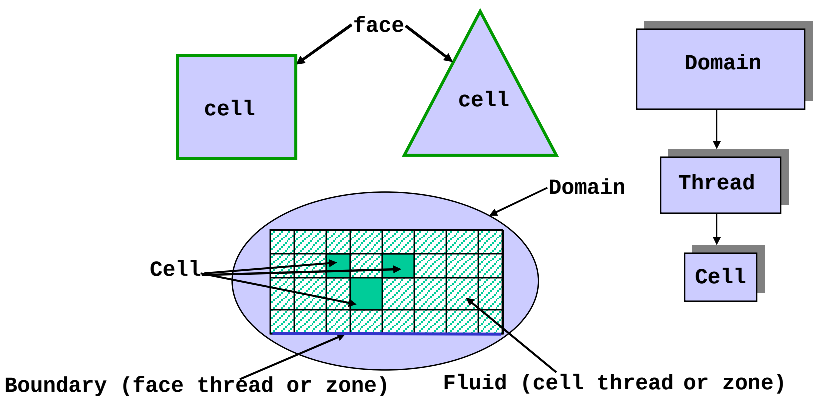
Fluent Data Structure and Macros

Advanced UDF Modeling Course

Data structures in FLUENT



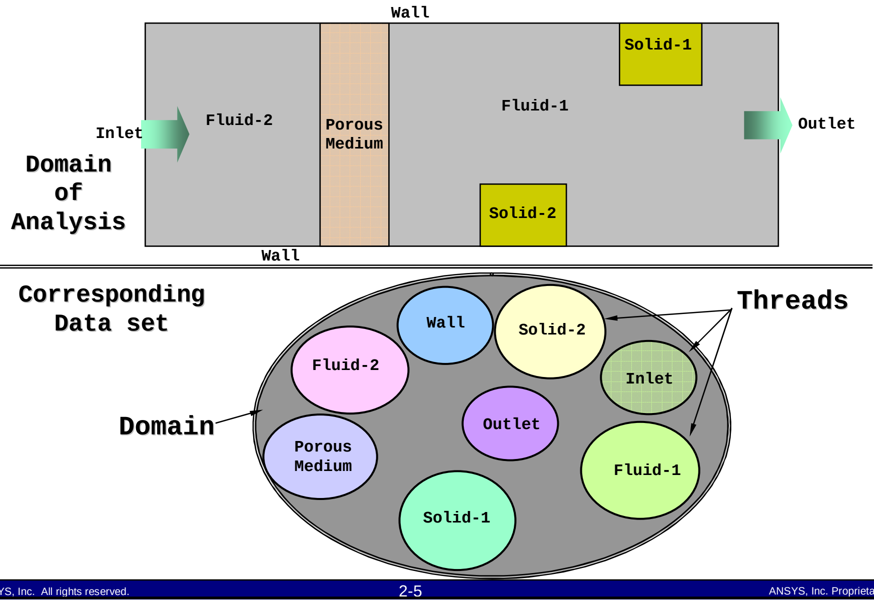
The Domain

- "Domain" is the set of connectivity and hierarchy info for the entire data structure in a given problem. It includes:
  - » all fluid zones ('fluid threads')
  - » all solid zones ('solid threads')
  - » all boundary zones ('boundary threads')
- Cell/face - Computational unit, face is one side. Conservation equations are solved over a cell
- Thread - is the collection of cells or faces; defines a fluid/solid/boundary zone
- FLUENT6 introduces the concept of multi-"domain" for multiphase simulations (singlephase simulations use single domain only)
  - Each phase has its own "Domain-structure"
  - Geometric and common property information are shared among 'sub-domains'
  - Multiphase UDF will be discussed later

The Threads

- A '**`Thread`**' is a sub-set of the '**`Domain`**' structure
- Individual '**`fluid`**', '**`solid`**' and each '**`boundary`**' zones are identified as '**`zones`**' and their datatype is maintained as '**`Thread`**'
- '**`Zone`**' and '**`Thread`**' terms are often used interchangeably
- But **`Zone/Thread ID`** and **`Thread-datatype`** are different:
  - Zones are identified at mesh level with an 'integer' **`ID`** in the **`Define → Boundary Condition`** panel
  - **`Threads,`** a Fluent-specific datatype, that store structured information about the mesh, connectivity, models, property, etc. all in one place
  - Users identify zones through the **`ID`**'s
  - **`Zone/Thread-ID`** and **`Thread`**s are correlated through UDF macro's

