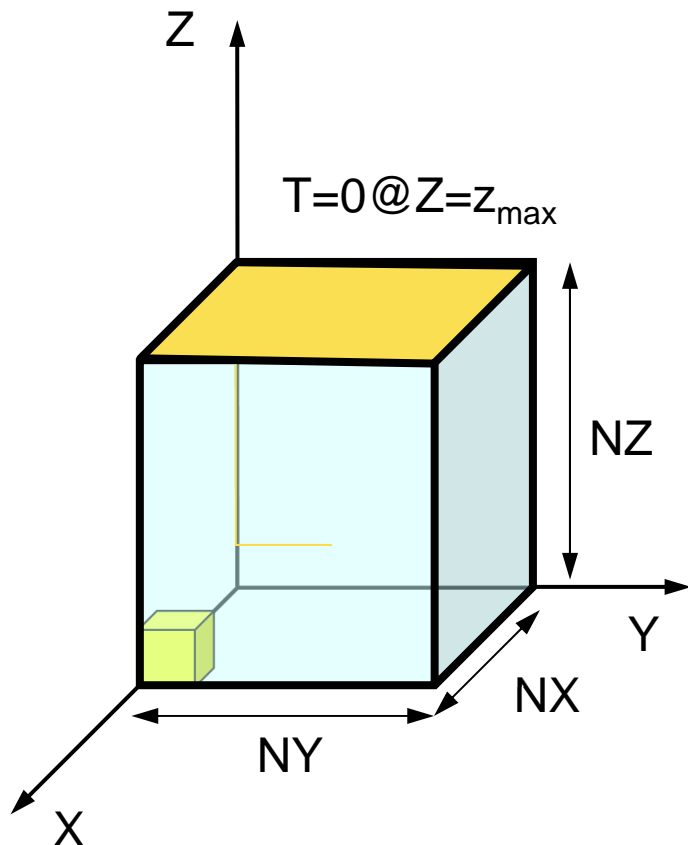


# **3D-FEM in C: Steady State Heat Conduction**

Kengo Nakajima  
Information Technology Center  
The University of Tokyo

# 3D Steady-State Heat Conduction

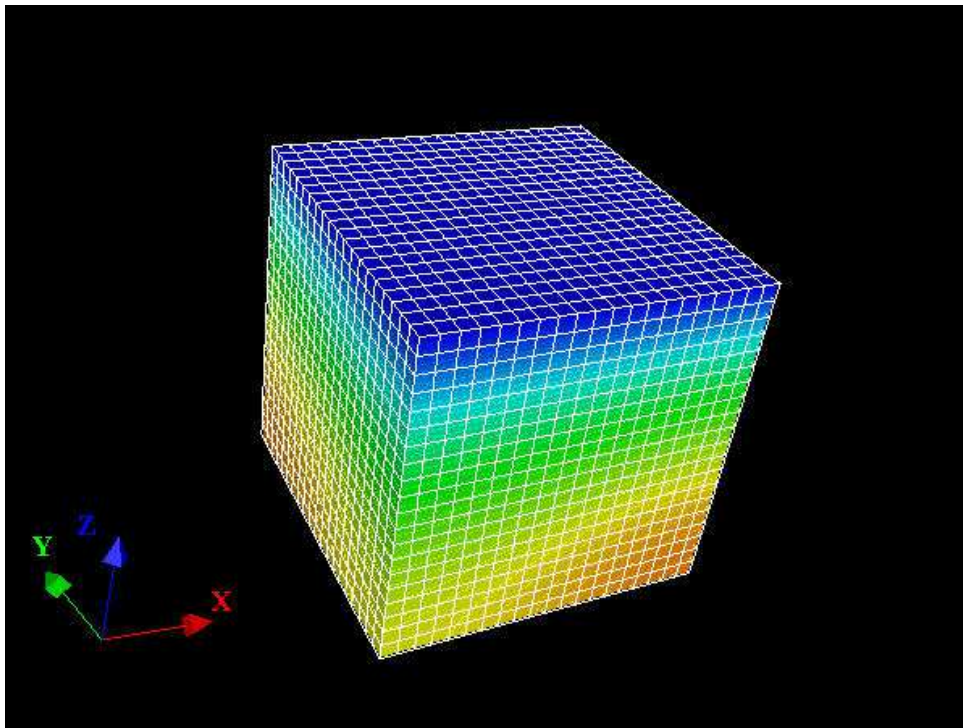
$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$



- Heat Generation
- Uniform thermal conductivity  $\lambda$
- HEX meshes
  - 1x1x1 cubes
  - NX, NY, NZ cubes in each direction
- Boundary Conditions
  - T=0@Z=z<sub>max</sub>
- Heat Gen. Rate is a function of location (cell center:  $x_c, y_c$ )
  - $\dot{Q}(x, y, z) = QVOL|x_c + y_c|$

# 3D Steady-State Heat Conduction

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$



- Higher temperature at nodes far from the origin.
- Heat Gen. Rate is a function of location (cell center:  $x_c, y_c$ )

$$\dot{Q}(x, y, z) = |x_c + y_c|$$

movie

# Finite-Element Procedures

- Governing Equations
- Galerkin Method: Weak Form
- Element-by-Element Integration
  - Element Matrix
- Global Matrix
- Boundary Conditions
- Linear Solver

# FEM Procedures: Program

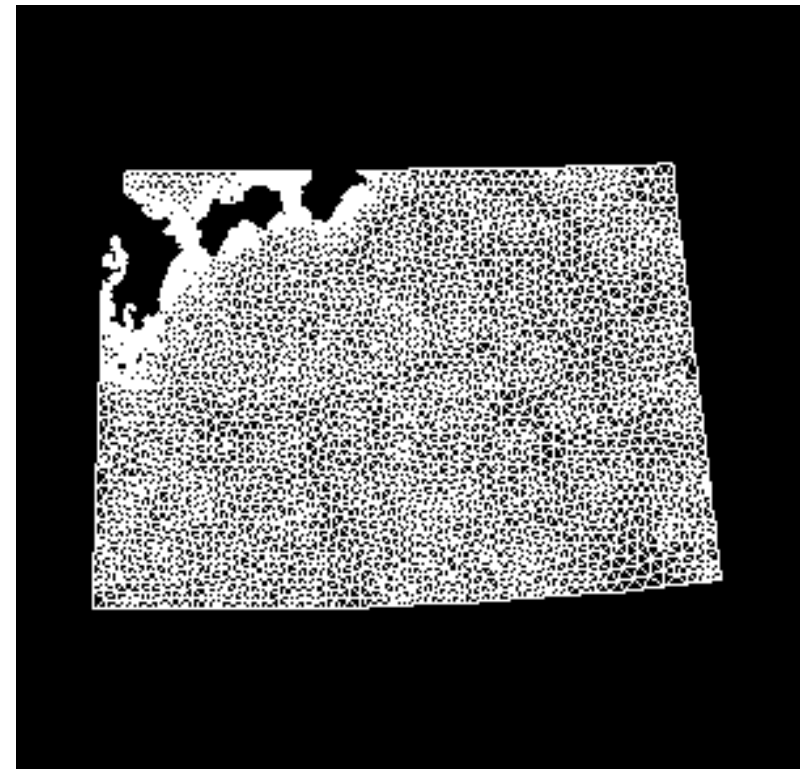
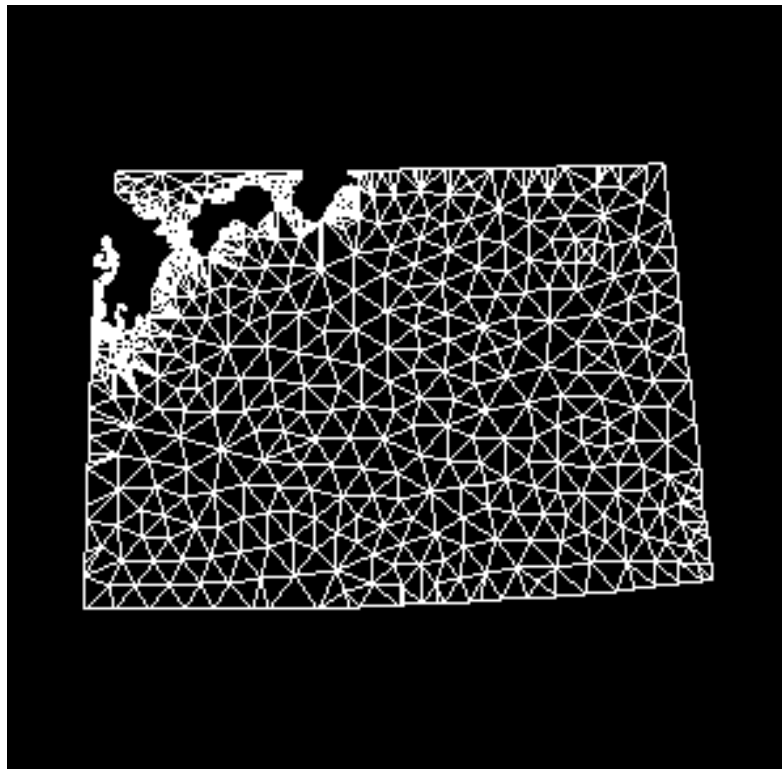
- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method

- Formulation of 3D Element
- 3D Heat Equations
  - Galerkin Method
  - Element Matrices
- Running the Code
- Data Structure
- Overview of the Program

# Extension to 2D Prob.: Triangles

## 三角形要素

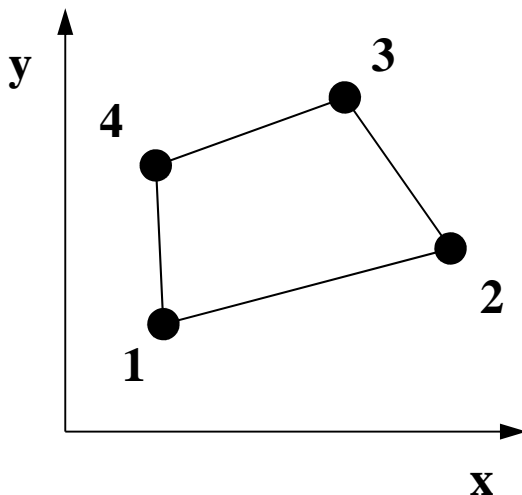
- Triangles can handle arbitrarily shaped object
- “Linear” triangular elements provide low accuracy, therefore they are not used in practical applications.



# Extension to 2D Prob.: Quadrilaterals

## 四角形要素

- Formulation of quad. elements is possible if same shape functions in 1D elements are applied along X- and Y- axis.
  - More accurate than triangles
- Each edge must be “parallel” with X- and Y- axis.
  - Similar to FDM

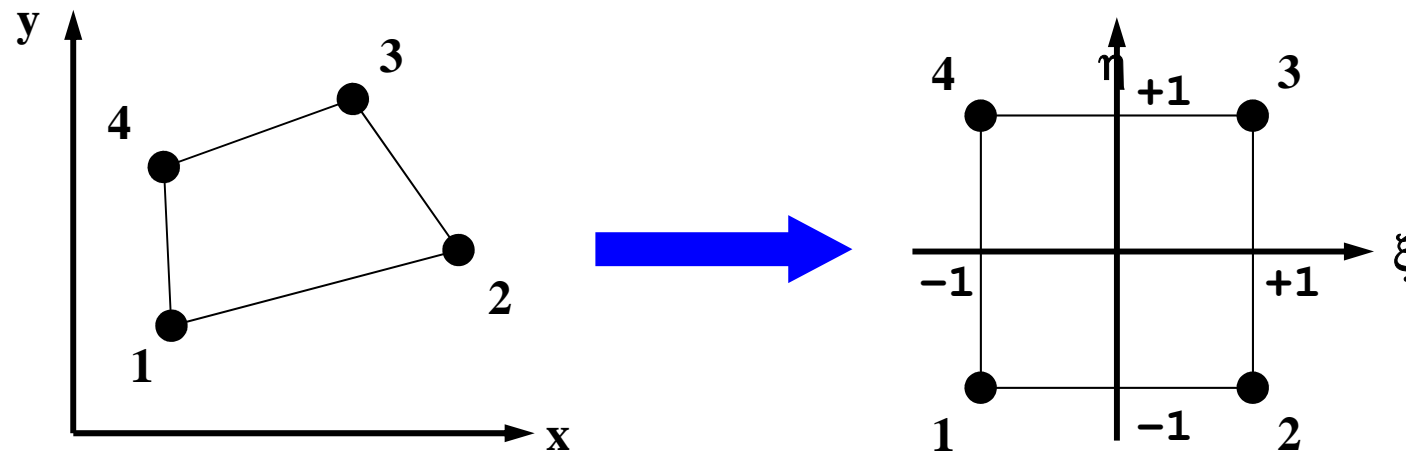


- This type of elements cannot be considered.



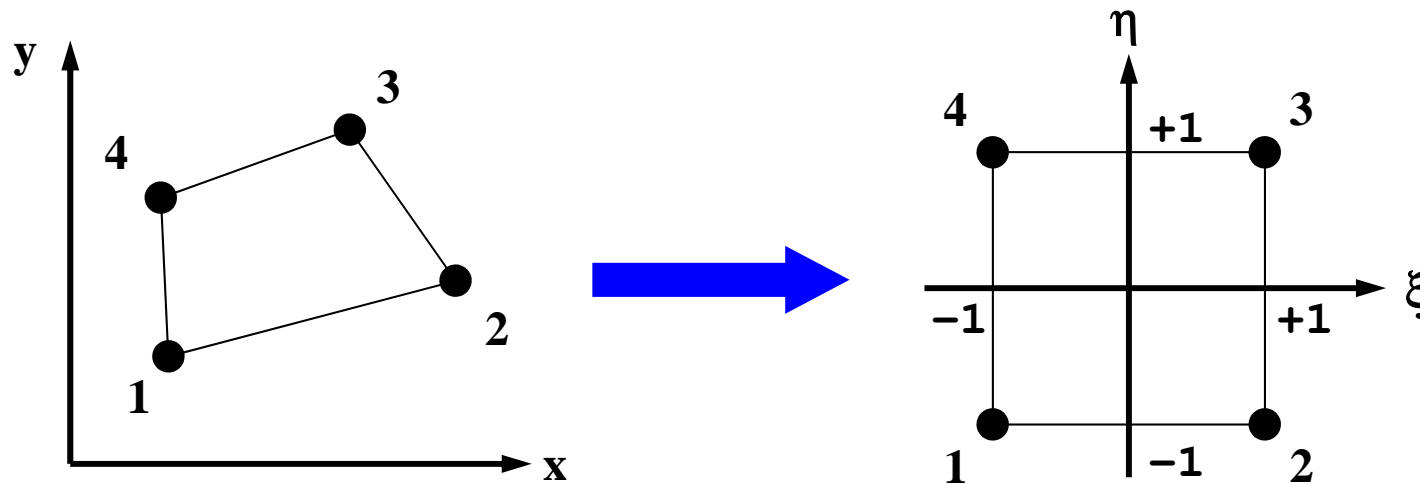
# Isoparametric Element (1/3)

- Each element is mapped to square element  $[\pm 1, \pm 1]$  on natural/local coordinate  $(\xi, \eta)$



- Components of global coordinate system of each node  $(x, y)$  for certain kinds of elements are defined by shape functions  $[N]$  on natural/local coordinate system, where shape functions  $[N]$  are also used for interpolation of dependent variables.

# Isoparametric Element (2/3)

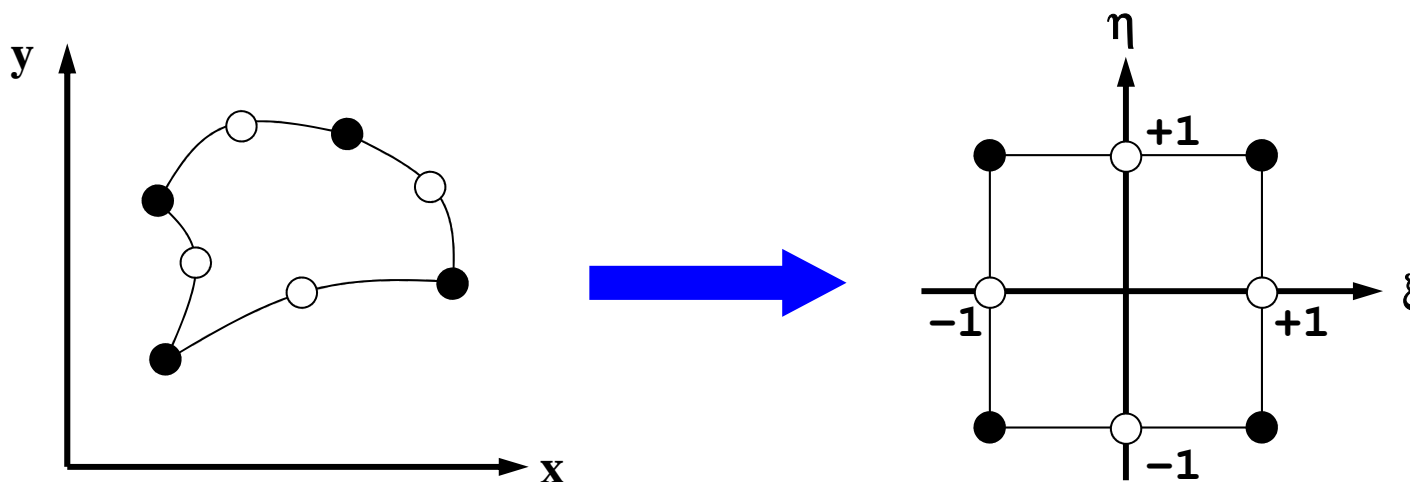


- Coordinate of each node:  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ ,  $(x_4, y_4)$
- Temperature at each node:  $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$

$$T = \sum_{i=1}^4 N_i(\xi, \eta) \cdot T_i$$

$$x = \sum_{i=1}^4 N_i(\xi, \eta) \cdot x_i, \quad y = \sum_{i=1}^4 N_i(\xi, \eta) \cdot y_i$$

# Isoparametric Element (3/3)



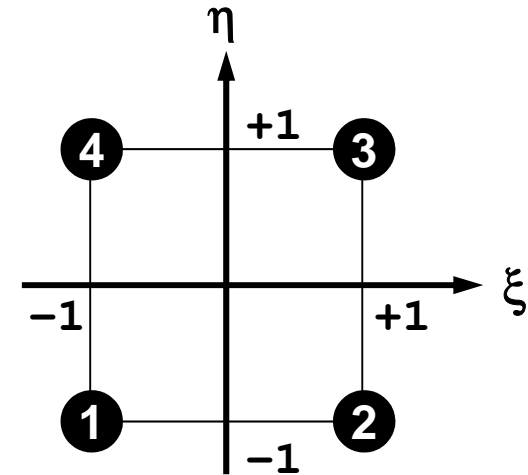
- Higher-order shape function can handle curved lines/surfaces.
- “Natural” coordinate system

Sub-Parametric  
Super-Parametric

# Shape Fn's on 2D Natural Coord. (1/3)

- Polynomial shape functions on squares of natural coordinate:

$$T = \alpha_1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \xi \eta$$



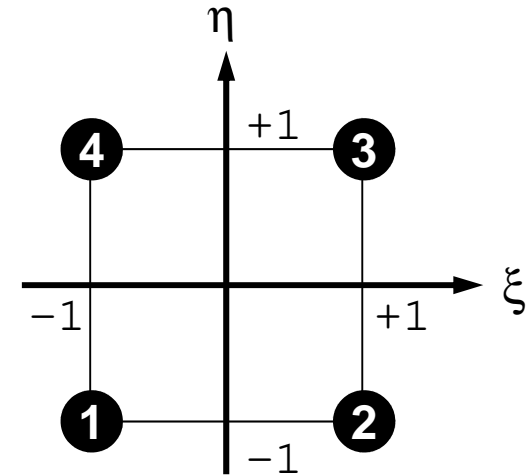
- Coefficients are calculated as follows:

$$\alpha_1 = \frac{T_1 + T_2 + T_3 + T_4}{4}, \quad \alpha_2 = \frac{-T_1 + T_2 + T_3 - T_4}{4},$$

$$\alpha_3 = \frac{-T_1 - T_2 + T_3 + T_4}{4}, \quad \alpha_4 = \frac{T_1 - T_2 + T_3 - T_4}{4}$$

