Grid Data Structures and Algorithms

Jialin Lou

Soochow University

Motivation

- Why:
  - ➢Play a Major Role in Any Filed Solver
  - ➢Enable Rapid Access of Data
  - ➢Have Major Impact on CPU/Memory Requirements

Representation of Grid

- Points: **Coordinates**

  COORD(1:NDIMN, 1:NPOIN)

  Where:

  COORD: Coordinates of the Points; **NDIMN**: Number of Dimensions; **NPOIN**: Number of Points

- Elements: **Connectivity Matrix**

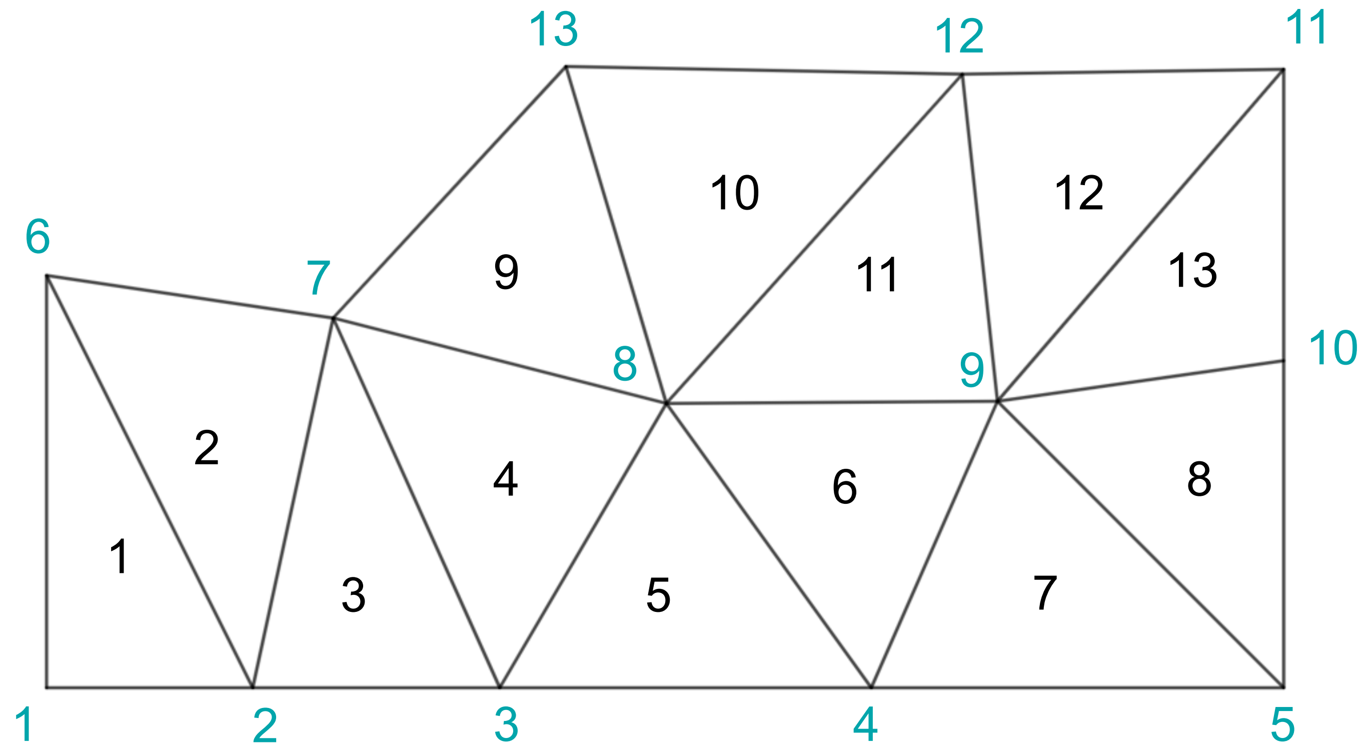
  INPOEL(1:NNODE, 1:NELEM)

  Where:

  INPOEL: Points in Each Element; **NNODE**: Number of Nodes of Each Element; **NELEM**: Number of Elements

These Two Arrays Define the Grid

Typical Mesh



Unknowns

- Unknowns:

  UNKNP(1:NUNKP, 1:NPOIN)

  UNKNE(1:NUNKE, 1:NELEM)

Where:

UNKNP: Unknowns at Points; **UNKNE**: Unknowns at Elements;

NUNKP: Number of Unknowns per Point; **NUNKE**: Number of Unknows per Element.