Domain and Threads



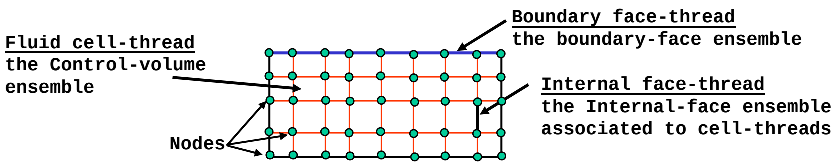
Cell and Face Datatypes

- Control volumes (equivalent of 'FEM:Elements') of fluid and solid zones are called '**cell**s' in FLUENT
  - The data structure for the cell zones is typed as '**cell_t**' (the cell thread)
  - The data structure for the cell faces is typed as '**face_t**' (the face thread)
- A fluid or solid zone is called a cell zone, which can be accessed by using cell threads
- Boundary or internal faces can be accessed by using face threads

Some additional info on Faces

- Each Control volume will have a finite number of faces (4 for tets, 6 for hex and 5 for pyramids, and wedges)
  - Faces on the boundary are also typed '**face_t**'; their ensemble are listed as boundary **face-threads** with the fluid & solid cell-threads under **Define-Boundary_Condition** panel
  - Those faces which are inside the flow-domain and do not share any external boundary are not accessible from GUI (because you do not need them)
  - They can still be accessed from User-Defined-Functions

Cell- & face-Threads



Type	Example	Details
Domain	***d**	pointer to the collection of all threads
Thread	***t**	pointer to a thread
cell_t	**c**	cell identifier
face_t	**f**	face identifier
Node	***node**	pointer to a node

Geometry Macros

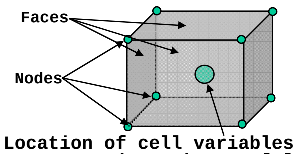
The argument **(c,t)** stands for a cell, c of a thread, t

- **C_NNODES(c, t);** Number of nodes in a cell
- **C_NFACES(c, t);** No. of faces in a cell
- **F_NNODES(f, t);** No. of nodes in a face
- **C_CENTROID(x, c, t);** x, y, z-coords of cell centroid
- **F_CENTROID(x, f, t);** x, y, z-coords of face centroid
- **F_AREA(A, f, t);** Area vector of a face;
- **NV_MAG(A);** Area-magnitude
- **C_VOLUME(c, t);** Volume of a cell
- **C_VOLUME_2D(c, t);** Volume of a 2D cell

(Depth is 1m in 2D; 2*π m in axisymmetric)

- **NODE_X(nn);** Node x-coord;
- **NODE_Y(nn);** Node x-coord;
- **NODE_Z(nn);** Node x-coord;

A Hex cell



Location of cell variables
C_CENTROID(X,c,t); X: X[3]

```
C_NNODES(c,t) = 8
C_NFACES(c,t) = 6
F_NNODES(f,t) = 4 each
```


Looping Macros for Geometry

- **thread_loop_c(t, d);** Loop over cell threads
- **thread_loop_f(t, d);** Loop over face threads
- **begin_c_loop(c, t);**
- **end_c_loop(c, t);** } Loop over cells in a cell thread
- **begin_f_loop**
- **end_f_loop** } Loop over faces in a face thread
- **f_edge_loop(f, t,en);** Loop over edges in a face thread
- **f_node_loop(f, t,nn);** Loop over nodes in a face thread
- **c_node_loop(c, t,nn);** Loop over nodes in a cell thread
- **c_face_loop(c, t,fn);** Loop over faces in a cell thread

Pointer to a Thread

- Given the integer **ID** of a **thread**, it is possible to retrieve the pointer to that thread -

```
int ID = 1;
Thread *tf = Lookup_Thread(domain, ID);
```

- Conversely, given the pointer to a **thread**, it is possible to retrieve the integer **ID** of that **thread** -

```
int ID = 1;
if (THREAD_ID(tf)==1)...
```

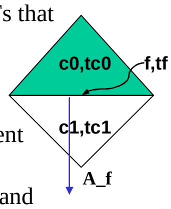
```
int ID = 1;
Thread *tf =
   Lookup_Thread(domain,ID);
begin_f_loop(f, tf)
 {
  F_CENTROID(FC, f, tf);
  printf("x:%f y:%f",FC[0],
  FC[1]);
 }
end_f_loop(f, tf)
```

```
int ID = 1;
thread_loop_f (tf, domain)
 {
  if (THREAD_ID(tf)==1)
   begin_f_loop(f, tf)
    {
     F_CENTROID(FC, f, tf);

     printf("x:%f y:%f",FC[0],FC[1]);
    }
   end_f_loop(f, tf)
 }
```


Cells across a face and Their Threads

- These macros identify the neighboring cells of a face
- This information may be required of some of the more sophisticated UDFs that loop through
  - faces of a boundary thread or
  - a particular cell



- Associated with a given face f, and its thread tf, are potentially two adjacent cells denoted `c0` and `c1` (face normals are always pointing outwardly)
  - If the face is on the boundary of the domain, `c1` is defined as NULL and only `c0` exists
- The following macros return the ID of the cells `c0` and `c1`, as well as the associated threads:

c0 = F_C0(f,tf);	/* returns thread ID for cell c0*/
tc0 = THREAD_T0(tf);	/* returns the cell thread pointer for c0 */
c1 = F_C1(f,tf);	/*returns thread ID for c1 */
tc1 = THREAD_T1(tf);	/* returns the cell thread pointer for c1 */