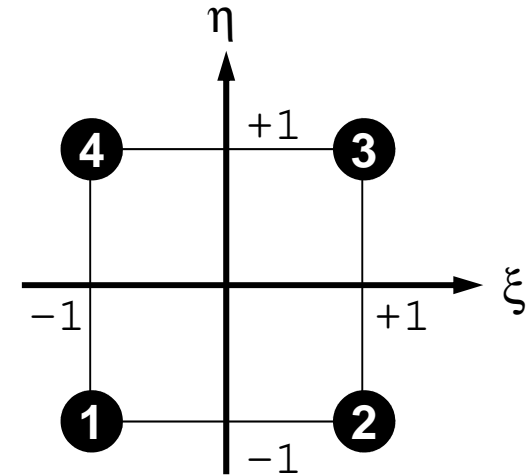
# Shape Fn's on 2D Natural Coord. (2/3)

$$\begin{aligned}
 T &= \alpha_1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \xi \eta \\
 &= \frac{T_1 + T_2 + T_3 + T_4}{4} + \frac{-T_1 + T_2 + T_3 - T_4}{4} \xi + \\
 &\quad \frac{-T_1 - T_2 + T_3 + T_4}{4} \eta + \frac{T_1 - T_2 + T_3 - T_4}{4} \xi \eta \\
 &= \frac{1}{4} (1 - \xi - \eta + \xi \eta) T_1 + \frac{1}{4} (1 + \xi - \eta - \xi \eta) T_2 + \\
 &\quad \frac{1}{4} (1 + \xi + \eta + \xi \eta) T_3 + \frac{1}{4} (1 - \xi + \eta - \xi \eta) T_4 \\
 &= \frac{1}{4} (1 - \xi)(1 - \eta) T_1 + \frac{1}{4} (1 + \xi)(1 - \eta) T_2 + \\
 &\quad \frac{1}{4} (1 + \xi)(1 + \eta) T_3 + \frac{1}{4} (1 - \xi)(1 + \eta) T_4
 \end{aligned}$$



# Shape Fn's on 2D Natural Coord. (2/3)

$$\begin{aligned}
 T &= \alpha_1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \xi \eta \\
 &= \frac{T_1 + T_2 + T_3 + T_4}{4} + \frac{-T_1 + T_2 + T_3 - T_4}{4} \xi + \\
 &\quad \frac{-T_1 - T_2 + T_3 + T_4}{4} \eta + \frac{T_1 - T_2 + T_3 - T_4}{4} \xi \eta \\
 &= \frac{1}{4} (1 - \xi - \eta + \xi \eta) T_1 + \frac{1}{4} (1 + \xi - \eta - \xi \eta) T_2 + \\
 &\quad \frac{1}{4} (1 + \xi + \eta + \xi \eta) T_3 + \frac{1}{4} (1 - \xi + \eta - \xi \eta) T_4 \\
 &= \overset{N_1}{\boxed{\frac{1}{4} (1 - \xi)(1 - \eta)}} T_1 + \overset{N_2}{\boxed{\frac{1}{4} (1 + \xi)(1 - \eta)}} T_2 + \\
 &\quad \overset{N_3}{\boxed{\frac{1}{4} (1 + \xi)(1 + \eta)}} T_3 + \overset{N_4}{\boxed{\frac{1}{4} (1 - \xi)(1 + \eta)}} T_4
 \end{aligned}$$



# Shape Fn's on 2D Natural Coord. (3/3)

- $T$  is defined as follows according to  $T_i$  :

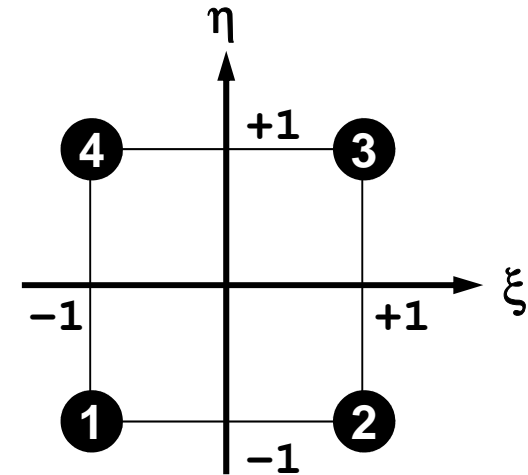
$$T = N_1 T_1 + N_2 T_2 + N_3 T_3 + N_4 T_4$$

- Shape functions  $N_i$  :

$$N_1(\xi, \eta) = \frac{1}{4}(1-\xi)(1-\eta), \quad N_2(\xi, \eta) = \frac{1}{4}(1+\xi)(1-\eta),$$

$$N_3(\xi, \eta) = \frac{1}{4}(1+\xi)(1+\eta), \quad N_4(\xi, \eta) = \frac{1}{4}(1-\xi)(1+\eta)$$

- Also known as “bi-linear” interpolation
- Calculate  $N_i$  at each node



# Extension to 3D Problems

- Tetrahedron/Tetrahedra (四面体) : Triangles in 2D
  - can handle arbitrary shape objects
  - Linear elements are generally less accurate, not practical
  - Higher-order tetrahedral elements are widely used.
- In this class, “tri-linear” hexahedral elements (isoparametric) are used (六面体要素)



# Shape Fn's: 3D Natural/Local Coord.

$$N_1(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta) \quad N_5(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$$

$$N_2(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta) \quad N_6(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$

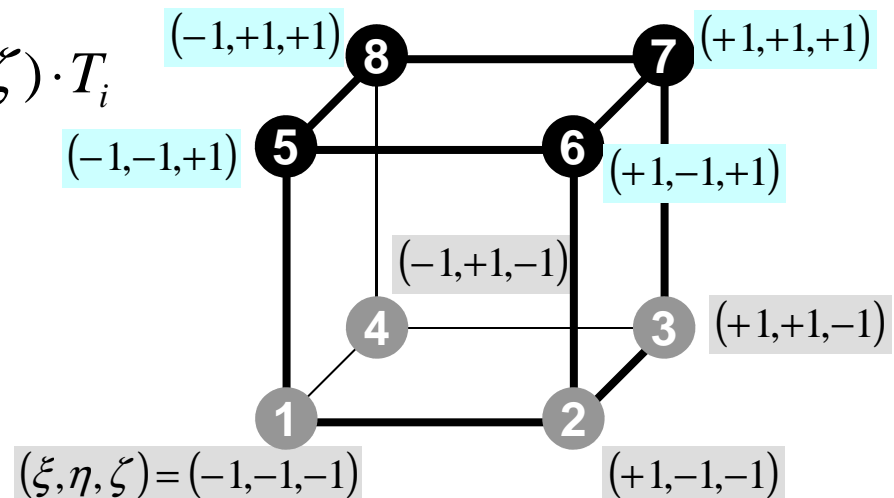
$$N_3(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta) \quad N_7(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$$

$$N_4(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta) \quad N_8(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

$$x = \sum_{i=1}^8 N_i(\xi, \eta, \zeta) \cdot x_i, \quad T = \sum_{i=1}^8 N_i(\xi, \eta, \zeta) \cdot T_i$$

$$y = \sum_{i=1}^8 N_i(\xi, \eta, \zeta) \cdot y_i$$

$$z = \sum_{i=1}^8 N_i(\xi, \eta, \zeta) \cdot z_i$$



- Formulation of 3D Element
- **3D Heat Equations**
  - **Galerkin Method**
  - **Element Matrices**
- Running the Code
- Data Structure
- Overview of the Program

# Galerkin Method (1/3)

- Governing Equation for 3D Steady State Heat Conduction Problems (uniform  $\lambda$ ):

$$\left( \lambda \frac{\partial^2 T}{\partial x^2} \right) + \left( \lambda \frac{\partial^2 T}{\partial y^2} \right) + \left( \lambda \frac{\partial^2 T}{\partial z^2} \right) + \dot{Q} = 0$$

$$T = [N]\{\phi\} \quad \text{Distribution of temperature in each element (matrix form), } \phi: \text{ Temperature at each node}$$

- Following integral equation is obtained at each element by Galerkin method, where  $[N]$ 's are also weighting functions:

$$\int_V [N]^T \left\{ \lambda \left( \frac{\partial^2 T}{\partial x^2} \right) + \lambda \left( \frac{\partial^2 T}{\partial y^2} \right) + \lambda \left( \frac{\partial^2 T}{\partial z^2} \right) + \dot{Q} \right\} dV = 0$$

# Galerkin Method (2/3)

- Green's Theorem (3D)

$$\int_V A \left( \frac{\partial^2 B}{\partial x^2} + \frac{\partial^2 B}{\partial y^2} + \frac{\partial^2 B}{\partial z^2} \right) dV = \int_S A \frac{\partial B}{\partial n} dS - \int_V \left( \frac{\partial A}{\partial x} \frac{\partial B}{\partial x} + \frac{\partial A}{\partial y} \frac{\partial B}{\partial y} + \frac{\partial A}{\partial z} \frac{\partial B}{\partial z} \right) dV$$

- Apply this to the 1st 3-parts of the equation with 2<sup>nd</sup>-order diff. (surface integration terms are ignored):

$$\begin{aligned} & \int_V [N]^T \{ \lambda(T_{,xx}) + \lambda(T_{,yy}) + \lambda(T_{,zz}) \} dV \\ & = - \int_V \{ \lambda([N_{,x}]^T T_{,x}) + \lambda([N_{,y}]^T T_{,y}) + \lambda([N_{,z}]^T T_{,z}) \} dV \end{aligned}$$

- Consider the following terms:

$$T = [N]\{\phi\}, \quad T_{,x} = [N_{,x}]\{\phi\}, \quad T_{,y} = [N_{,y}]\{\phi\}, \quad T_{,z} = [N_{,z}]\{\phi\}$$

## Galerkin Method (3/3)

- Finally, following equation is obtained by considering heat generation term  $\dot{Q}$  :

$$-\int_V \left\{ \lambda \left( [N_{,x}]^T [N_{,x}] \right) + \lambda \left( [N_{,y}]^T [N_{,y}] \right) + \lambda \left( [N_{,z}]^T [N_{,z}] \right) \right\} dV \cdot \{\phi\} + \int_V \dot{Q} [N] dV = 0$$

- This is called “weak form (弱形式)”. Original PDE consists of terms with 2<sup>nd</sup>-order diff., but this “weak form” only includes 1<sup>st</sup>-order diff by Green’s theorem.
  - Requirements for shape functions are “weaker” in “weak form”. Linear functions can describe effects of 2<sup>nd</sup>-order differentiation.
  - Same as 1D problem

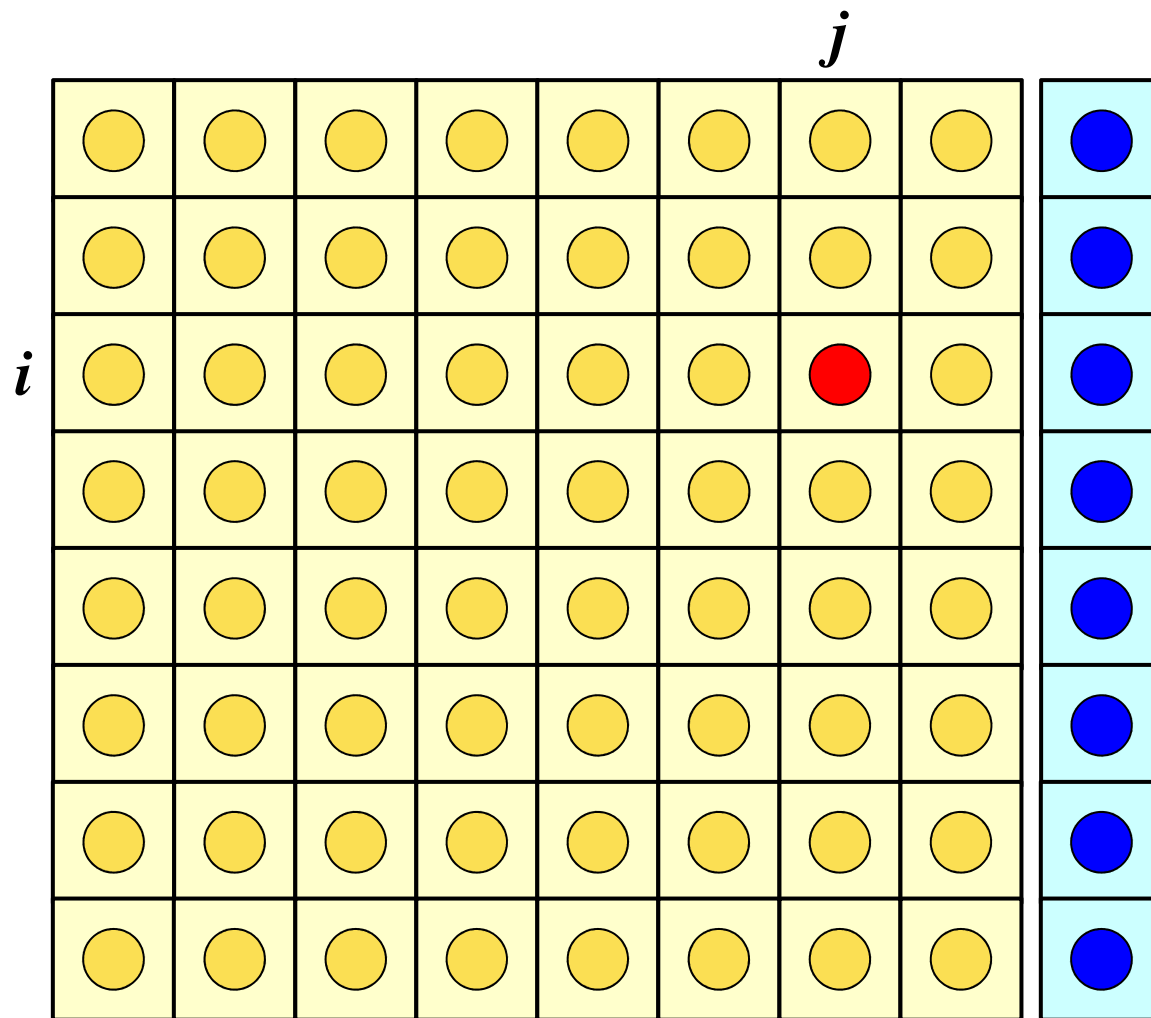
Weak Form with B.C.: on each elem.

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)}$$

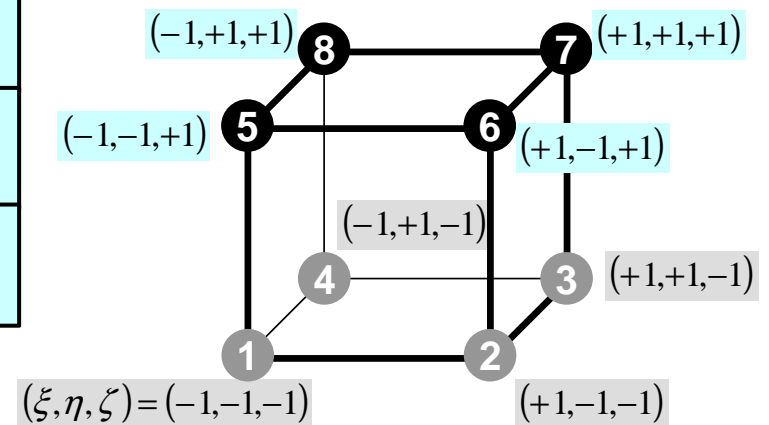
$$[k]^{(e)} = \int_V \lambda ([N_{,x}]^T [N_{,x}]) dV + \int_V \lambda ([N_{,y}]^T [N_{,y}]) dV \\ + \int_V \lambda ([N_{,z}]^T [N_{,z}]) dV$$

$$\{f\}^{(e)} = \int_V \dot{Q} [N]^T dV$$

# Element Matrix: 8x8

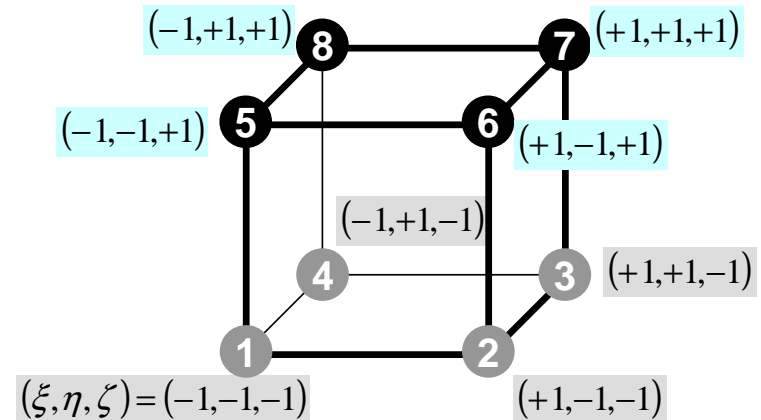


$$[k_{ij}] \quad (i, j = 1 \dots 8)$$



# Element Matrix: $k_{ij}$

$$[k_{ij}] \quad (i, j = 1 \dots 8)$$



$$[k]^{(e)} = \int_V \lambda ([N_{,x}]^T [N_{,x}]) dV + \int_V \lambda ([N_{,y}]^T [N_{,y}]) dV + \int_V \lambda ([N_{,z}]^T [N_{,z}]) dV$$



$$k_{ij} = - \int_V \{ \lambda \cdot N_{i,x} \cdot N_{j,x} + \lambda \cdot N_{i,y} \cdot N_{j,y} + \lambda \cdot N_{i,z} \cdot N_{j,z} \} dV$$

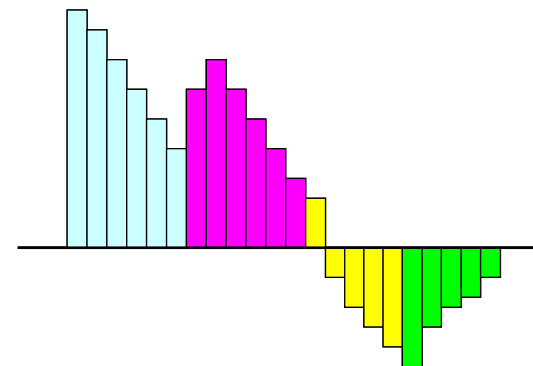


# Next Stage: Integration

# Methods for Numerical Integration

- Trapezoidal Rule
- Simpson's Rule
- Gaussian Quadrature (or Gauss-Legendre)
  - accurate
  
- Values of functions at finite numbers of sample points are utilized:

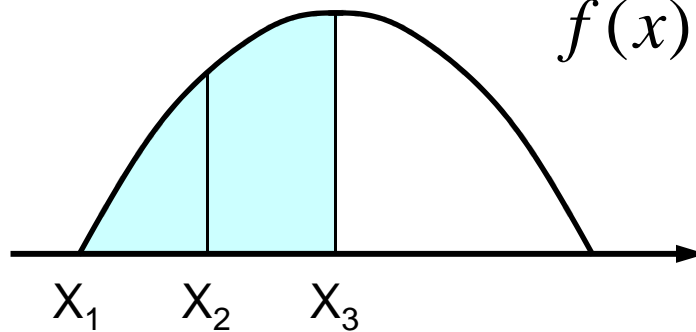
$$\int_{x_1}^{x_2} f(x) dx \Rightarrow \sum_{k=1}^m [w_k \cdot f(x_k)]$$