Remark:

In practice, one can store DG solution as **UNKNE**(1:NDEGR,1:NEQN,1:NELEM)

where: **NDEGR**: Number of Degrees of Freedom per Variable, **NEQN**: Number of Equations

Boundary Conditions

- B.Cs:

  BCOND(1:NCONI, 1:NBFAC)

Where:

  BCOND: Boundary Condition Array;

  NCONI: Number of B.C. Information; **NBFAC**: Number of Boundary Faces

Remark:

  In 2D, default setting for NCONI = 3, where it stores ip1, ip2 (point index) and boundary condition type/flag.

Derived Data Structures

- **Static or Dynamic Data?**
- **Memory vs CPU?**

For Static Data:

- Elements Surrounding Points – Linked Lists
- Points Surrounding Points
- Elements Surrounding Elements
- Edges
- Edges of An Element
- External Faces

For Dynamic Data:

- N-Trees
- Heap Lists
- Bins
- Binary Trees
- Quadtrees and Octrees
- Nearest Neighbors and Graphs
- Distance to Wall

Elements Surrounding Points

- Dilemma: Number of Elements Surrounding a Point Different from Point to Point
- Options:

  "Leave Space": Fast, But Excessive Storage

  Linked List: Most Compact Storage Scheme, But Slower

Linked List:

ESUP1 (1:MESUP), ESUP2 (1:NPOIN+1)

where

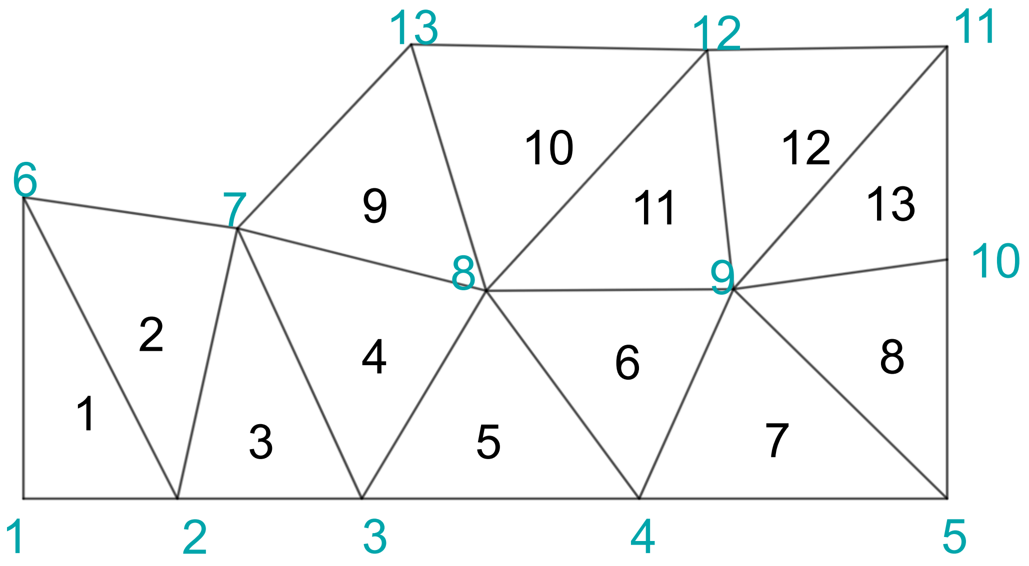
ESUP1: Stores the Element

ESUP2: Stores the Storage Locations

Elements Surrounding Point IPOIN: Stored in **ESUP2(IPOIN)+1** to **ESUP2(IPOIN+1)**

Elements Surrounding Points

- **EX**



ESUP1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
1	1	2	3	3	4	5	5	6	7	7	8	1	2	2	3	4	9	4	5	6	9	10	11	6	7	8	11	12	13	8	13	12	13	10	11	12	9	10

ESUP2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	4	7	10	12	14	18	24	30	32	34	37	39

