью ND_ND, вместо работы с отдельными компонентами.
- Утилита NV_V работает как оператор между двумя векторами.

```
NV_V(a,=,x); or
a[0]=x[0]; a[1]=x[1];
```

- Оператор ”+=”вместо -”приведёт к

```
a[0]+=x[0];
```

Макрос NV

- Утилита NV_VV производит действия над элементами вектора. Оператор, который будет применяться над элементами вектора, определяется аргументом (-,+,=)

```
NV_VV(a,=,x,+y)
a[0]=x[0]+y[0]; a[1]=x[1]+y[1];
```

Макрос NV

- Утилита NV_V_VS прибавляет вектор к вектору, который умножен на скаляр.

```
NV_V_VS(a,=,x,+y,*,0.5)
a[0]=x[0]+(y[0]*0.5); a[1]=x[1]+(y[1]*0.5);
```

- Утилита NV_VS_VS прибавляет вектор к вектору, каждый из которых умножен на скаляр.

```
NV_VS_VS(a,=,x,*,2,+y,*,0.5)
a[0]=(x[0]*2)+(y[0]*0.5); a[1]=(x[1]*2)+(y[1]*0.5);
```

Макросы с векторными операциями

- NV_MAG(x) определяет магнитуду вектора.

```
2D: sqrt(x[0]*x[0]+x[1]*x[1]);
3D: sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);
```

- NV_MAG2(x) определяет сумму квадратов компонентов вектора.

```
2D: (x[0]*x[0]+x[1]*x[1]);
3D: (x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);
```

Скалярное произведение векторов

- ND_DOT(x,y,z,u,v,w).

```
2D: (x*u+y*v);
3D: (x*u+y*v+z*w);
```

- NV_DOT(x,u).

```
2D: (x[0]*u[0]+x[1]*u[1]);
3D: (x[0]*u[0]+x[1]*u[1]+x[2]*u[2]);
```

- NVD_DOT(x,u,v,w).

```
2D: (x[0]*u+x[1]*v);
3D: (x[0]*u+x[1]*v+x[2]*w);
```

Векторное произведение векторов

- ND_CROSS_X(x0,x1,x2,y0,y1,y2).

```
2D: 0.0;
3D: x1*y2-y1*x2;
```

- ND_CROSS_Y(x0,x1,x2,y0,y1,y2).

```
2D: 0.0;
3D: x2*y0-y2*x0;
```

- ND_CROSS_Z(x0,x1,x2,y0,y1,y2).

```
2D: 0.0;
3D: x0*y1-y0*x1;
```

Векторное произведение векторов

- ND_CROSS_X(x,y) eq ND_CROSS_X(x[0],x[1],x[2],u[0],y[1],y[2]).
- ND_CROSS_Y(x,y) eq ND_CROSS_Y(x[0],x[1],x[2],u[0],y[1],y[2]).
- ND_CROSS_Z(x,y) eq ND_CROSS_X(x[0],x[1],x[2],u[0],y[1],y[2]).

- NV_CROSS(a,x,y).

```
a[0]=NV_CROSS_X(x,y);
a[1]=NV_CROSS_Y(x,y);
a[2]=NV_CROSS_Z(x,y);
```

Свойства

Свойства веществ

$$\mu = Ae^{B/T} \tag{1}$$

Вещество	Формула	A [mPa s]	B [K]	Диапазон температур
Бром	Br_2	0.0445	907.6	269-302

DEFINE_PROPERTY

```
#include "udf.h"
#include "math.h"
real A = 0.0445;
real B = 907.6;
DEFINE_PROPERTY(cell_viscosity, c, t)
{
   real mu_lam;
   real temp = C_T(c, t);
   if (temp > 269 && temp < 302)
      mu_lam = A*exp(B/temp);
   else
      mu_lam = 0.00095;
   return mu_lam;
}
```

Изменение расхода по синусоиде

```
#include "udf.h"
#include "math.h"
real pi = 3.141;
DEFINE_PROFILE(sine_mass_flow, t, i)
{
   face_t f;
   real flow_max = 3.5, flow_min = 1.5, flow_amp;
   flow_amp = (flow_max-flow_min);
   real flow_time = CURRENT_TIME;
   begin_f_loop(f,t)
   {
      F_PROFILE(f,t,i)=flow_min+0.5*flow_amp+0.5*flow_amp*sin(2*pi*flow_time);
   }
   end_f_loop(f,t)
}
```