# Gaussian Quadrature in 1D

more accurate than Simpson's rule

**Simpson's**

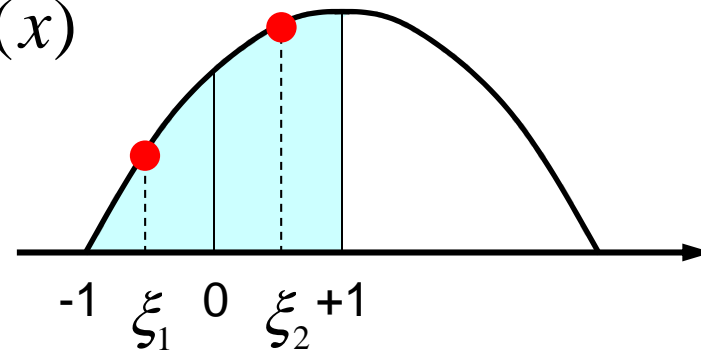


$$X_1 = 0, \quad X_2 = \frac{\pi}{4}, \quad X_3 = \frac{\pi}{2}$$

$$h = X_2 - X_1 = X_3 - X_2 = \frac{\pi}{4}$$

$$S = \frac{h}{3} [f(X_1) + 4f(X_2) + f(X_3)] = 1.0023$$

**Gauss**



$$\xi_1, \xi_2 = \pm 0.5773502692$$

$$S = \int_0^{\pi/2} f(x) dx = \int_{-1}^{+1} f(\xi) h d\xi$$

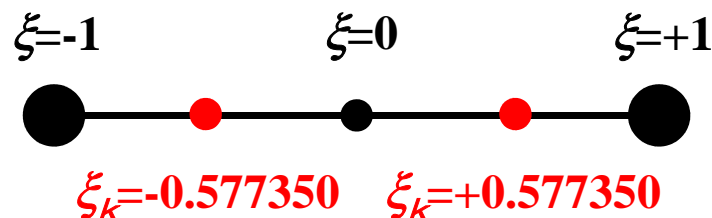
$$\cong h \sum_{k=1}^2 W_k \cdot f(\xi_k) = 0.99847$$

# Gaussian Quadrature

## ガウスの積分公式

- On normalized “natural (or local)” coordinate system  $[-1,+1]$  (自然座標系, 局所座標系)
- Can approximate up to  $(2m-1)$ -th order of functions by  $m$  quadrature points ( $m=2$  is enough for quadratic shape functions).

$$\int_{-1}^{+1} f(\xi) d\xi = \sum_{k=1}^m [w_k \cdot f(\xi_k)]$$



$$m = 1 \quad \xi_k = 0.00, w_k = 2.00$$

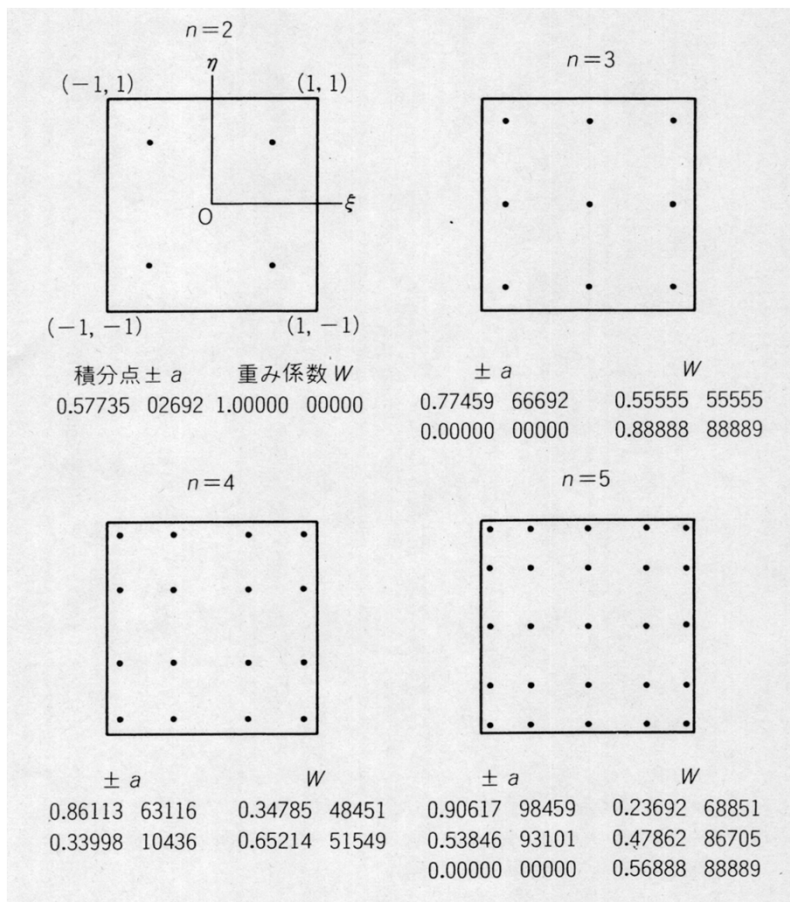
$$m = 2 \quad \xi_k = \pm 0.577350, w_k = 1.00$$

$$m = 3 \quad \xi_k = 0.00, w_k = 8/9$$

$$\xi_k = \pm 0.774597, w_k = 5/9$$

# Gaussian Quadrature

can be easily extended to 2D & 3D



$$I = \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta) d\xi d\eta$$

$$= \sum_{i=1}^m \sum_{j=1}^n [W_i \cdot W_j \cdot f(\xi_i, \eta_j)]$$

$m, n$ : number of quadrature points in  $\xi, \eta$ -direction

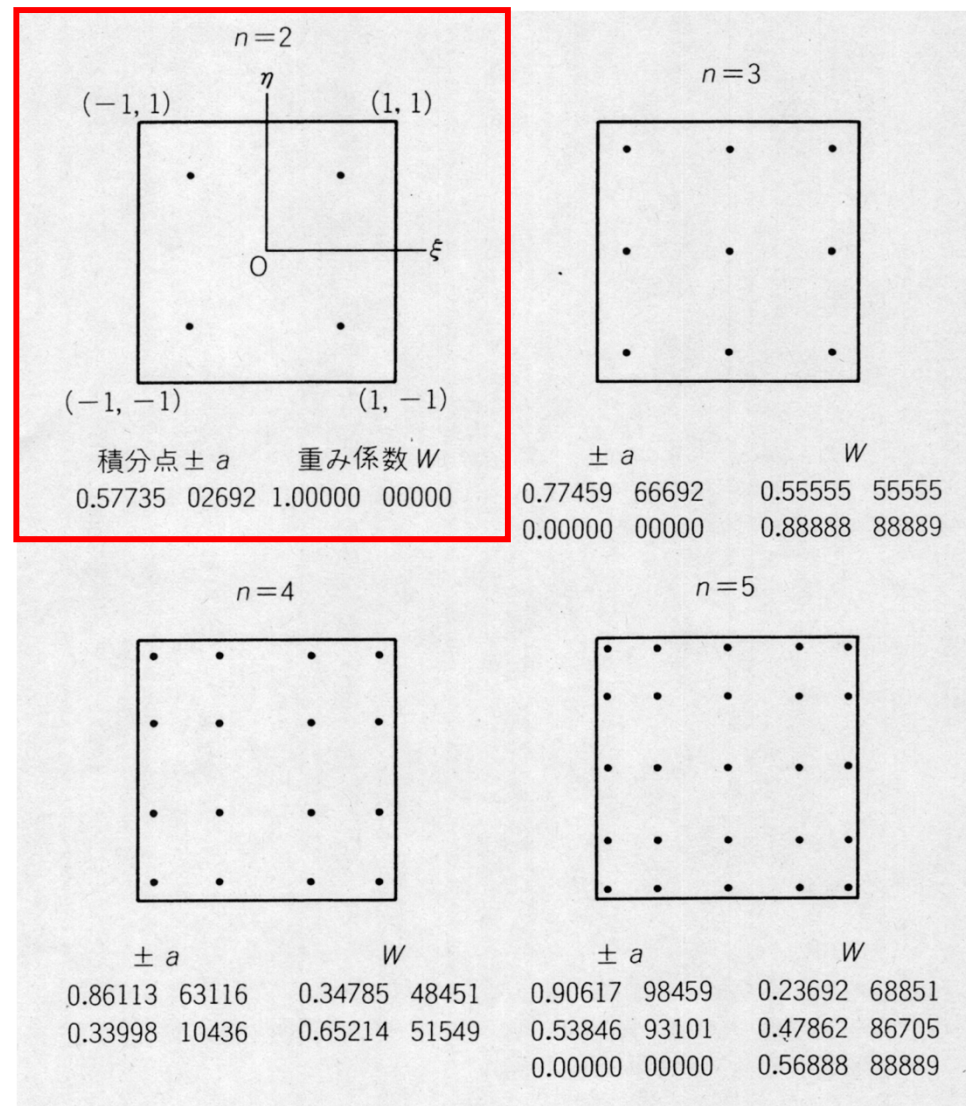
$(\xi_i, \eta_j)$ : Coordinates of Quad's

$W_i, W_j$ : Weighting Factor

# Gaussian Quadrature

## ガウスの積分公式

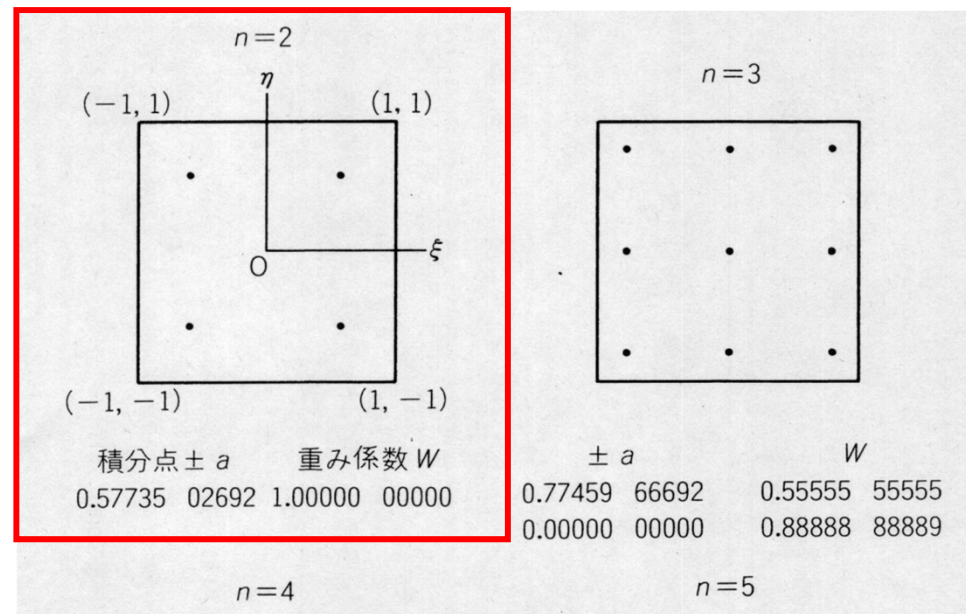
This configuration is widely used. In 2D problem, integration is done using values of “f” at 4 quad. points.



# Gaussian Quadrature

## ガウスの積分公式

This configuration is widely used. In 2D problem, integration is done using values of “f” at 4 quad. points.



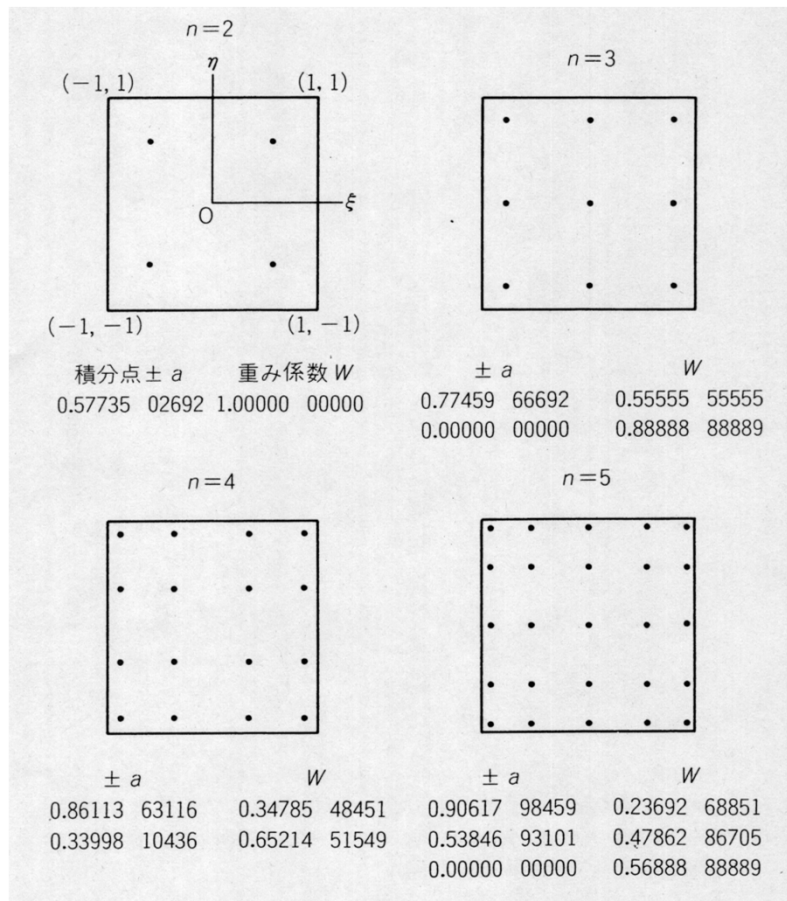
$$I = \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta) d\xi d\eta = \sum_{i=1}^m \sum_{j=1}^n [W_i \cdot W_j \cdot f(\xi_i, \eta_j)]$$

$$= 1.0 \times 1.0 \times f(-0.57735, -0.57735) + 1.0 \times 1.0 \times f(-0.57735, +0.57735) \\ + 1.0 \times 1.0 \times f(+0.57735, +0.57735) + 1.0 \times 1.0 \times f(+0.57735, -0.57735)$$

± a		W	
0.33998	0.10436	0.65214	0.51549
0.53846	0.93101	0.47802	0.60703
0.00000	0.00000	0.56888	0.88889

# Next Stage: Integration

- 3D Natural/Local Coordinate  $(\xi, \eta, \zeta)$ :
  - Gaussian Quadrature



$$I = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta, \zeta) d\xi d\eta d\zeta$$

$$= \sum_{i=1}^L \sum_{j=1}^M \sum_{k=1}^N [W_i \cdot W_j \cdot W_k \cdot f(\xi_i, \eta_j, \zeta_k)]$$

$L, M, N$ : number of quadrature points in  $\xi, \eta, \zeta$ -direction

$(\xi_i, \eta_j, \zeta_k)$  : Coordinates of Quad's

$W_i, W_j, W_k$  : Weighting Factor



# Partial Diff. on Natural Coord. (1/4)

- According to formulae:

$$\frac{\partial N_i(\xi, \eta, \zeta)}{\partial \xi} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \xi} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \xi}$$

$$\frac{\partial N_i(\xi, \eta, \zeta)}{\partial \eta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \eta} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \eta}$$

$$\frac{\partial N_i(\xi, \eta, \zeta)}{\partial \zeta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \zeta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \zeta} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \zeta}$$

$\left[ \frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} \right]$  can be easily derived according to definitions.

$\left[ \frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} \right]$  are required for computations.

# Partial Diff. on Natural Coord. (2/4)

- In matrix form:

$$\left\{ \begin{array}{c} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{array} \right\} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \left\{ \begin{array}{c} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{array} \right\} = [J] \left\{ \begin{array}{c} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{array} \right\}$$

$[J]$  : Jacobi matrix, Jacobian

# Partial Diff. on Natural Coord. (3/4)

- Components of Jacobian:

$$J_{11} = \frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^8 N_i x_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \xi} x_i, \quad J_{12} = \frac{\partial y}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^8 N_i y_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \xi} y_i,$$

$$J_{13} = \frac{\partial z}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^8 N_i z_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \xi} z_i$$

$$J_{21} = \frac{\partial x}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^8 N_i x_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \eta} x_i, \quad J_{22} = \frac{\partial y}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^8 N_i y_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \eta} y_i,$$

$$J_{23} = \frac{\partial z}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^8 N_i z_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \eta} z_i$$

$$J_{31} = \frac{\partial x}{\partial \zeta} = \frac{\partial}{\partial \zeta} \left( \sum_{i=1}^8 N_i x_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \zeta} x_i, \quad J_{32} = \frac{\partial y}{\partial \zeta} = \frac{\partial}{\partial \zeta} \left( \sum_{i=1}^8 N_i y_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \zeta} y_i,$$

$$J_{33} = \frac{\partial z}{\partial \zeta} = \frac{\partial}{\partial \zeta} \left( \sum_{i=1}^8 N_i z_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \zeta} z_i$$

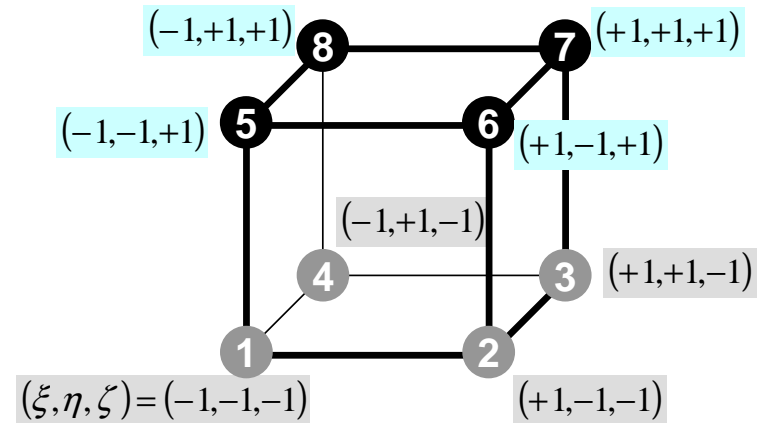
# Partial Diff. on Natural Coord. (4/4)

- Partial differentiation on global coordinate system is introduced as follows (with inverse of Jacobian matrix ( $3 \times 3$ ))

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix}$$

# Integration on Element

$$[k_{ij}] \quad (i, j = 1 \dots 8)$$

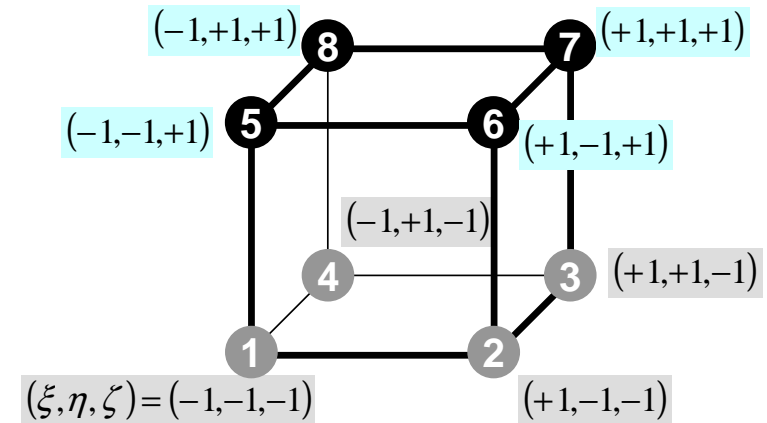


$$k_{ij} = - \int_V \left\{ \lambda \cdot N_{i,x} \cdot N_{j,x} + \lambda \cdot N_{i,y} \cdot N_{j,y} + \lambda \cdot N_{i,z} \cdot N_{j,z} \right\} dV$$

$$= - \int_V \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} dV$$

# Integration on Natural Coord.

$$[k_{ij}] \quad (i, j = 1 \dots 8)$$



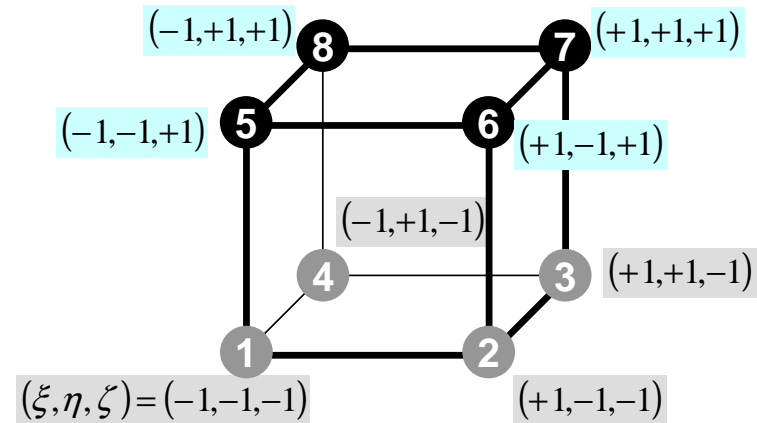
$$-\int_V \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} dV =$$

$$-\iiint \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} dx dy dz =$$

$$-\int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det |J| d\xi d\eta d\zeta$$