Elements Surrounding Points

- **Two-Pass Strategy:**

Pass 1: Count Storage Requirements

Pass 2: Store Elements

Element Pass 1:

```
!Initialization
ESUP2(1:NPOIN+1)=0
!Loop over the elements
DO IELEM = 1, NELEM
  !Loop over Nodes of the element
  DO INODE = 1, NNODE
    !Update Storage Counter, Storing Ahead
      ESUP2(INPOEL(INODE,IELEM)+1)=ESUP2(INPOEL(INODE,IELEM)+1)+1
  END DO
END DO
```

Elements Surrounding Points

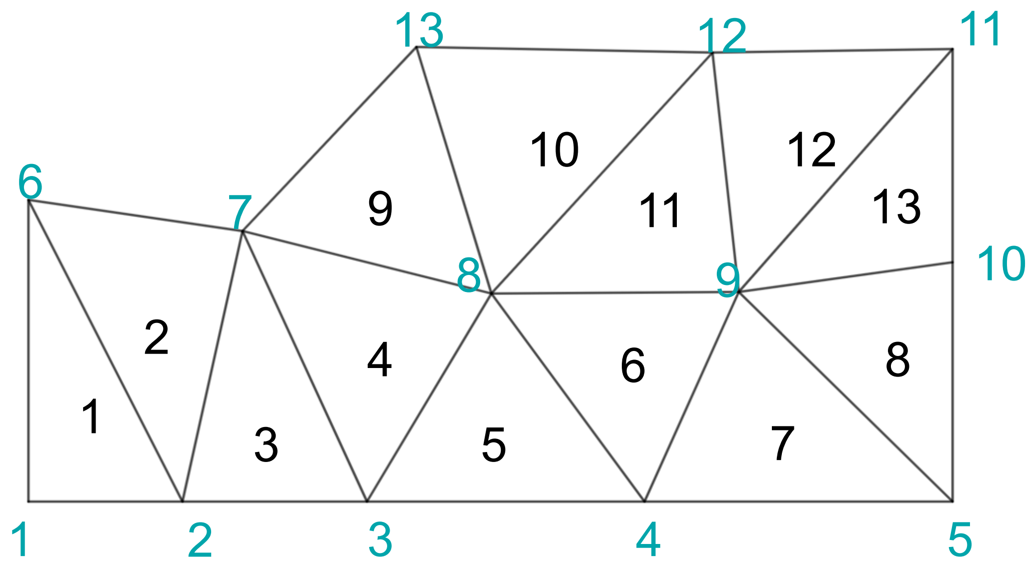
ESUP2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	3	3	3	2	2	4	6	6	2	2	3	2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	4	7	10	12	14	18	24	30	32	34	37	39

Storage/ Reshuffling Pass 1:

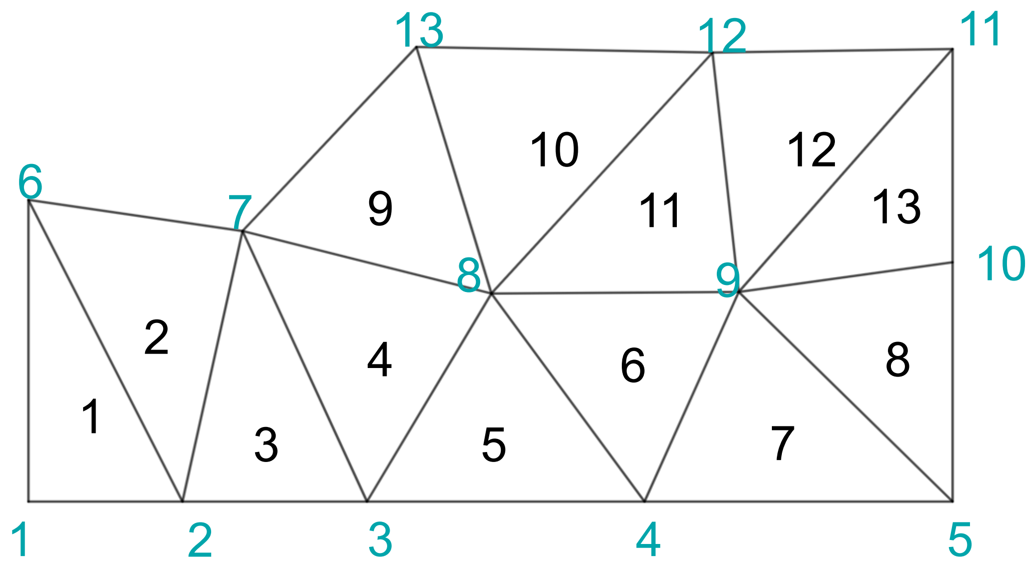
```
!Loop over the points
DO IPOIN = 2, NPOIN+1
  !Update Storage Counter and store
  ESUP2(IPOIN)=ESUP2(IPOIN)+ESUP2(IPOIN-1)
END DO
MESUP=ESUP2(NPOIN+1)
```