Cell Variables (1)

`C_R(c,t)`	Density
`C_P(c,t)`	Pressure
`C_U(c,t)`	Velocity components
`C_V(c,t)`	
`C_W(c,t)`	
`C_T(c,t)`	Temperature
`C_H(c,t)`	Enthalpy
`C_K(c,t)`	Turbulent kinetic energy
`C_D(c,t)`	Turbulent energy dissipation
`C_YI(c,t,i)`	Species mass fraction
`C_UDSI(c,t,i)`	User defined scalar

Cell Variables (2)

`C_DUDX(c,t)`	Velocity derivatives
`C_DUDY(c,t)`	
`C_DUDZ(c,t)`	
`C_DVDX(c,t)`	
`C_DVDY(c,t)`	
`C_DVDZ(c,t)`	
`C_DWDX(c,t)`	
`C_DWDY(c,t)`	
`C_DWDZ(c,t)`	
`C_MU_L(c,t)`	Viscosities
`C_MU_T(c,t)`	
`C_MU_EFF(c,t)`	
`C_DP(c,t)[i]`	Pressure derivatives
`C_D_DENSITY(c,t)[i]`	Density derivatives

Cell Variables (3)

- `C_K_L(c,t)` } Thermal conductivities
- `C_K_T(c,t)`
- `C_K_EFF(c,t)`
- `C_CP(c,t)` Specific heat
- `C_RGAS(c,t)` Gas constant
- `C_DIFF_L(c,t,i)` } Species diffusivity
- `C_DIFF_EFF(c,t,i)`

Face Variables

- `F_P(f,t)` Pressure
- `F_U(f,t)` } Velocity components
- `F_V(f,t)`
- `F_W(f,t)`
- `F_T(f,t)` Temperature
- `F_H(f,t)` Enthalpy
- `F_K(f,t)` Turbulent kinetic energy
- `F_D(f,t)` Turbulent energy dissipation
- `F_YI(f,t,i)` Species mass fraction
- `F_UDSI(f,t,i)` User defined scalar
- `F_PROFILE(f,t,i)` Boundary profile storage

UDF Macro-s (Types of UDF)

- UDF's in FLUENT are available for:
  - Boundary conditions : **`Profiles`**
  - Fluid and solid zones : **`Source terms`**
  - Fluid/solid, particle, flow : **`Properties`**
  - UDS unsteady, flux, diffusivity : **`Scalar Functions`**
  - Zone and variable specific initialization : **`Initialization`**
  - Adjust, read/write, execute_on_demand : **`Global Function`**
  - Convective & radiative : **`Wall-heat-flux`** *(Alternative: profile)*
  - Reaction rates, dpm, slip velocity,… : **`Model Specific Functions`**

UDF Macro-s (Types of UDF)

- Available UDF Macro-s :
  - Profiles : **`DEFINE_PROFILE`**
  - Source terms : **`DEFINE_SOURCE`**
  - Properties : **`DEFINE_PROPERTY`**
  - Scalar Functions : **`DEFINE_UNSTEADY`**
`DEFINE_FLUX`
`DEFINE_DIFFUSIVITY`
  - Initialization : **`DEFINE_INIT`**
  - Global Functions : **`DEFINE_ADJUST`**
`DEFINE_ON_DEMAND`
`DEFINE_RW_FILE`
  - Wall-heat-flux : **`DEFINE_HEAT_FLUX`**
  - Model Specific Functions : **`DEFINE_DPM_…`**
`DEFINE_SR_RATE`
`DEFINE_VR_RATE`
`DEFINE_SCAT_PHASE_FUNC`
`DEFINE_DRIFT_DIAMETER`
`DEFINE_SLIP_VELOCITY`

The udf.h File

- The udf-macros are defined in the **'udf.h'** file
- **udf.h** is a fluent header file in the **~/Fluent.Inc/Fluentx.y/src/** directory
- **udf.h** must be included at the top in each and every udf file
  - A file may contain more than one UDF
  - User can use multiple files for UDF
- Any UDF you might write **must** use one of the **'DEFINE_…'** macros from this **udf.h** file

Part of the 'udf.h' file from ~/Fluent.Inc/fluentx.y/src directory

```
#define DEFINE_PROFILE(name, t, i) void name(Thread *t, int i)
#define DEFINE_PROPERTY(name,c,t) real name(cell_t c, Thread *t)
#define DEFINE_SOURCE(name, c, t, dS, i) \
        real name(cell_t c, Thread *t, real dS[], int i)
#define DEFINE_INIT(name, domain) void name(Domain *domain)
#define DEFINE_ADJUST(name, domain) void name(Domain *domain)
#define DEFINE_DIFFUSIVITY(name, c, t, i) \
        real name(cell_t c, Thread *t, int i)
```