# Gaussian Quadrature

$$[k_{ij}] \quad (i, j = 1 \dots 8)$$



$$- \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det |J| d\xi d\eta d\zeta$$

$$I = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta, \zeta) d\xi d\eta d\zeta$$

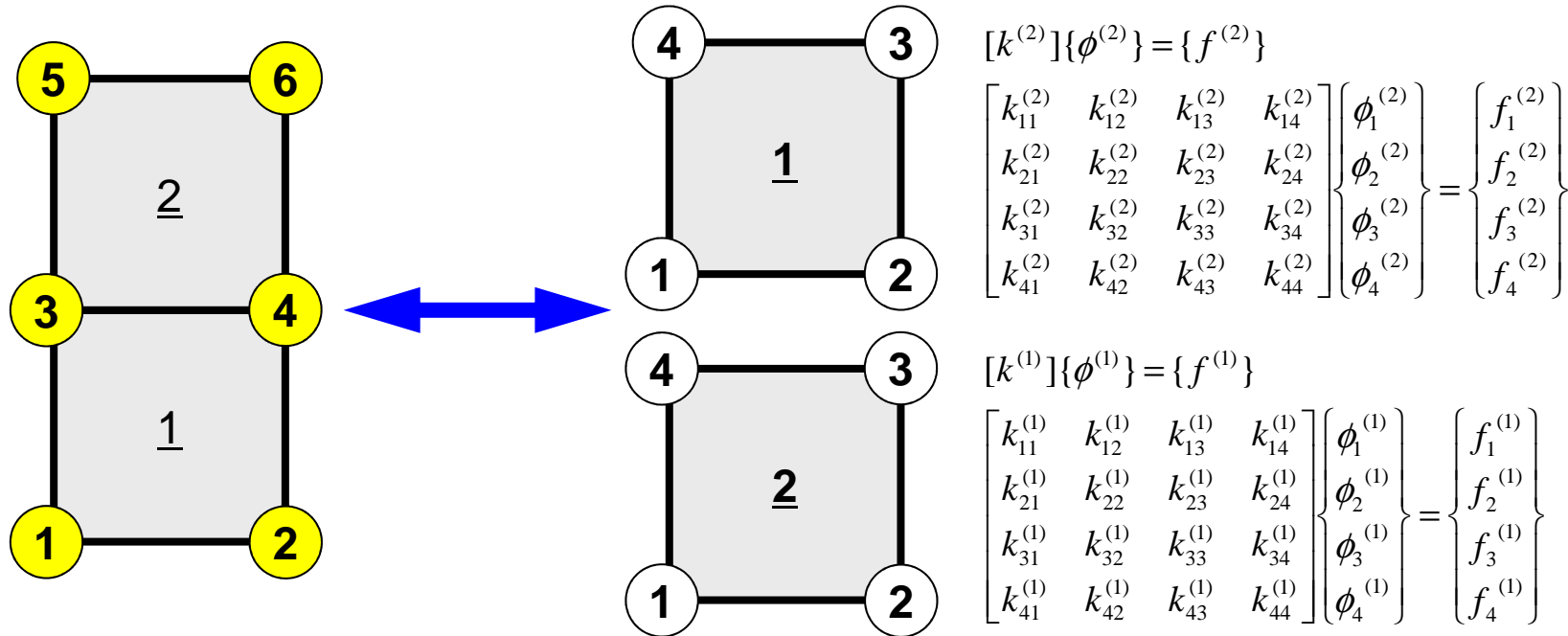
$$= \sum_{i=1}^L \sum_{j=1}^M \sum_{k=1}^N [w_i \cdot w_j \cdot w_k \cdot f(\xi_i, \eta_j, \zeta_k)]$$

# Remaining Procedures

- Element matrices have been formed.
- Accumulation to Global Matrix
- Implementation of Boundary Conditions
- Solving Linear Equations
- Details of implementation will be discussed in classes later than next week through explanation of programs



# Accumulation: Local -> Global Matrices



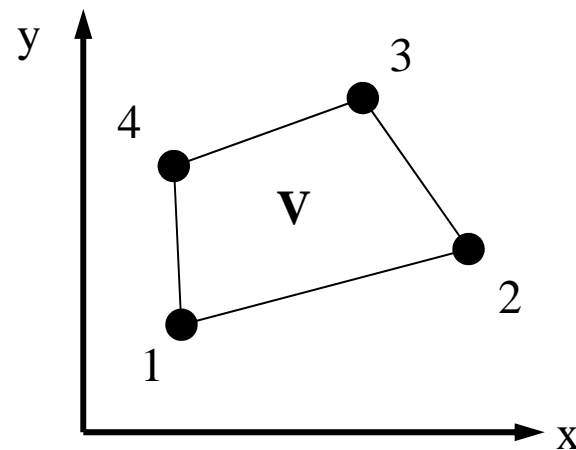
$$[K]\{\Phi\} = \{F\}$$

$$\begin{bmatrix} D_1 & X & X & X & & & \\ X & D_2 & X & X & & & \\ X & X & D_3 & X & X & X & \\ X & X & X & D_4 & X & X & \\ & & X & X & D_5 & X & \\ & & X & X & X & D_6 & \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \end{Bmatrix} = \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \end{Bmatrix}$$

- Formulation of 3D Element
- 3D Heat Equations
  - Galerkin Method
  - Element Matrices
  - **Exercise**
- Running the Code
- Data Structure
- Overview of the Program

# Exercise

- Develop a program and calculate area of the following quadrilateral using Gaussian Quadrature.



1: (1.0, 1.0)  
2: (4.0, 2.0)  
3: (3.0, 5.0)  
4: (2.0, 4.0)

$$I = \int_V dV = \int_{-1}^{+1} \int_{-1}^{+1} \det|J| d\xi d\zeta$$

# Tips (1/2)

- Calculate Jacobian
- Apply Gaussian Quadrature (n=2)

$$I = \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta) d\xi d\eta = \sum_{i=1}^m \sum_{j=1}^n [W_i \cdot W_j \cdot f(\xi_i, \eta_j)]$$

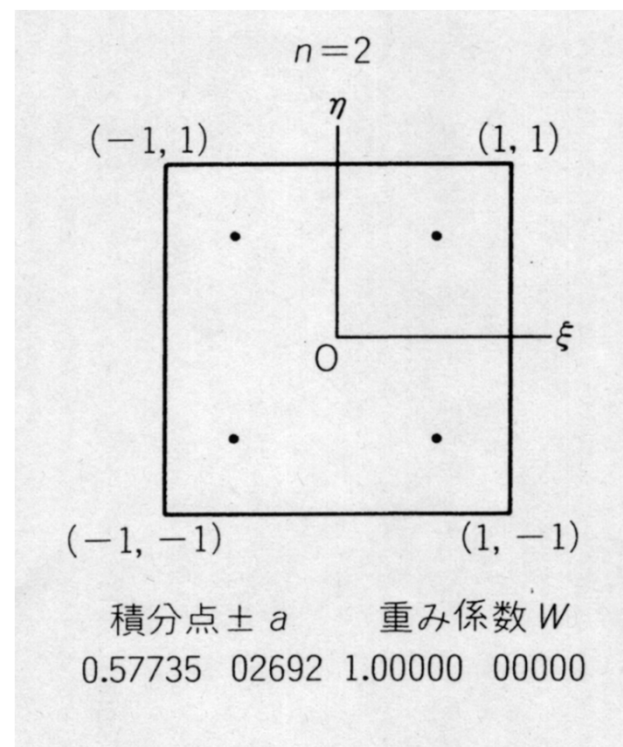
```

implicit REAL*8 (A-H,O-Z)
real*8 W(2)
real*8 POI(2)

W(1)= 1.0d0
W(2)= 1.0d0
POI(1)= -0.5773502692d0
POI(2)= +0.5773502692d0

SUM= 0.d0
do jp= 1, 2
do ip= 1, 2
FC = F(POI(ip), POI(jp))
SUM= SUM + W(ip)*W(jp)*FC
enddo
enddo

```



## Tips (2/2)

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}, \quad \det|J| = \frac{\partial x}{\partial \xi} \cdot \frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi} \cdot \frac{\partial x}{\partial \eta}$$

$$\frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^4 N_i x_i \right) = \sum_{i=1}^4 \frac{\partial N_i}{\partial \xi} x_i, \quad \frac{\partial y}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^4 N_i y_i \right) = \sum_{i=1}^4 \frac{\partial N_i}{\partial \xi} y_i,$$

$$\frac{\partial x}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^4 N_i x_i \right) = \sum_{i=1}^4 \frac{\partial N_i}{\partial \eta} x_i, \quad \frac{\partial y}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^4 N_i y_i \right) = \sum_{i=1}^4 \frac{\partial N_i}{\partial \eta} y_i$$

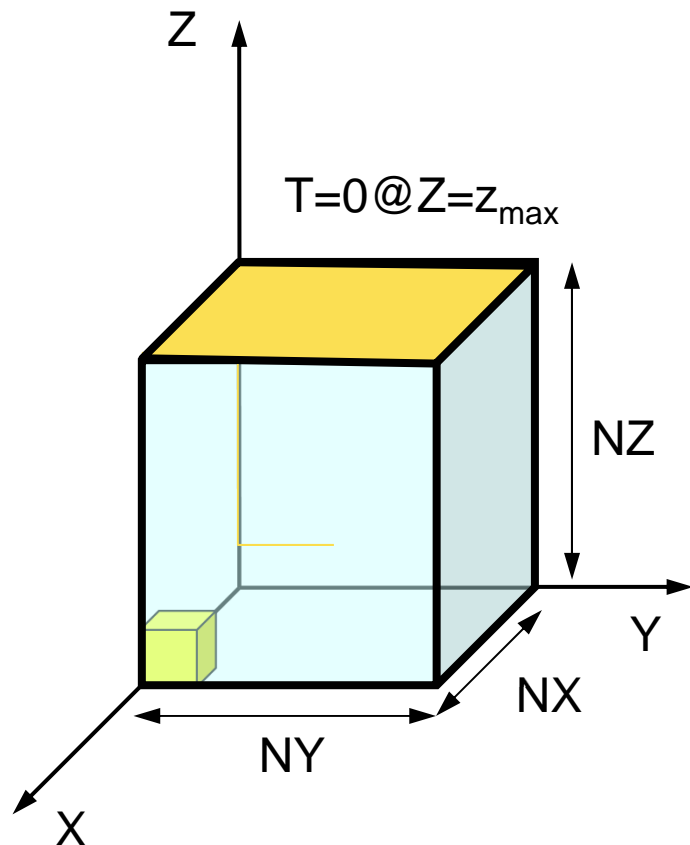
$$N_1(\xi, \eta) = \frac{1}{4} (1 - \xi)(1 - \eta), \quad N_2(\xi, \eta) = \frac{1}{4} (1 + \xi)(1 - \eta),$$

$$N_3(\xi, \eta) = \frac{1}{4} (1 + \xi)(1 + \eta), \quad N_4(\xi, \eta) = \frac{1}{4} (1 - \xi)(1 + \eta)$$

- Formulation of 3D Element
- 3D Heat Equations
  - Galerkin Method
  - Element Matrices
- **Running the Code**
- Data Structure
- Overview of the Program

# 3D Steady-State Heat Conduction

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$



- Heat Generation
- Uniform thermal conductivity  $\lambda$
- HEX meshes
  - 1x1x1 cubes
  - NX, NY, NZ cubes in each direction
- Boundary Conditions
  - $T=0@Z=z_{\max}$
- Heat Gen. Rate is a function of location (cell center:  $x_c, y_c$ )
  - $\dot{Q}(x, y, z) = QVOL|x_c + y_c|$

# Copy/Installation

## Install

```
>$ cd <$E-TOP>/fem-c/fem3d/src  
>$ make  
>$ ls ../run/sol  
sol
```

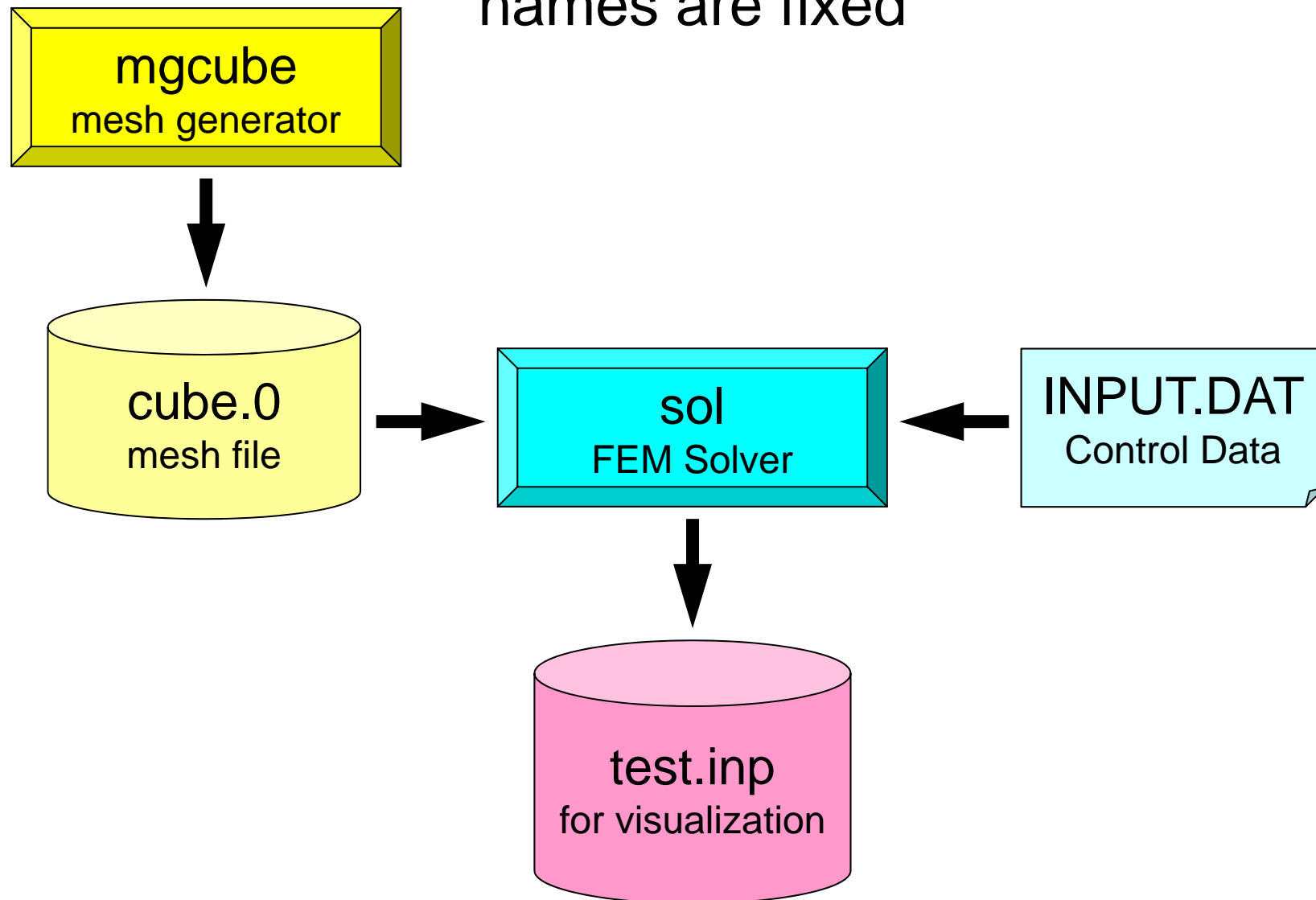
## Install of Mesh Generator

```
>$ cd <$E-TOP>/fem-c/fem3d/run  
>$ gcc -O3 mgcube.c -o mgcube
```

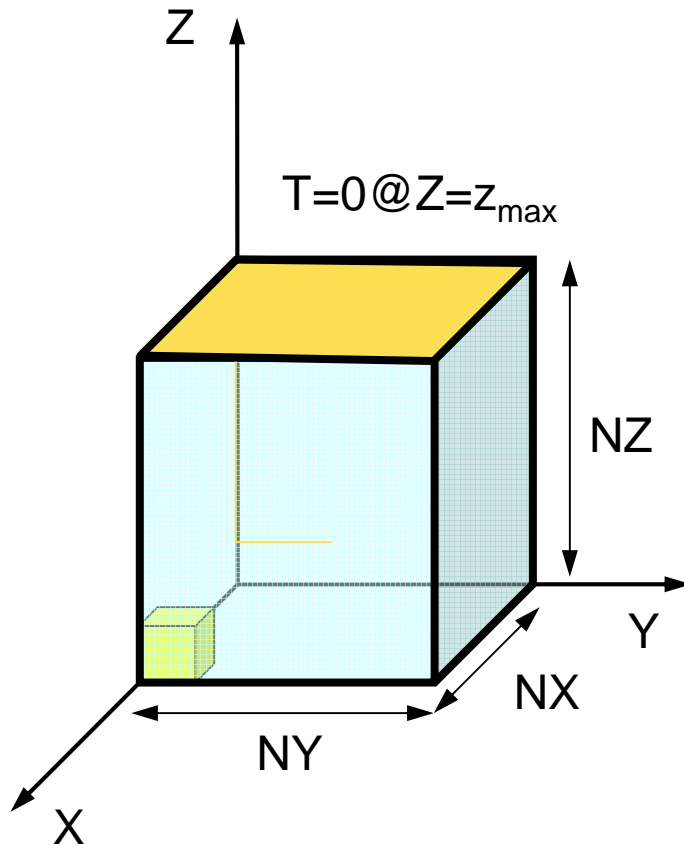


# Operations

Starting from Grid Generation to Computation, File-  
names are fixed



# Mesh Generation



```
>$ cd <$E-TOP>/fem-c/fem3d/run  
>$ ./mgcube
```

$NX, NY, NZ$

← **Number of  
Elem's in each  
direction**

$20, 20, 20$

← **example**

```
>$ ls cube.0
```

**confirmation**

```
cube.0
```

# Control File: INPUT.DAT

## INPUT.DAT

```

cube.0      fname
2000        ITER
1.0 1.0     COND, QVOL
1.0e-08     RESID

```

- fname : Name of Mesh File
- ITER : Max. Iterations for CG
- COND : Thermal Conductivity
- QVOL : Heat Generation Rate
- RESID : Criteria for Convergence of CG

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

# Running

```
>$ cd <$E-TOP>/fem-c/fem3d/run  
>$ ./sol
```

```
>$ ls test.inp          Confirmation  
    test.inp
```

# ParaView

- <http://www.paraview.org/>
- Opening files
- Displaying figures
- Saving image files
  - <http://nkl.cc.u-tokyo.ac.jp/class/HowtouseParaViewE.pdf>
  - <http://nkl.cc.u-tokyo.ac.jp/class/HowtouseParaViewJ.pdf>

# UCD Format (1/3)

## Unstructured Cell Data

要素の種類

点

線

三角形

四角形

四面体

角錐

三角柱

六面体

二次要素

線2

三角形2

四角形2

四面体2

角錐2

三角柱2

六面体2

キーワード

pt

line

tri

quad

tet

pyr

prism

hex

line2

tri2

quad2

tet2

pyr2

prism2

hex2

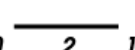
点



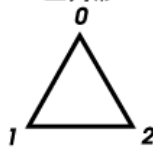
線



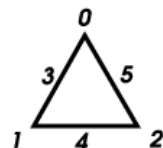
線2



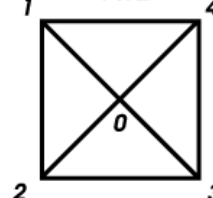
三角形



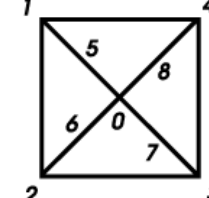
三角形2



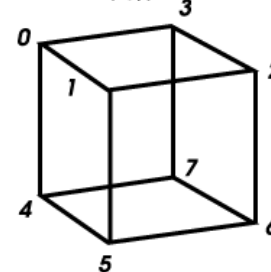
四角錐



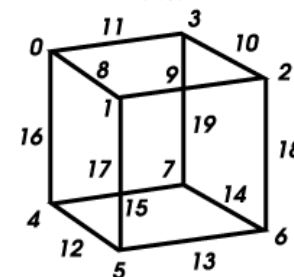
四角錐2



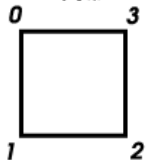
六面体



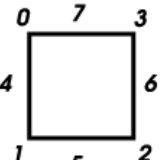
六面体2



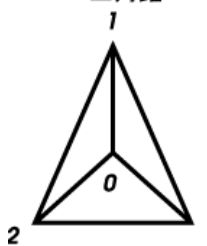
四角形



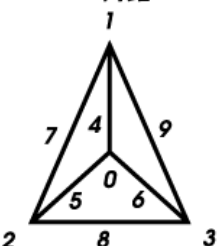
四角形2



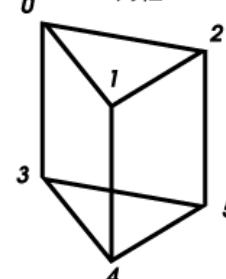
三角錐



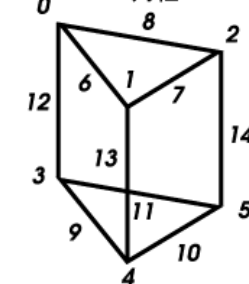
三角錐2



三角柱



三角柱2



## UCD Format (2/3)

- Originally for AVS, microAVS
- Extension of the UCD file is “inp”
- There are two types of formats. Only old type can be read by ParaView.