Elements Surrounding Points

Element Pass 2:

```
!Loop over the elements
DO IELEM = 1, NELEM
  !Loop over Nodes of the element
  DO INODE = 1, NNODE
    !Update Storage Counter, Storing in ESUP1
      IPOIN=INPOEL(INODE,IELEM)
      ISTOR=ESUP2(IPOIN)+1
      ESUP2(IPOIN)=ISTOR
      ESUP1(ISTOR)=IELEM
  END DO
END DO
```



ESUP2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	10	12	14	18	24	30	32	34	37	39	39

ESUP1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
1	1	2	3	3	4	5	5	6	7	7	8	1	2	2	3	4	9	4	5	6	9	10	11	6	7	8	11	12	13	8	13	12	13	10	11	12	9	10

Elements Surrounding Points

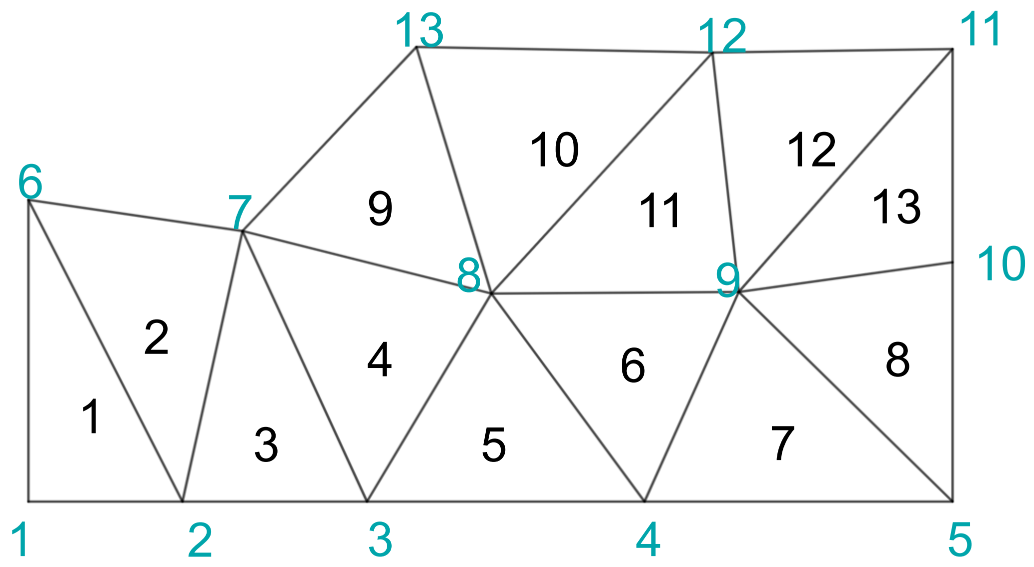
Storage/ Reshuffling Pass 2:

```
!Loop over points, In reverse order
DO IPOIN = NPOIN+1, 2, -1
  ESUP2(IPOIN)=ESUP2(IPOIN-1)
END DO
ESUP2(1)=0
```

ESUP2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	10	12	14	18	24	30	32	34	37	39	39

1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	4	7	10	12	14	18	24	30	32	34	37	39



Points Surrounding Points

- As Before, use **Linked List** with Two Pass Strategy

Linked List:

PSUP1 (1:MPSUP), PSUP2 (1:NPOIN+1)

where:

PSUP1: Stores the Points

PSUP2: Stores the Storage Locations

Points Surrounding Point IPOIN: Stored in **PSUP2(IPOIN)+1** to **PSUP2(IPOIN+1)**

Elements Surrounding Elements

- Stored in

  ESUEL (1:NFAEL, 1:NELEM)

where:

  ESUEL: Elements Surrounding Elements

  NFAEL: Number of Faces of an Element (For triangle elements, NFAEL=3)

Built with:

  -**INPOEL**, **ESUP1**, **ESUP2**

  -Help-Arrays: **LPOIN**(1:NPOIN), **LHELP**(NNOFA), **LNOFA**(NNOFA,NFAEL)

where:

  NNOFA: Number of nodes per face (In 2D, NNOFA=2)

  NFAEL: Number of faces per element (For triangle elements, NFAEL=3)

Elements Surrounding Elements

- Example

ESUEL(1,IE)=KE

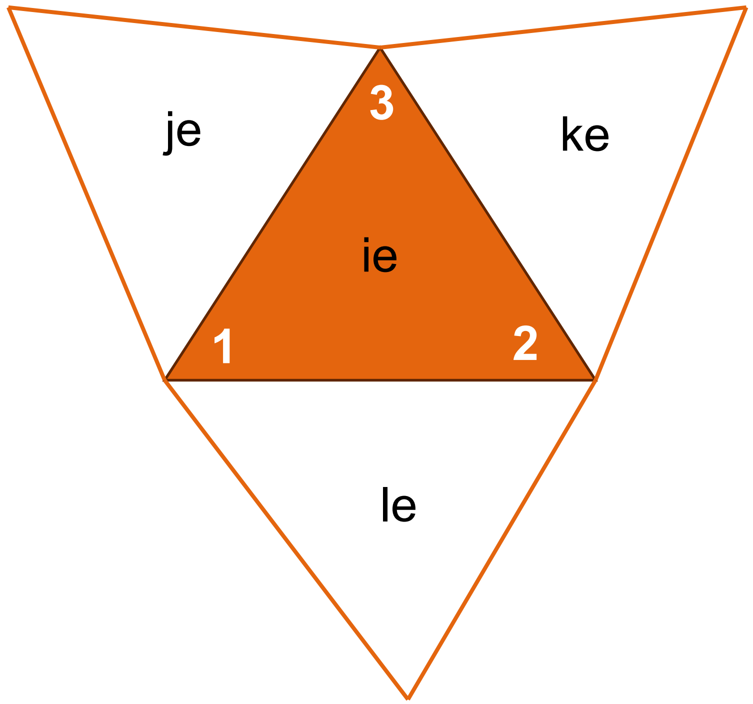
ESUEL(2,IE)=JE

ESUEL(3,IE)=LE

LNOFA is an auxiliary array, which indicates the start and ending local node index of each faces

For triangle, we have

LNOFA(1,1)=2, LNOFA(2,1)=3
LNOFA(1,2)=3, LNOFA(2,2)=1
LNOFA(1,3)=1, LNOFA(2,3)=2



Elements Surrounding Elements

```
!Initialization
LPOIN=0, ESUEL=0
!Loop over the elements
DO IELEM = 1, NELEM
  !Loop over the Faces of the element IELEM
  DO IFAEL = 1, NFAEL
    !Obtain the nodes of this face, and store the points in LHELP
    LHELP(:)=INPOEL(LNOFA(:,IFAEL),IELEM)
    LPOIN(LHELP(:))=1 !Mark in LPOIN
    IPOIN=INPOEL(INODE,IELEM) !Select a Point of the face
    !Loop over the Elements Surrounding Point IPOIN
    DO  ISTOR=ESUP2(IPOIN)+1, ESUP2(IPOIN+1)
      JELEM=ESUP1(ISTOR)
      IF (JELEM .NE. IELEM) THEN
      !Loop over the faces of the Element JELEM
        DO JFAEL=1, NFAEL
          !Count the number of equal entries
          ICOUN=0
          DO JOFA = 1, NNOFA
            JPOIN = INPOEL(LNOFA(JNOFA,JFAEL), JELEM)
            IF (LPOIN(JPOIN) .EQ. 1) ICOUN = ICOUN+1
          END DO
          IF (ICOUN .EQ. NNOFA) THEN
          !Store the Elements
            ESUEL(IFAEL, IELEM) = JELEM
            ESUEL(JFAEL,JELEM) = IELEM
          END IF
        END DO
      END IF
    END DO
    LPOIN(LHELP(1:NNOFA))=0 !Reset LPOIN
  END DO
END DO
```

Q: How about boundary cells?