# UCD Format (3/3): Old Format

(全節点数) (全要素数) (各節点のデータ数) (各要素のデータ数) (モデルのデータ数)

(節点番号1) (X座標) (Y座標) (Z座標)

(節点番号2) (X座標) (Y座標) (Z座標)

⋮

(要素番号1) (材料番号) (要素の種類) (要素を構成する節点のつながり)

(要素番号2) (材料番号) (要素の種類) (要素を構成する節点のつながり)

⋮

(節点のデータ成分数) (成分1の構成数) (成分2の構成数) ⋯ (各成分の構成数)

(節点データ成分1のラベル), (単位)

(節点データ成分2のラベル), (単位)

⋮

(各節点データ成分のラベル), (単位)

(節点番号1) (節点データ1) (節点データ2) ⋯

(節点番号2) (節点データ1) (節点データ2) ⋯

⋮

(要素のデータ成分数) (成分1の構成数) (成分2の構成数) ⋯ (各成分の構成数)

(要素データ成分1のラベル), (単位)

(要素データ成分2のラベル), (単位)

⋮

(各要素データ成分のラベル), (単位)

(要素番号1) (要素データ1) (要素データ2) ⋯

(要素番号2) (要素データ1) (要素データ2) ⋯

⋮



- Formulation of 3D Element
- 3D Heat Equations
  - Galerkin Method
  - Element Matrices
- Running the Code
- **Data Structure**
- Overview of the Program

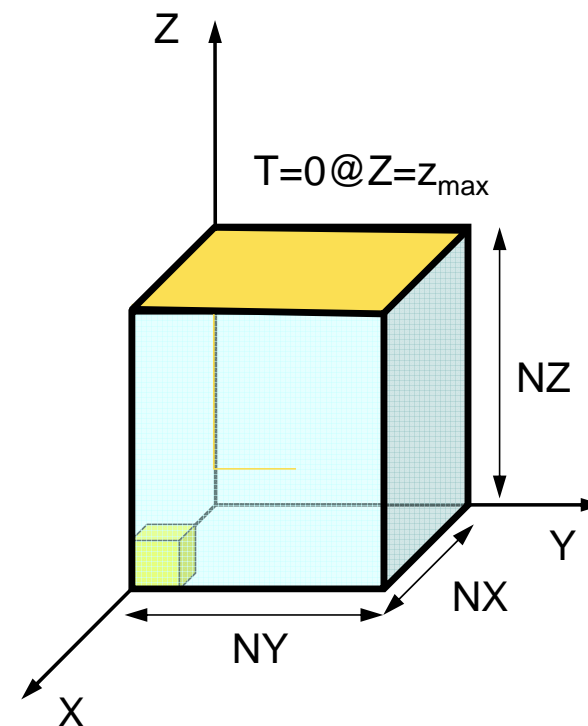
# Overview of Mesh File: cube.0

numbering starts from “1”

- Nodes
  - Node # (How many nodes ?)
  - Node ID, Coordinates
- Elements
  - Element #
  - Element Type
  - Element ID, Material ID, Connectivity
- Node Groups
  - Group #
  - Node # in each group
  - Group Name
  - Nodes in each group

# Example of “cube.0” (NX=NY=NZ=4) Node

125				=5*5*5 (Node #)
1	0.00	0.00	0.00	
2	1.00	0.00	0.00	
3	2.00	0.00	0.00	
4	3.00	0.00	0.00	
5	4.00	0.00	0.00	
6	0.00	1.00	0.00	
7	1.00	1.00	0.00	
8	2.00	1.00	0.00	
9	3.00	1.00	0.00	
...				
121	0.00	4.00	4.00	
122	1.00	4.00	4.00	
123	2.00	4.00	4.00	
124	3.00	4.00	4.00	
125	4.00	4.00	4.00	
Node ID	X-coord.	Y	Z	



movie

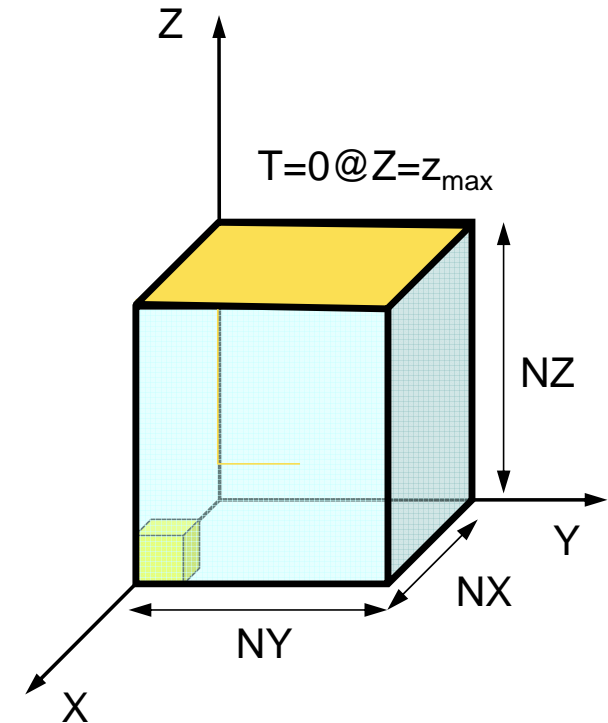
# Example of “cube.0” (NX=NY=NZ=4) Element (1/2)

```

64                                     =4*4*4 (Element #)
361 361 361 361 361 361 361 361 361 361
361 361 361 361 361 361 361 361 361 361
361 361 361 361 361 361 361 361 361 361
361 361 361 361 361 361 361 361 361 361
361 361 361 361 361 361 361 361 361 361
361 361 361 361 361 361 361 361 361 361
361 361 361 361

```

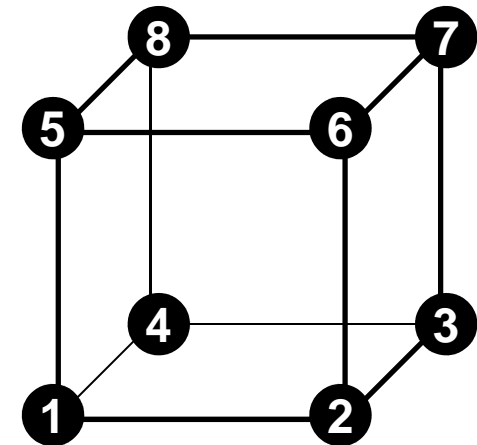
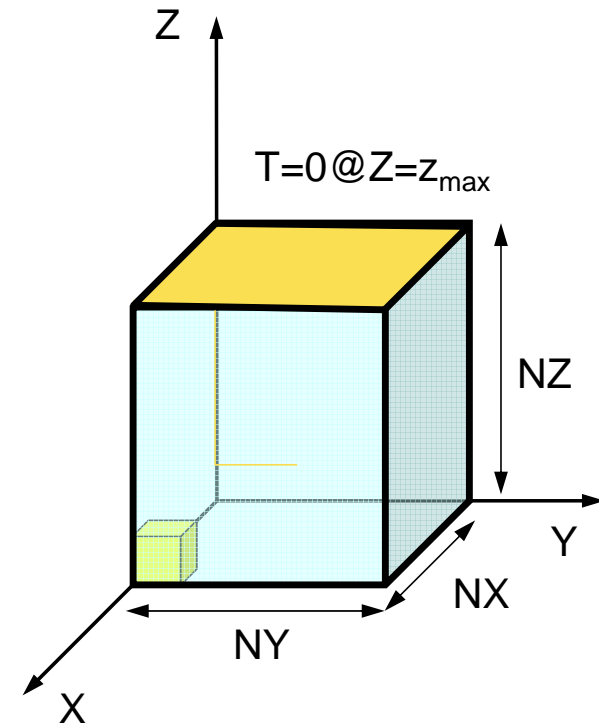
Element Type: 361  
3D, Hexahedron, Linear (1<sup>st</sup> order)



# Example of “cube.0” Element (2/2)

1	1	1	2	7	6	26	27	32	31
2	1	2	3	8	7	27	28	33	32
3	1	3	4	9	8	28	29	34	33
4	1	4	5	10	9	29	30	35	34
5	1	6	7	12	11	31	32	37	36
6	1	7	8	13	12	32	33	38	37
7	1	8	9	14	13	33	34	39	38
8	1	9	10	15	14	34	35	40	39
9	1	11	12	17	16	36	37	42	41
10	1	12	13	18	17	37	38	43	42
11	1	13	14	19	18	38	39	44	43
12	1	14	15	20	19	39	40	45	44
13	1	16	17	22	21	41	42	47	46
...									
53	1	81	82	87	86	106	107	112	111
54	1	82	83	88	87	107	108	113	112
55	1	83	84	89	88	108	109	114	113
56	1	84	85	90	89	109	110	115	114
57	1	86	87	92	91	111	112	117	116
58	1	87	88	93	92	112	113	118	117
59	1	88	89	94	93	113	114	119	118
60	1	89	90	95	94	114	115	120	119
61	1	91	92	97	96	116	117	122	121
62	1	92	93	98	97	117	118	123	122
63	1	93	94	99	98	118	119	124	123
64	1	94	95	100	99	119	120	125	124

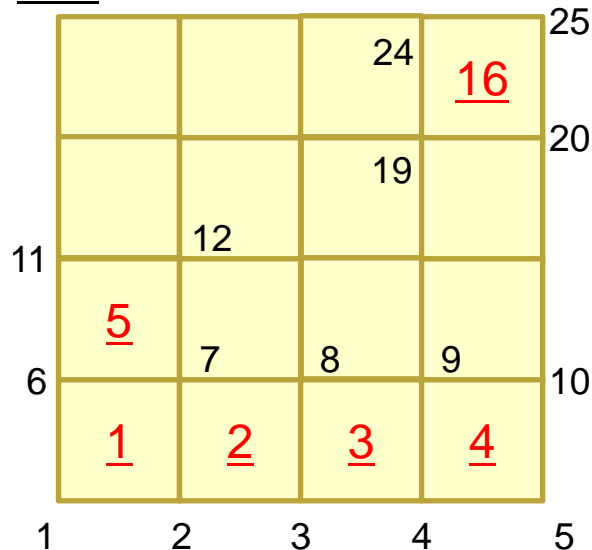
Elem ID    MAT-ID                    ID of 8 nodes



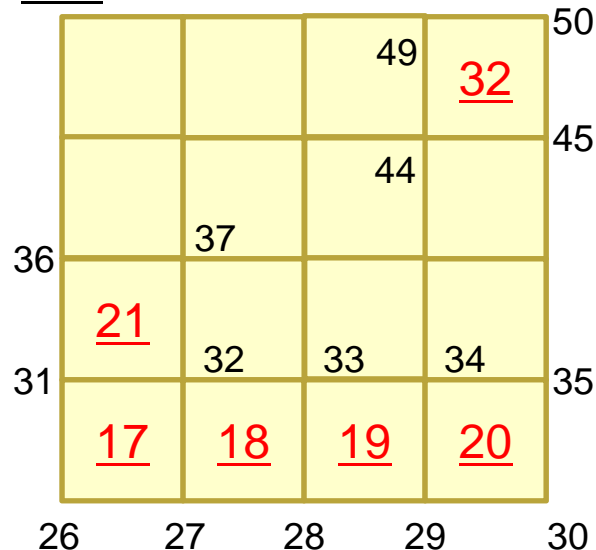
$NX=NY=NZ=4, NXP1=NYP1=NZP1=5$

$ICELTOT= 64, INODTOT= 125, IBNODTOT= 25$

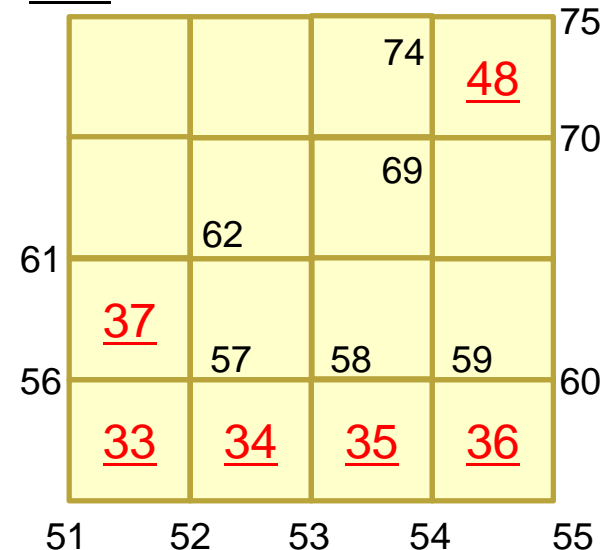
**k=1**



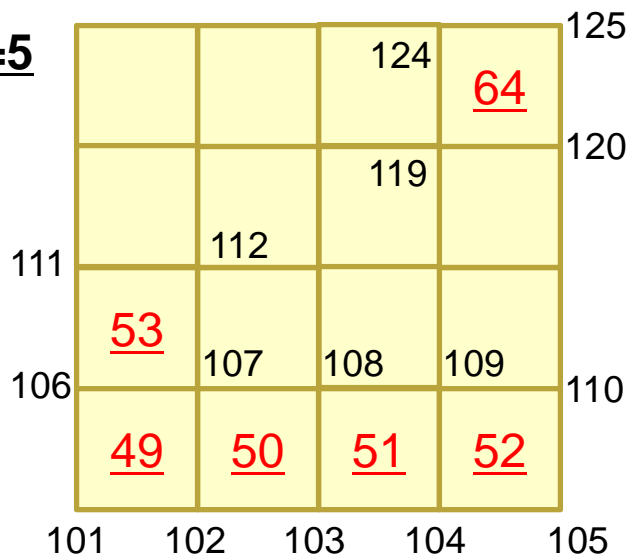
**k=2**



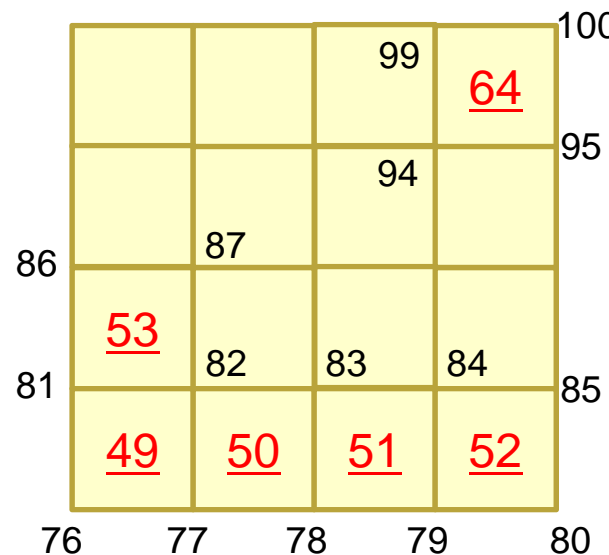
**k=3**



**k=5**

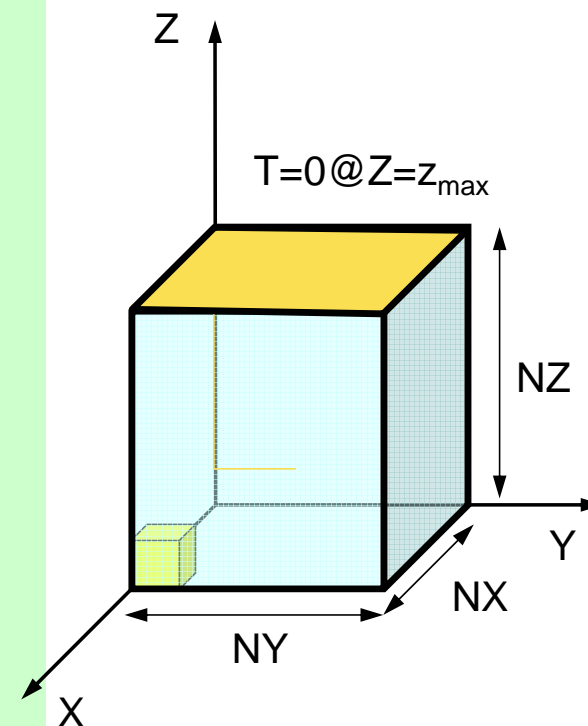


**k=4**



# Example of “cube.0” Node Grp. Info.

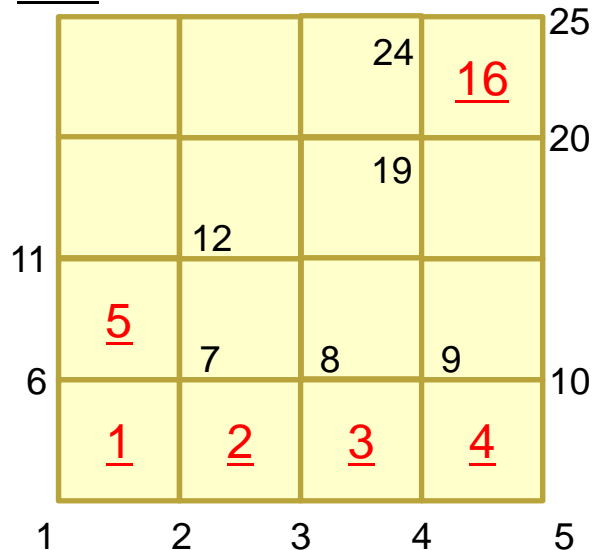
										Number of Groups	
										Number of Nodes (ea. grp.)	
4											
25	50	75	100								
Xmin											
1	6	11	16	21	26	31	36	41	46		
51	56	61	66	71	76	81	86	91	96		
101	106	111	116	121							
Ymin											
1	2	3	4	5	26	27	28	29	30		
51	52	53	54	55	76	77	78	79	80		
101	102	103	104	105							
Zmin											
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							
Zmax											
101	102	103	104	105	106	107	108	109	110		
111	112	113	114	115	116	117	118	119	120		
121	122	123	124	125							
no use after this line											