A: ESUEL(IFAEL,IELEM)=0 if there is no JELEM to be found.

Q: How to improve the current code?

```
!Initialization
LPOIN=0, ESUEL=0
!Loop over the elements
DO IELEM = 1, NELEM
 !Loop over the Faces of the element IELEM
 DO IFAEL = 1, NFAEL
  !Obtain the nodes of this face, and store the points in LHELP
  LHELP(:)=INPOEL(LNOFA(:,IFAEL),IELEM)
  LPOIN(LHELP(:))=1 !Mark in LPOIN
  IPOIN=INPOEL(INODE,IELEM) !Select a Point of the face
  FLAG = 0 !Flag for boundary faces
  !Loop over the Elements Surrounding Point IPOIN
  DO  ISTOR=ESUP2(IPOIN)+1, ESUP2(IPOIN+1)
     JELEM=ESUP1(ISTOR)
     IF (JELEM .NE. IELEM) THEN
     !Loop over the faces of the Element JELEM
       DO JFAEL=1, NFAEL
         !Count the number of equal entries
         ICOUN=0
         DO JOFA = 1, NNOFA
           JPOIN = INPOEL(LNOFA(JNOFA,JFAEL), JELEM)
           IF (LPOIN(JPOIN) .EQ. 1) ICOUN = ICOUN+1
         END DO
         IF (ICOUN .EQ. NNOFA) THEN
         !Store the Elements
           ESUEL(IFAEL, IELEM) = JELEM
           ESUEL(JFAEL,JELEM) = IELEM
           FLAG = 1
         END IF
       END DO
     END IF
  END DO
  IF (FLAG .EQ. 0) THEN
   DO IFACE = 1, NBFAC
     IF ((BFACE(1,IFACE) .EQ. LHELP(1)) .AND.
        (BFACE(2,IFACE) .EQ. LHELP(2)))
       ESUEL(IFAEL,IELEM)=IFACE + NELEM
       EXIT
     END IF
   END DO
  END IF
  LPOIN(LHELP(1:NNOFA))=0 !Reset LPOIN
 END DO
END DO
```

Number of Total Faces

- Number of boundary faces is given when mesh is provided.
- You may need number of face-related information if your code need to loop over faces many times.
- To begin with, you need **NAFAC**: number of all faces.
- **NAFAC** = **NBFAC** + **NIFAC**, where **NIFAC**: number of internal faces.

```
NAFAC = NBFAC ! Initialization
!Loop over the elements
DO IELEM = 1, NELEM
  !Loop over Neighbors of the element
  DO IFAEL = 1, NFAEL
    JELEM=ESUEL(IFAEL, IELEM)
    IF (JELEM .GT. IELEM) .AND. (JELEM .LE. NELEM)
      NAFAC = NAFAC + 1
  END DO
END DO
```

Interface Array

- For DG/FV related computation, loop over faces are very common.
- It is suggested that you have an interface array to store necessary information to speed up the computation.
- **INTFAC**(NIFFA, NAFAC)

where:

INTFAC: interface array (face-connecting array)

NIFFA: number of face related information that need to be stored (at least 2, to store ieL and ieR (make sure ieL<ieR. For boundary faces, ieR=iface+Nelem. You may want to increase to 4 to store ip1 and ip2 as well.)

- With INTFAC, you can also build **FSUEL**(faces surrounding element) easily if you need.

Geometry Arrays

- You may want to build some geometry related arrays. Suggestions are **GEOEL** and **GEOFAC**.
- **GEOEL**: you can store some useful elemental geometry information. For instance, cell center coordinates, volume/area, etc.
- **GEOFAC**: you can store some useful interfacial geometry information. For instance, outward unit normal vector, face area/length, etc.