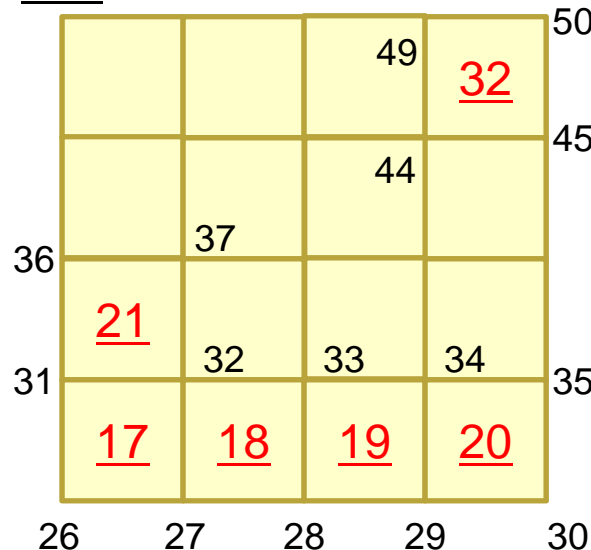
$NX=NY=NZ=4, NXP1=NYP1=NZP1=5$

$ICELTOT= 64, INODTOT= 125, IBNODTOT= 25$

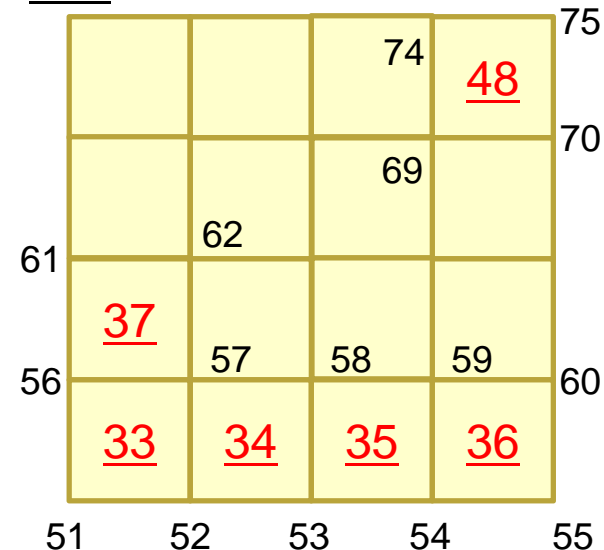
**k=1**



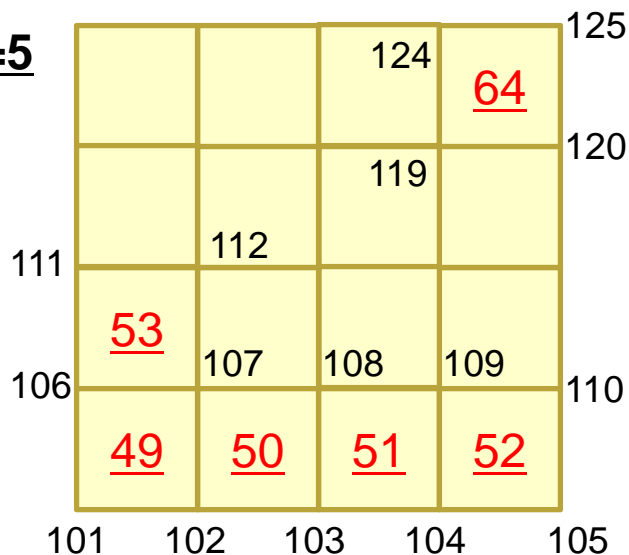
**k=2**



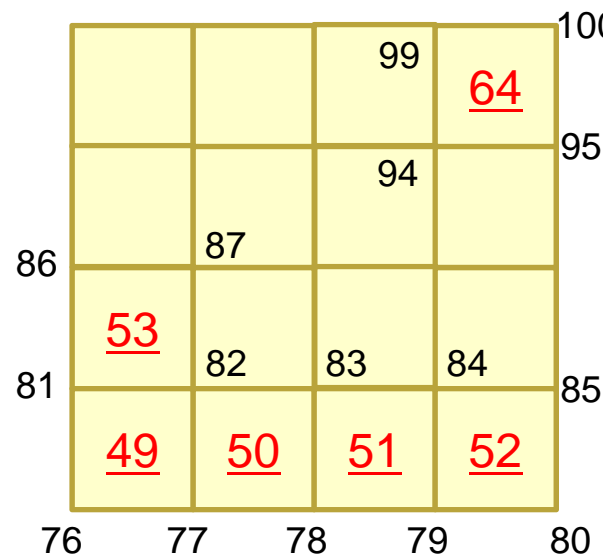
**k=3**



**k=5**



**k=4**



**Xmin: i=1  
Ymin: j=1  
Zmin: k=1  
Zmax:k=5**



# Mesh Generation

- Big Technical & Research Issue
  - Complicated Geometry
  - Large Scale
- Parallelization is difficult
- Commercial Mesh Generator
  - FEMAP
    - Interface to CAD Data Format

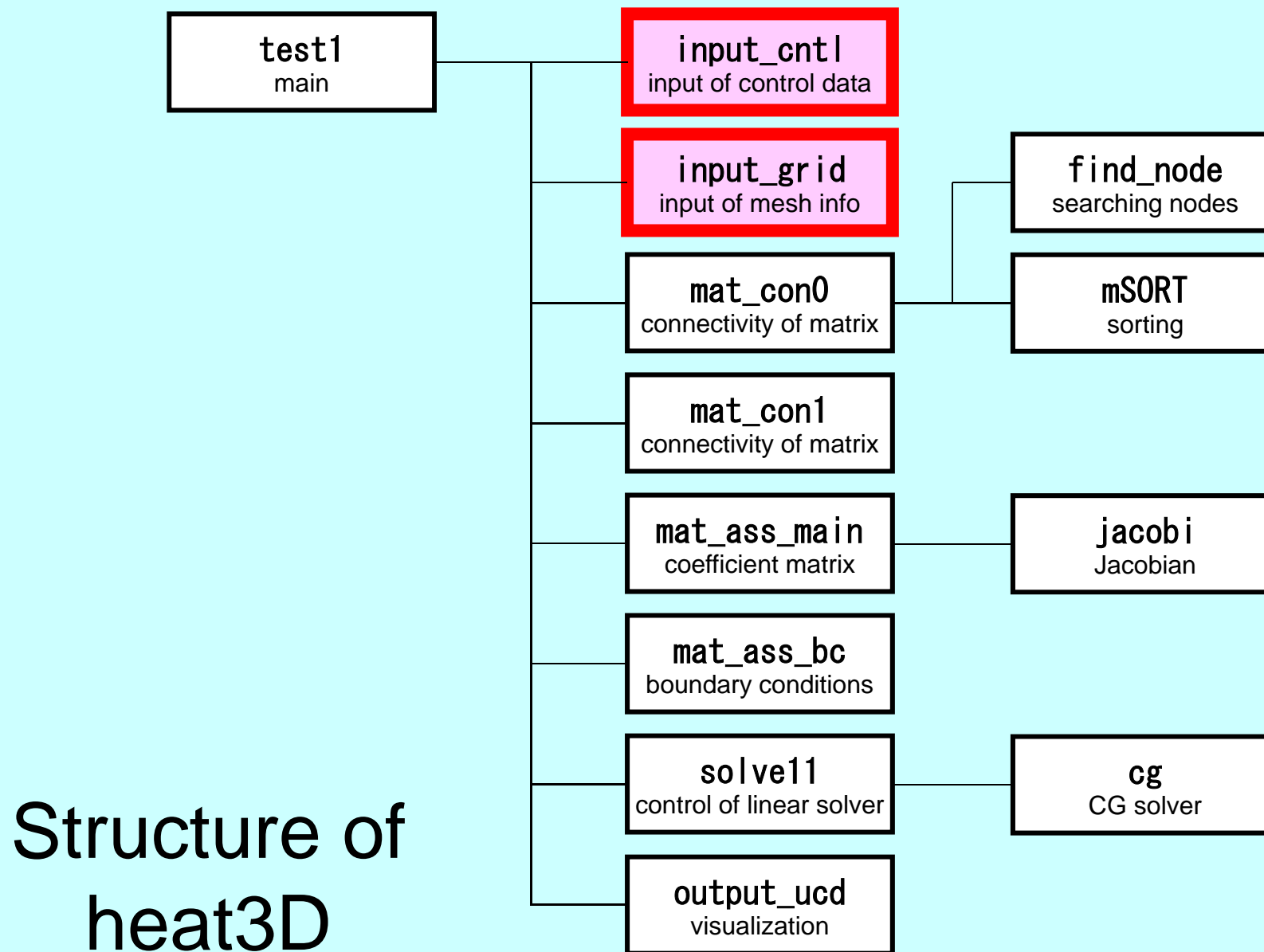


movie

- Formulation of 3D Element
- 3D Heat Equations
  - Galerkin Method
  - Element Matrices
- Running the Code
- Data Structure
- **Overview of the Program**

# FEM Procedures: Program

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method



# Main Part

```
/**
    program heat3D
**/
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
// #include "solver11.h"
extern void INPUT_CNTL ();
extern void INPUT_GRID ();
extern void MAT_CONO ();
extern void MAT_CON1 ();
extern void MAT_ASS_MAIN ();
extern void MAT_ASS_BC ();
extern void SOLVE11 ();
extern void OUTPUT_UCD ();
int main()
{
    INPUT_CNTL ();
    INPUT_GRID ();

    MAT_CONO ();
    MAT_CON1 ();

    MAT_ASS_MAIN ();
    MAT_ASS_BC ();

    SOLVE11 ();

    OUTPUT_UCD ();
}
```

# Global Variables: pfem\_util.h (1/3)

Name	Type	Size	I/O	Definition
<b>fname</b>	C	[80]	I	Name of mesh file
<b>N, NP</b>	I		I	# Node
<b>ICELTOT</b>	I		I	# Element
<b>NODGRPtot</b>	I		I	# Node Group
<b>XYZ</b>	R	[N] [3]	I	Node Coordinates
<b>ICELNOD</b>	I	[ICELTOT] [8]	I	Element Connectivity
<b>NODGRP_INDEX</b>	I	[NODGRPtot+1]	I	# Node in each Node Group
<b>NODGRP_ITEM</b>	I	[NODGRP_INDEX [NODGRPtot+1] ]	I	Node ID in each Node Group
<b>NODGRP_NAME</b>	C80	[NODGRP_INDEX [NODGRPtot+1] ]	I	Name of NodeGroup
<b>NLU</b>	I		O	# Non-Zero Off-Diagonals at each node
<b>NPLU</b>	I		O	# Non-Zero Off-Diagonals
<b>D</b>	R	[N]	O	Diagonal Block of Global Matrix
<b>B, X</b>	R	[N]	O	RHS, Unknown Vector

# Global Variables: pfem\_util.h (2/3)

Name	Type	Size	I/O	Definition
<b>AMAT</b>	R	[NPLU]	○	Non-Zero Off-Diagonal Components of Global Matrix
<b>indexLU</b>	I	[N+1]	○	# Non-Zero Off-Diagonal Components
<b>itemLU</b>	I	[NPLU]	○	Column ID of Non-Zero Off-Diagonal Components
<b>INLU</b>	I	[N]	○	Number of Non-Zero Off-Diagonal Components at Each Node
<b>IALU</b>	I	[N] [NLU]	○	Column ID of Non-Zero Off-Diagonal Components at Each Node
<b>IWKX</b>	I	[N] [2]	○	Work Arrays
<b>ITER, ITERactual</b>	I		I	Number of CG Iterations (MAX, Actual)
<b>RESID</b>	R		I	Convergence Criteria (fixed as 1.e-8)
<b>pfemIarray</b>	I	[100]	○	Integer Parameter Array
<b>pfemRarray</b>	R	[100]	○	Real Parameter Array

# Global Variables: pfem\_util.h (3/3)

Name	Type	Size	I/O	Definition
<b>O8th</b>	R		I	= 0.125
<b>PNQ, PNE, PNT</b>	R	[2][2][8]	O	$\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i=1 \sim 8)$ at each Gaussian Quad. Point
<b>POS, WEI</b>	R	[2]	O	Coordinates, Weighting Factor at each Gaussian Quad. Point
<b>NCOL1, NCOL2</b>	I	[100]	O	Work arrays for sorting
<b>SHAPE</b>	R	[2][2][2][8]	O	$N_i (i=1 \sim 8)$ at each Gaussian Quad Point
<b>PNX, PNY, PNZ</b>	R	[2][2][2][8]	O	$\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i=1 \sim 8)$ at each Gaussian Quad. Point
<b>DETJ</b>	R	[2][2][2]	O	Determinant of Jacobian Matrix at each Gaussian Quad. Point
<b>COND, QVOL</b>	R		I	Thermal Conductivity, Heat Generation Rate

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$



# INPUT\_CNTL: Control Data

```

/**
** INPUT_CNTL
**/
#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"
/** **/
void INPUT_CNTL ()
{
    FILE *fp;

    if( (fp=fopen("INPUT.DAT","r")) == NULL) {
        fprintf(stdout,"input file cannot be opened!\n");
        exit(1);
    }
    fscanf(fp,"%s",fname);
    fscanf(fp,"%d",&ITER);
    fscanf(fp,"%lf %lf", &COND, &QVOL);
    fscanf(fp,"%lf", &RESID);
    fclose(fp);

    pfemRarray[0]= RESID;
    pfemIarray[0]= ITER;
}

```

## INPUT.DAT

cube.0	fname
2000	ITER
1.0 1.0	COND, QVOL
1.0e-08	RESID

# INPUT\_GRID (1/3)

```
#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"
#include "allocate.h"
void INPUT_GRID()
{
    FILE *fp;
    int i, j, k, ii, kk, nn, icel, iS, iE;
    int NTYPE, IMAT;

    if( (fp=fopen(fname, "r")) == NULL) {
        fprintf(stdout, "input file cannot be opened!\n");
        exit(1);
    }

    /**
    NODE
    **/
    fscanf(fp, "%d", &N);
    NP=N;
    XYZ=(KREAL**) allocate_matrix(sizeof(KREAL), N, 3);

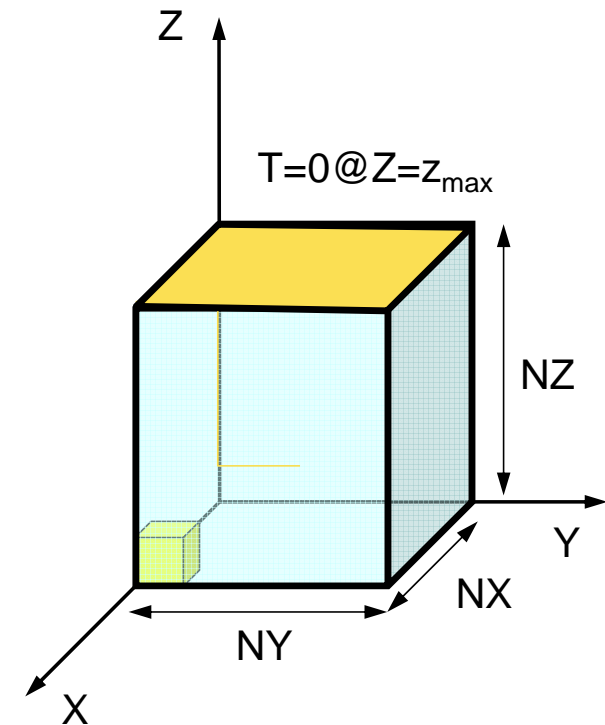
    for (i=0; i<N; i++) {
        for (j=0; j<3; j++) {
            XYZ[i][j]=0.0;
        }
    }

    for (i=0; i<N; i++) {
        fscanf(fp, "%d %lf %lf %lf", &ii, &XYZ[i][0], &XYZ[i][1], &XYZ[i][2]);
    }
}
```

# Example of “cube.0” Node

125				= N
1	0.00	0.00	0.00	
2	1.00	0.00	0.00	
3	2.00	0.00	0.00	
4	3.00	0.00	0.00	
5	4.00	0.00	0.00	
6	0.00	1.00	0.00	
7	1.00	1.00	0.00	
8	2.00	1.00	0.00	
9	3.00	1.00	0.00	
...				
121	0.00	4.00	4.00	
122	1.00	4.00	4.00	
123	2.00	4.00	4.00	
124	3.00	4.00	4.00	
125	4.00	4.00	4.00	

XYZ[i][3]



# allocate, deallocate

```

#include <stdio.h>
#include <stdlib.h>
void* allocate_vector(int size, int m)
{
    void *a;
    if ( ( a=(void *)malloc( m * size ) ) == NULL ) {
        fprintf(stdout, "Error:Memory does not enough! in vector %n");
        exit(1);
    }
    return a;
}

void deallocate_vector(void *a)
{
    free( a );
}

void** allocate_matrix(int size, int m, int n)
{
    void **aa;
    int i;
    if ( ( aa=(void **)malloc( m * sizeof(void*) ) ) == NULL ) {
        fprintf(stdout, "Error:Memory does not enough! aa in matrix %n");
        exit(1);
    }
    if ( ( aa[0]=(void *)malloc( m * n * size ) ) == NULL ) {
        fprintf(stdout, "Error:Memory does not enough! in matrix %n");
        exit(1);
    }
    for(i=1; i<m; i++) aa[i]=(char*)aa[i-1]+size*n;
    return aa;
}

void deallocate_matrix(void **aa)
{
    free( aa );
}

```

Same interface with FORTRAN

# INPUT\_GRID (2/3)

```
/**  
ELEMENT  
**/  
fscanf(fp, "%d", &ICELTOT);  
ICELNOD=(KINT**) allocate_matrix(sizeof(KINT), ICELTOT, 8);  
for (i=0; i<ICELTOT; i++) fscanf(fp, "%d", &NTYPE);  
  
for (icel=0; icel<ICELTOT; icel++) {  
    fscanf(fp, "%d %d %d %d %d %d %d %d %d %d", &i, &IMAT,  
           &ICELNOD[icel][0], &ICELNOD[icel][1], &ICELNOD[icel][2], &ICELNOD[icel][3],  
           &ICELNOD[icel][4], &ICELNOD[icel][5], &ICELNOD[icel][6], &ICELNOD[icel][7]);  
}
```

ICELNOD[i][j]:  
Node ID starting from "1".  
Element ID starts from "0".

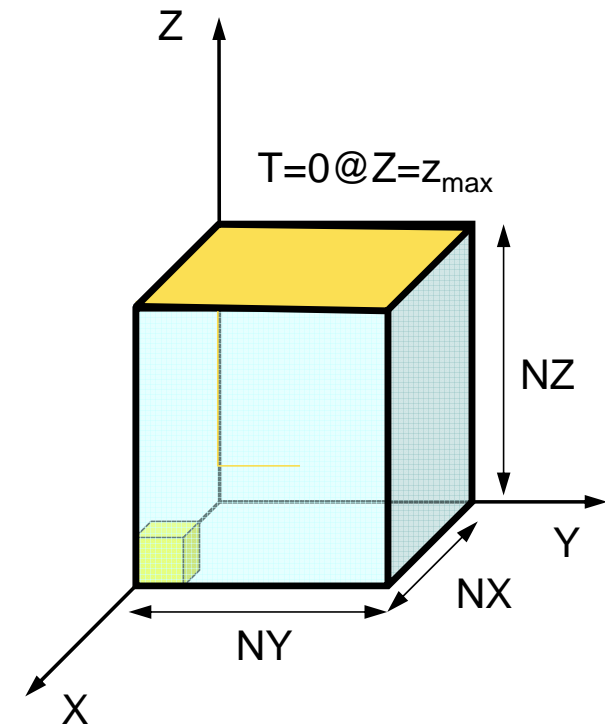
# Example of “cube.0” Element (1/2)

64 = ICELTOT

361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361
361	361	361	361	361	361	361	361	361	361

Element Type: 361

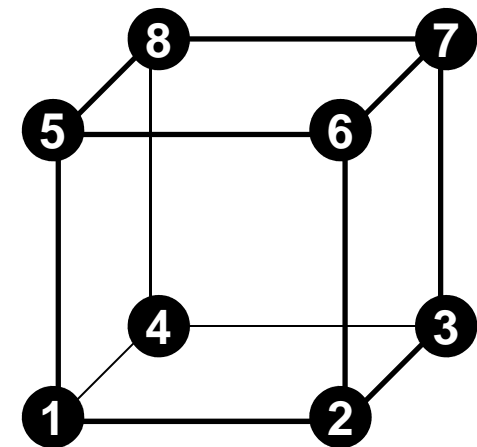
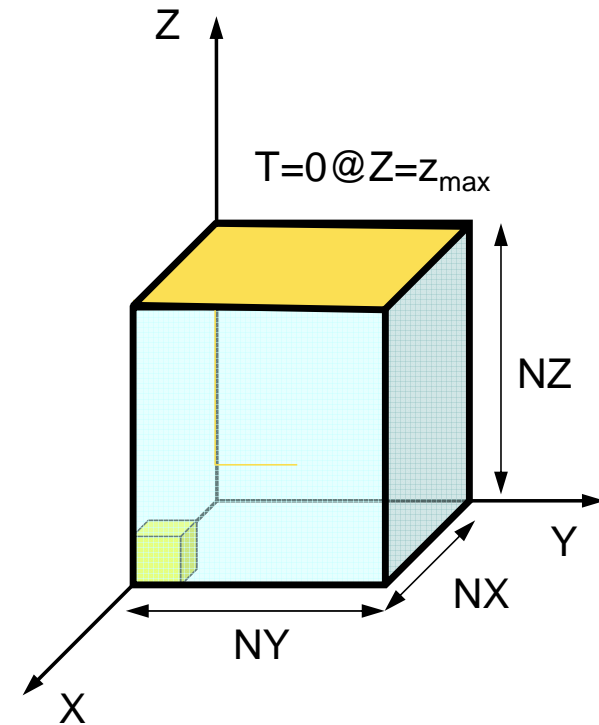
3D, Hexahedron, Linear (1<sup>st</sup> order)



# Example of “cube.0” Element (2/2)

1	1	1	2	7	6	26	27	32	31
2	1	2	3	8	7	27	28	33	32
3	1	3	4	9	8	28	29	34	33
4	1	4	5	10	9	29	30	35	34
5	1	6	7	12	11	31	32	37	36
6	1	7	8	13	12	32	33	38	37
7	1	8	9	14	13	33	34	39	38
8	1	9	10	15	14	34	35	40	39
9	1	11	12	17	16	36	37	42	41
10	1	12	13	18	17	37	38	43	42
11	1	13	14	19	18	38	39	44	43
12	1	14	15	20	19	39	40	45	44
13	1	16	17	22	21	41	42	47	46
...									
53	1	81	82	87	86	106	107	112	111
54	1	82	83	88	87	107	108	113	112
55	1	83	84	89	88	108	109	114	113
56	1	84	85	90	89	109	110	115	114
57	1	86	87	92	91	111	112	117	116
58	1	87	88	93	92	112	113	118	117
59	1	88	89	94	93	113	114	119	118
60	1	89	90	95	94	114	115	120	119
61	1	91	92	97	96	116	117	122	121
62	1	92	93	98	97	117	118	123	122
63	1	93	94	99	98	118	119	124	123
64	1	94	95	100	99	119	120	125	124

iMAT
[CELNOD[icel]][8]



# INPUT\_GRID (3/3)

```

/**
**/
NODE grp. info.
fscanf (fp, "%d", &NODGRPtot);

NODGRP_INDEX=(KINT* ) allocate_vector (sizeof (KINT), NODGRPtot+1);
NODGRP_NAME =(CHAR80*) allocate_vector (sizeof (CHAR80), NODGRPtot);
for (i=0; i<NODGRPtot+1; i++) NODGRP_INDEX[i]=0;

for (i=0; i<NODGRPtot; i++) fscanf (fp, "%d", &NODGRP_INDEX[i+1]);
nn=NODGRP_INDEX[NODGRPtot];
NODGRP_ITEM=(KINT*) allocate_vector (sizeof (KINT), nn);

for (k=0; k<NODGRPtot; k++) {
    iS= NODGRP_INDEX[k];
    iE= NODGRP_INDEX[k+1];
    fscanf (fp, "%s", NODGRP_NAME[k]. name);
    nn= iE - iS;
    if ( nn != 0 ) {
        for (kk=iS; kk<iE; kk++) fscanf (fp, "%d", &NODGRP_ITEM[kk]);
    }
}

fclose (fp);
}

```

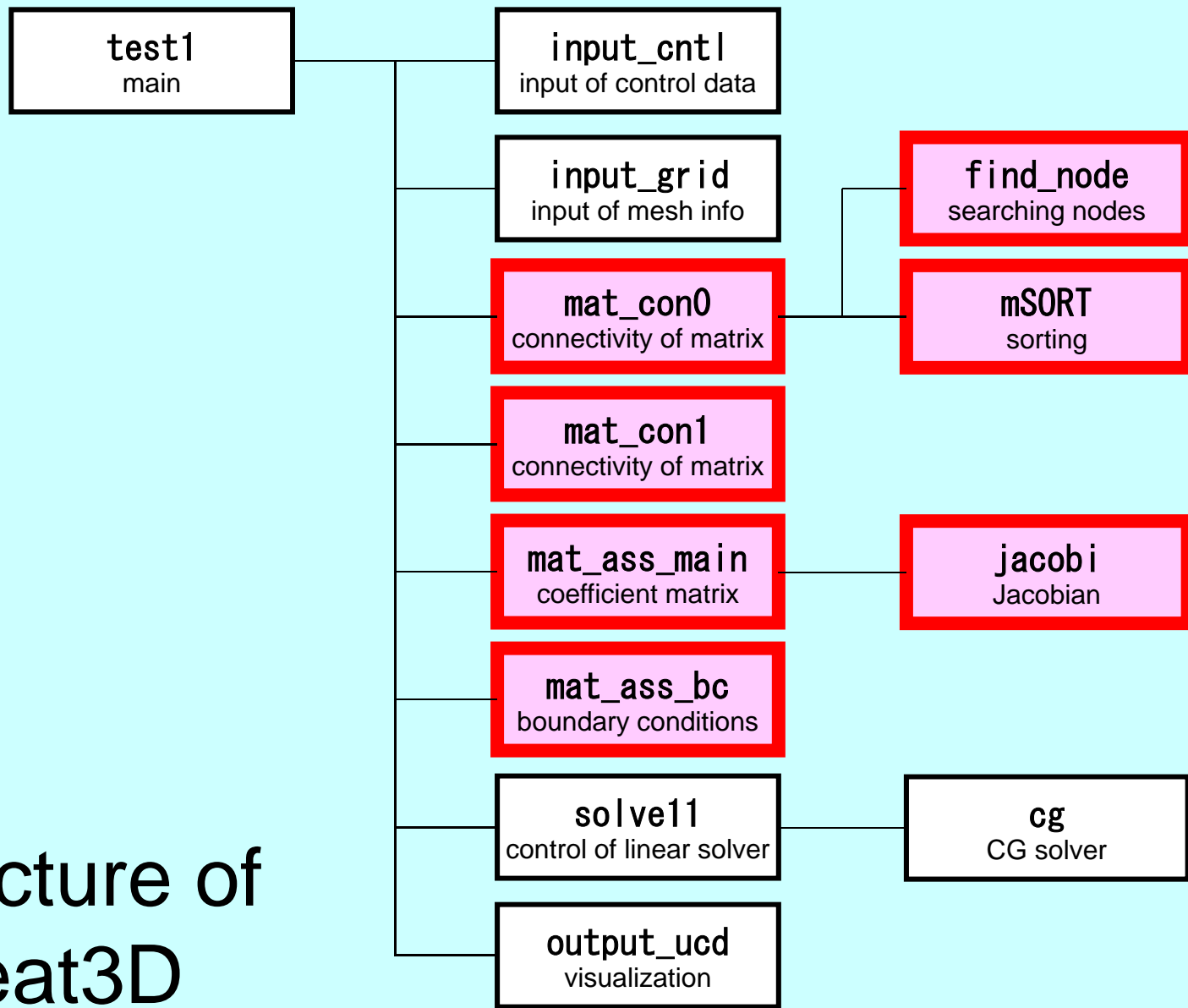
Node Group:  
Node ID's start from "1"



# Example of “cube.0” Node Grp. Info.

4											NODGRPtot
25	50	75	100								NODGRP_INDEX[1-4]
Xmin											NODGRP_NAME[0]
1	6	11	16	21	26	31	36	41	46	NODGRP_ITEM[0-24]	
51	56	61	66	71	76	81	86	91	96		
101	106	111	116	121							
Ymin											NODGRP_NAME[1]
1	2	3	4	5	26	27	28	29	30	NODGRP_ITEM[25-49]	
51	52	53	54	55	76	77	78	79	80		
101	102	103	104	105							
Zmin											NODGRP_NAME[2]
1	2	3	4	5	6	7	8	9	10	NODGRP_ITEM[50-74]	
11	12	13	14	15	16	17	18	19	20		
21	22	23	24	25							
Zmax											NODGRP_NAME[3]
101	102	103	104	105	106	107	108	109	110	NODGRP_ITEM[75-99]	
111	112	113	114	115	116	117	118	119	120		
121	122	123	124	125							

# Structure of heat3D



# Global Variables: pfem\_util.h (1/3)

Name	Type	Size	I/O	Definition
<b>fname</b>	C	[80]	I	Name of mesh file
<b>N, NP</b>	I		I	# Node
<b>ICELTOT</b>	I		I	# Element
<b>NODGRPtot</b>	I		I	# Node Group
<b>XYZ</b>	R	[N] [3]	I	Node Coordinates
<b>ICELNOD</b>	I	[ICELTOT] [8]	I	Element Connectivity
<b>NODGRP_INDEX</b>	I	[NODGRPtot+1]	I	# Node in each Node Group
<b>NODGRP_ITEM</b>	I	[NODGRP_INDEX [NODGRPtot+1] ]	I	Node ID in each Node Group
<b>NODGRP_NAME</b>	C80	[NODGRP_INDEX [NODGRPtot+1] ]	I	Name of NodeGroup
<b>NLU</b>	I		O	# Non-Zero Off-Diagonals at each node
<b>NPLU</b>	I		O	# Non-Zero Off-Diagonals
<b>D</b>	R	[N]	O	Diagonal Block of Global Matrix
<b>B, X</b>	R	[N]	O	RHS, Unknown Vector

# Global Variables: pfem\_util.h (2/3)

Name	Type	Size	I/O	Definition
<b>AMAT</b>	R	[NPLU]	○	Non-Zero Off-Diagonal Components of Global Matrix
<b>indexLU</b>	I	[N+1]	○	# Non-Zero Off-Diagonal Components
<b>itemLU</b>	I	[NPLU]	○	Column ID of Non-Zero Off-Diagonal Components
<b>INLU</b>	I	[N]	○	Number of Non-Zero Off-Diagonal Components at Each Node
<b>IALU</b>	I	[N] [NLU]	○	Column ID of Non-Zero Off-Diagonal Components at Each Node
<b>IWKX</b>	I	[N] [2]	○	Work Arrays
<b>ITER, ITERactual</b>	I		I	Number of CG Iterations (MAX, Actual)
<b>RESID</b>	R		I	Convergence Criteria (fixed as 1.e-8)
<b>pfemIarray</b>	I	[100]	○	Integer Parameter Array
<b>pfemRarray</b>	R	[100]	○	Real Parameter Array

# Global Variables: pfem\_util.h (3/3)

Name	Type	Size	I/O	Definition
<b>O8th</b>	R		I	= 0.125
<b>PNQ, PNE, PNT</b>	R	[2][2][8]	O	$\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i=1 \sim 8)$ at each Gaussian Quad. Point
<b>POS, WEI</b>	R	[2]	O	Coordinates, Weighting Factor at each Gaussian Quad. Point
<b>NCOL1, NCOL2</b>	I	[100]	O	Work arrays for sorting
<b>SHAPE</b>	R	[2][2][2][8]	O	$N_i (i=1 \sim 8)$ at each Gaussian Quad Point
<b>PNX, PNY, PNZ</b>	R	[2][2][2][8]	O	$\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i=1 \sim 8)$ at each Gaussian Quad. Point
<b>DETJ</b>	R	[2][2][2]	O	Determinant of Jacobian Matrix at each Gaussian Quad. Point
<b>COND, QVOL</b>	R		I	Thermal Conductivity, Heat Generation Rate

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

# Towards Matrix Assembling

- In 1D, it was easy to obtain information related to index and item.
  - 2 non-zero off-diagonals for each node
  - ID of non-zero off-diagonal :  $i+1$ ,  $i-1$ , where “i” is node ID
- In 3D, situation is more complicated:
  - Number of non-zero off-diagonal components is between 7 and 26 for the current target problem
  - More complicated for real problems.
  - Generally, there are no information related to number of non-zero off-diagonal components beforehand.



movie

# Towards Matrix Assembling

- In 1D, it was easy to obtain information related to index and item.
  - 2 non-zero off-diagonals for each node
  - ID of non-zero off-diagonal :  $i+1$ ,  $i-1$ , where “i” is node ID
- In 3D, situation is more complicated:
  - Number of non-zero off-diagonal components is between 7 and 26 for the current target problem
  - More complicated for real problems.
  - Generally, there are no information related to number of non-zero off-diagonal components beforehand.
- **Count number of non-zero off-diagonals using arrays: INLU[N], IALU[N][NLU]**

# Main Part

```

/**
    program heat3D
**/
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
// #include "solver11.h"
extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CON0();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE11();
extern void OUTPUT_UCD();
int main()
{
    INPUT_CNTL();
    INPUT_GRID();

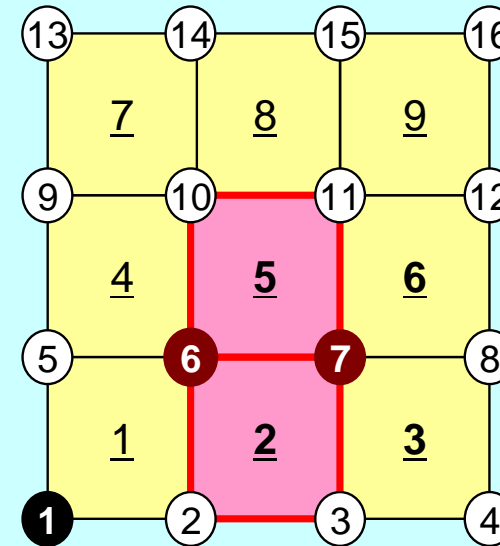
    MAT_CON0();
    MAT_CON1();

    MAT_ASS_MAIN();
    MAT_ASS_BC();

    SOLVE11();

    OUTPUT_UCD();
}

```



**MAT\_CON0:** generates INU, IALU  
**MAT\_CON1:** generates index, item

**Node ID** starting from “1”

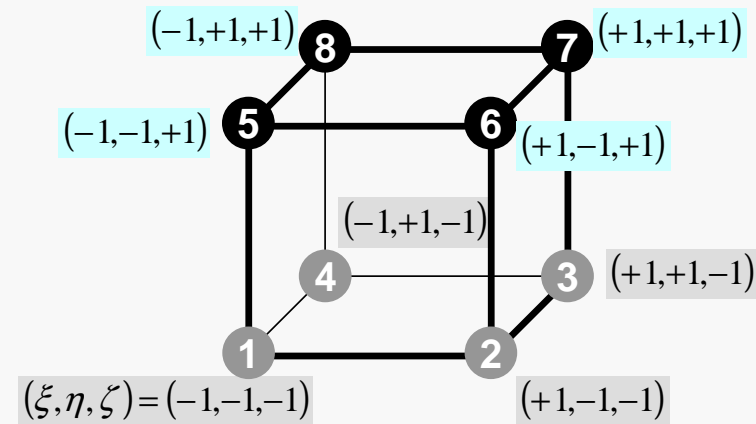
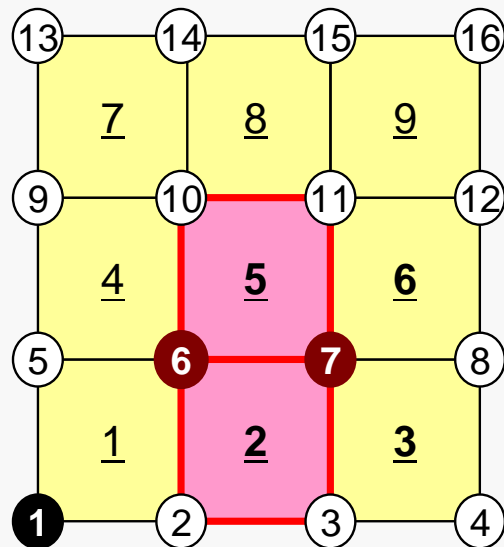


# MAT\_CON0: Overview

```

do icel= 1, ICELTOT
  generate INLU, IALU
  according to 8 nodes of hex. elements
  (FIND_NODE)
enddo

```



# Generating Connectivity of Matrix MAT\_CONO (1/4)

```

/**
** MAT_CONO
**/

#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"

extern FILE *fp_log;
/** external functions */
extern void mSORT(int*, int*, int);
/** static functions */
static void FIND_TS_NODE (int, int);

void MAT_CONO ()
{
    int i, j, k, icel, in;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    int NN;

    NLU= 26;

    INLU=(KINT* ) allocate_vector (sizeof (KINT), N) ;
    IALU=(KINT**) allocate_matrix (sizeof (KINT), N, NLU) ;

    for (i=0; i<N; i++) INLU[i]=0;
    for (i=0; i<N; i++) for (j=0; j<NLU; j++) IALU[i][j]=0;

```

## NLU:

Number of maximum number of connected nodes to each node (number of upper/lower non-zero off-diagonal nodes)

In the current problem, geometry is rather simple. Therefore we can specify NLU in this way.

If it's not clear ->  
Try more flexible implementation

# Generating Connectivity of Matrix MAT\_CON0 (1/4)

```

/**
** MAT_CON0
**/

#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"

extern FILE *fp_log;
/** external functions */
extern void mSORT(int*, int*, int);
/** static functions */
static void FIND_TS_NODE (int, int);

void MAT_CON0 ()
{
    int i, j, k, icel, in;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    int NN;

    NLU= 26;

    INLU=(KINT* ) allocate_vector (sizeof (KINT), N);
    IALU=(KINT**) allocate_matrix (sizeof (KINT), N, NLU);

    for (i=0; i<N; i++) INLU[i]=0;
    for (i=0; i<N; i++) for (j=0; j<NLU; j++) IALU[i][j]=0;

```

Array	Size	Description
INLU	[N]	Number of connected nodes to each node (lower/upper)
IALU	[N] [NLU]	Corresponding connected node ID (column ID)

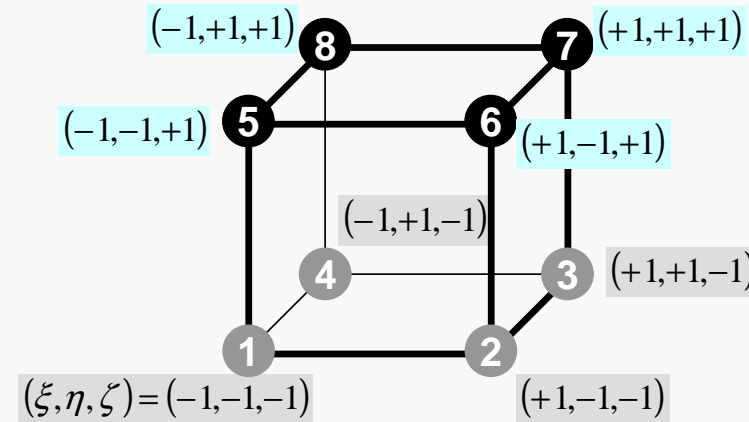
# Generating Connectivity of Matrix MAT\_CON0 (2/4): Starting from 1

```
for( icel=0; icel< ICELTOT; icel++) {
  in1=ICELNOD[ icel ][ 0 ];
  in2=ICELNOD[ icel ][ 1 ];
  in3=ICELNOD[ icel ][ 2 ];
  in4=ICELNOD[ icel ][ 3 ];
  in5=ICELNOD[ icel ][ 4 ];
  in6=ICELNOD[ icel ][ 5 ];
  in7=ICELNOD[ icel ][ 6 ];
  in8=ICELNOD[ icel ][ 7 ];
```

```
  FIND_TS_NODE (in1, in2);
  FIND_TS_NODE (in1, in3);
  FIND_TS_NODE (in1, in4);
  FIND_TS_NODE (in1, in5);
  FIND_TS_NODE (in1, in6);
  FIND_TS_NODE (in1, in7);
  FIND_TS_NODE (in1, in8);
```

```
  FIND_TS_NODE (in2, in1);
  FIND_TS_NODE (in2, in3);
  FIND_TS_NODE (in2, in4);
  FIND_TS_NODE (in2, in5);
  FIND_TS_NODE (in2, in6);
  FIND_TS_NODE (in2, in7);
  FIND_TS_NODE (in2, in8);
```

```
  FIND_TS_NODE (in3, in1);
  FIND_TS_NODE (in3, in2);
  FIND_TS_NODE (in3, in4);
  FIND_TS_NODE (in3, in5);
  FIND_TS_NODE (in3, in6);
  FIND_TS_NODE (in3, in7);
  FIND_TS_NODE (in3, in8);
```



# FIND\_TS\_NODE: Search Connectivity

## INLU,IALU: Automatic Search

```

/**
*** FIND_TS_NODE
**/

static void FIND_TS_NODE (int ip1, int ip2)
{
    int kk, icou;

    for (kk=1;kk<=INLU[ip1-1];kk++) {
        if(ip2 == IALU[ip1-1][kk-1]) return;
    }

    icou=INLU[ip1-1]+1;
    IALU[ip1-1][icou-1]=ip2;
    INLU[ip1-1]=icou;

    return;
}

```

Array	Size	Description
INLU	[N]	Number of connected nodes to each node (lower/upper)
IALU	[N] [NLU]	Corresponding connected node ID (column ID)

# FIND\_TS\_NODE: Search Connectivity

## INLU,IALU: Automatic Search

```

/**
 *** FIND_TS_NODE
 **/

static void FIND_TS_NODE (int ip1, int ip2)
{
  int kk, icou;

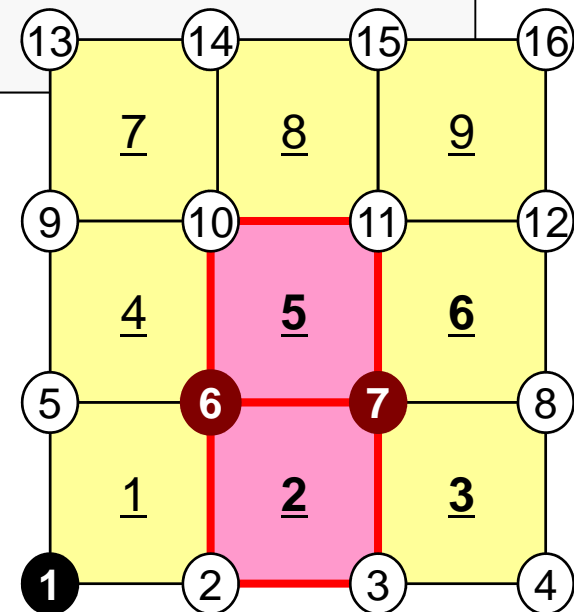
  for (kk=1;kk<=INLU[ip1-1];kk++) {
    if(ip2 == IALU[ip1-1][kk-1]) return;
  }

  icou=INLU[ip1-1]+1;
  IALU[ip1-1][icou-1]=ip2;
  INLU[ip1-1]=icou;

  return;
}

```

If the target node is already included in IALU, proceed to next pair of nodes



# FIND\_TS\_NODE: Search Connectivity

## INLU,IALU: Automatic Search

```

/**
 *** FIND_TS_NODE
 **/

static void FIND_TS_NODE (int ip1, int ip2)
{
  int kk, icou;

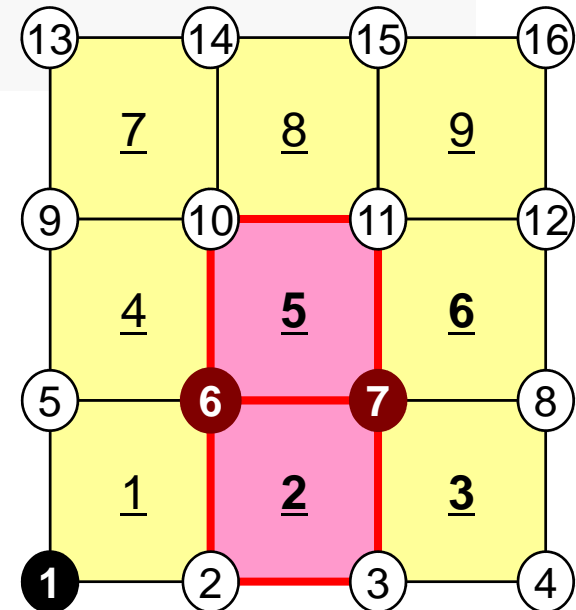
  for (kk=1;kk<=INLU[ip1-1];kk++) {
    if(ip2 == IALU[ip1-1][kk-1]) return;
  }

  icou=INLU[ip1-1]+1;
  IALU[ip1-1][icou-1]=ip2;
  INLU[ip1-1]=icou;

  return;
}

```

If the target node is NOT included in IALU, store the node in IALU, and add 1 to INLU.



# Generating Connectivity of Matrix MAT\_CON0 (3/4)

```

FIND_TS_NODE (in4, in1);
FIND_TS_NODE (in4, in2);
FIND_TS_NODE (in4, in3);
FIND_TS_NODE (in4, in5);
FIND_TS_NODE (in4, in6);
FIND_TS_NODE (in4, in7);
FIND_TS_NODE (in4, in8);

```

```

FIND_TS_NODE (in5, in1);
FIND_TS_NODE (in5, in2);
FIND_TS_NODE (in5, in3);
FIND_TS_NODE (in5, in4);
FIND_TS_NODE (in5, in6);
FIND_TS_NODE (in5, in7);
FIND_TS_NODE (in5, in8);

```

```

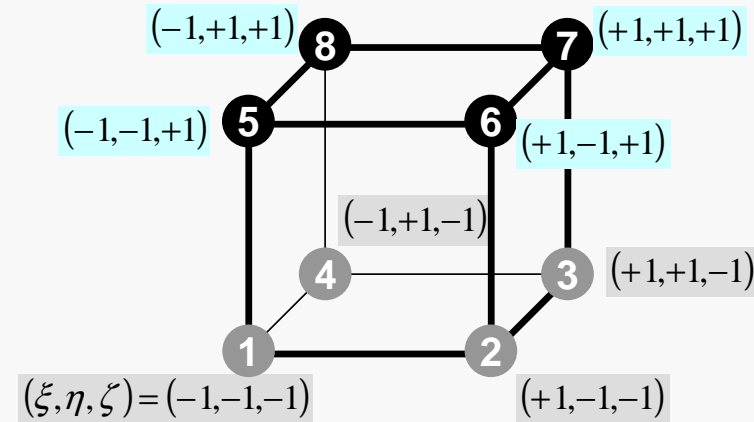
FIND_TS_NODE (in6, in1);
FIND_TS_NODE (in6, in2);
FIND_TS_NODE (in6, in3);
FIND_TS_NODE (in6, in4);
FIND_TS_NODE (in6, in5);
FIND_TS_NODE (in6, in7);
FIND_TS_NODE (in6, in8);

```

```

FIND_TS_NODE (in7, in1);
FIND_TS_NODE (in7, in2);
FIND_TS_NODE (in7, in3);
FIND_TS_NODE (in7, in4);
FIND_TS_NODE (in7, in5);
FIND_TS_NODE (in7, in6);
FIND_TS_NODE (in7, in8);

```





# Generating Connectivity of Matrix MAT\_CON0 (4/4)

```
FIND_TS_NODE (in8, in1);  
FIND_TS_NODE (in8, in2);  
FIND_TS_NODE (in8, in3);  
FIND_TS_NODE (in8, in4);  
FIND_TS_NODE (in8, in5);  
FIND_TS_NODE (in8, in6);  
FIND_TS_NODE (in8, in7);  
}  
  
for (in=0; in<N; in++) {  
    NN=INLU[in];  
    for (k=0; k<NN; k++) {  
        NCOL1[k]=IALU[in][k];  
    }  
  
    mSORT (NCOL1, NCOL2, NN);  
  
    for (k=NN; k>0; k--) {  
        IALU[in][NN-k]= NCOL1 [NCOL2[k-1]-1];  
    }  
}
```

Sort IALU[i][k] in ascending order by  
“bubble” sorting for less than 100  
components.

# MAT\_CON1: CRS format

```

#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1 ()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<N; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[N];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<N; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}

```

C

$$\text{index}[i+1] = \sum_{k=0}^i \text{INLU}[k]$$

$$\text{index}[0] = 0$$

FORTRAN

$$\text{index}(i) = \sum_{k=1}^i \text{INLU}(k)$$

$$\text{index}(0) = 0$$

# MAT\_CON1: CRS format

```

#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1 ()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<N; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[N];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<N; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}
MAT_CON1

```

**NPLU=indexLU[N]**  
**Size of array: itemLU**  
**Total number of non-zero off-diagonal blocks**

# MAT\_CON1: CRS format

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1 ()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<N; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[N];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<N; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}

MAT_CON1
```

itemLU

store node ID starting from 0

# MAT\_CON1: CRS format

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1 ()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<N; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[N];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<N; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}
MAT_CON1
```

Not required any more

# Main Part

```
/**
    program heat3D
**/
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
// #include "solver11.h"
extern void INPUT_CNTL ();
extern void INPUT_GRID ();
extern void MAT_CONO ();
extern void MAT_CON1 ();
extern void MAT_ASS_MAIN ();
extern void MAT_ASS_BC ();
extern void SOLVE11 ();
extern void OUTPUT_UCD ();
int main()
{
    INPUT_CNTL ();
    INPUT_GRID ();

    MAT_CONO ();
    MAT_CON1 ();

    MAT_ASS_MAIN ();
    MAT_ASS_BC ();

    SOLVE11 ();

    OUTPUT_UCD ();
}
```

# MAT\_ASS\_MAIN: Overview

```

do kpn= 1, 2      Gaussian Quad. points in  $\zeta$ -direction
  do jpn= 1, 2    Gaussian Quad. points in  $\eta$ -direction
    do ipn= 1, 2  Gaussian Quad. Points in  $\xi$ -direction
      Define Shape Function at Gaussian Quad. Points (8-points)
      Its derivative on natural/local coordinate is also defined.
    enddo
  enddo
enddo

```

```

do icel= 1, ICELTOT  Loop for Element
  Jacobian and derivative on global coordinate of shape functions at
  Gaussian Quad. Points are defined according to coordinates of 8 nodes. (JACOBI)

```

```

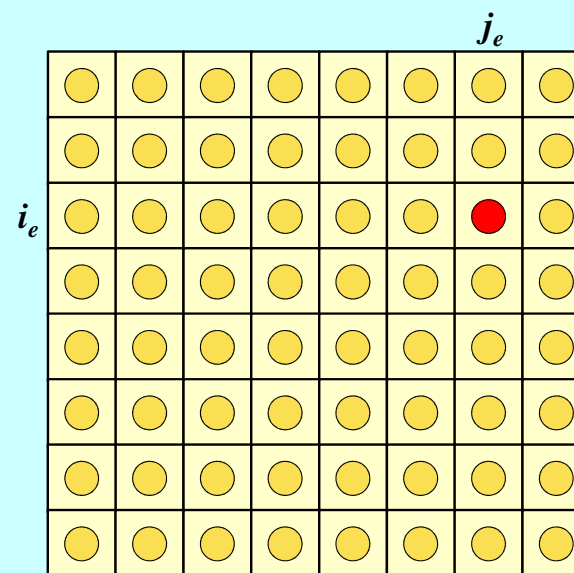
do ie= 1, 8          Local Node ID
  do je= 1, 8        Local Node ID
    Global Node ID: ip, jp
    Address of  $A_{ip, jp}$  in "itemLU" : kk

```

```

do kpn= 1, 2      Gaussian Quad. points in  $\zeta$ -direction
  do jpn= 1, 2    Gaussian Quad. points in  $\eta$ -direction
    do ipn= 1, 2  Gaussian Quad. points in  $\xi$ -direction
      integration on each element
      coefficients of element matrices
      accumulation to global matrix
    enddo
  enddo
enddo
enddo
enddo
enddo

```



# MAT\_ASS\_MAIN (1/6)

```
#include <stdio.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void JACOBI();
void MAT_ASS_MAIN()
{
    int i, k, kk;
    int ip, jp, kp;
    int ipn, jpn, kpn;
    int icel;
    int ie, je;
    int iiS, iiE;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    double SHi;
    double QP1, QM1, EP1, EM1, TP1, TM1;
    double X1, X2, X3, X4, X5, X6, X7, X8;
    double Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8;
    double Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8;
    double PNXi, PNYi, PNZi, PNXj, PNYj, PNZj;
    double CONDO, QVO, QVC, COEFij;
    double coef;
```

```
KINT nodLOCAL[8];
```

```
AMAT=(KREAL*) allocate_vector(sizeof(KREAL), NPLU);
B =(KREAL*) allocate_vector(sizeof(KREAL), N );
D =(KREAL*) allocate_vector(sizeof(KREAL), N);
X =(KREAL*) allocate_vector(sizeof(KREAL), N);
```

```
Non-Zero Off-Diagonal components (coef. matrix)
RHS vector
Diagonal components (coef. matrix)
Unknowns
```

```
for (i=0; i<NPLU; i++) AMAT[i]=0.0;
for (i=0; i<N ; i++) B[i]=0.0;
for (i=0; i<N ; i++) D[i]=0.0;
for (i=0; i<N ; i++) X[i]=0.0;
```

```
WEI[0]= 1.0000000000e0;
WEI[1]= 1.0000000000e0;
POS[0]= -0.5773502692e0;
POS[1]= 0.5773502692e0;
```



# MAT\_ASS\_MAIN (1/6)

```

#include <stdio.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void JACOBI();
void MAT_ASS_MAIN()
{
    int i, k, kk;
    int ip, jp, kp;
    int ipn, jpn, kpn;
    int icel;
    int ie, je;
    int iiS, iiE;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    double SHi;
    double QP1, QM1, EP1, EM1, TP1, TM1;
    double X1, X2, X3, X4, X5, X6, X7, X8;
    double Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8;
    double Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8;
    double PNXi, PNYi, PNZi, PNXj, PNYj, PNZj;
    double CONDO, QVO, QVC, COEFij;
    double coef;

```

```

KINT nodLOCAL[8];

```

```

AMAT=(KREAL*) allocate_vector(sizeof(KREAL), NPLU);
B=(KREAL*) allocate_vector(sizeof(KREAL), N);
D=(KREAL*) allocate_vector(sizeof(KREAL), N);
X=(KREAL*) allocate_vector(sizeof(KREAL), N);

```

```

for(i=0; i<NPLU; i++) AMAT[i]=0.0;
for(i=0; i<N; i++) B[i]=0.0;
for(i=0; i<N; i++) D[i]=0.0;
for(i=0; i<N; i++) X[i]=0.0;

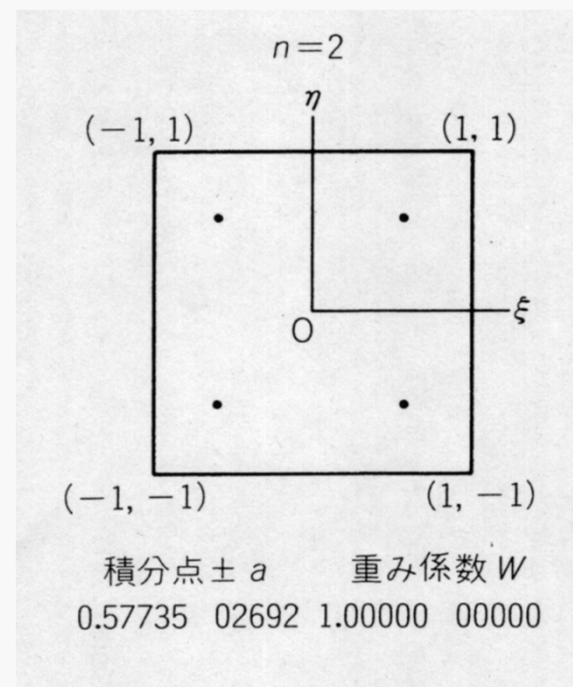
```

```

WEI[0]= 1.000000000e0;
WEI[1]= 1.000000000e0;
POS[0]= -0.5773502692e0;
POS[1]= 0.5773502692e0;

```

**POS:** Quad. Point  
**WEI:** Weighting Factor



# MAT\_ASS\_MAIN (2/6)

```
/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

for (ip=0; ip<2; ip++) {
  for (jp=0; jp<2; jp++) {
    for (kp=0; kp<2; kp++) {

      QP1= 1. e0 + POS[ip] ;
      QM1= 1. e0 - POS[ip] ;
      EP1= 1. e0 + POS[jp] ;
      EM1= 1. e0 - POS[jp] ;
      TP1= 1. e0 + POS[kp] ;
      TM1= 1. e0 - POS[kp] ;

      SHAPE[ip][jp][kp][0]= 08th * QM1 * EM1 * TM1 ;
      SHAPE[ip][jp][kp][1]= 08th * QP1 * EM1 * TM1 ;
      SHAPE[ip][jp][kp][2]= 08th * QP1 * EP1 * TM1 ;
      SHAPE[ip][jp][kp][3]= 08th * QM1 * EP1 * TM1 ;
      SHAPE[ip][jp][kp][4]= 08th * QM1 * EM1 * TP1 ;
      SHAPE[ip][jp][kp][5]= 08th * QP1 * EM1 * TP1 ;
      SHAPE[ip][jp][kp][6]= 08th * QP1 * EP1 * TP1 ;
      SHAPE[ip][jp][kp][7]= 08th * QM1 * EP1 * TP1 ;
```

# MAT\_ASS\_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

```

```

for (ip=0; ip<2; ip++) {
  for (jp=0; jp<2; jp++) {
    for (kp=0; kp<2; kp++) {

```

```

      QP1= 1. e0 + POS[ip];
      QM1= 1. e0 - POS[ip];
      EP1= 1. e0 + POS[jp];
      EM1= 1. e0 - POS[jp];
      TP1= 1. e0 + POS[kp];
      TM1= 1. e0 - POS[kp];

```

```

      SHAPE[ip][jp][kp][0] = 08th * QM1 * EM1 * TM1;
      SHAPE[ip][jp][kp][1] = 08th * QP1 * EM1 * TM1;
      SHAPE[ip][jp][kp][2] = 08th * QP1 * EP1 * TM1;
      SHAPE[ip][jp][kp][3] = 08th * QM1 * EP1 * TM1;
      SHAPE[ip][jp][kp][4] = 08th * QM1 * EM1 * TP1;
      SHAPE[ip][jp][kp][5] = 08th * QP1 * EM1 * TP1;
      SHAPE[ip][jp][kp][6] = 08th * QP1 * EP1 * TP1;
      SHAPE[ip][jp][kp][7] = 08th * QM1 * EP1 * TP1;

```

$$QP1(i) = (1 + \xi_i), \quad QM1(i) = (1 - \xi_i)$$

$$EP1(j) = (1 + \eta_j), \quad EM1(j) = (1 - \eta_j)$$

$$TP1(k) = (1 + \zeta_k), \quad TM1(k) = (1 - \zeta_k)$$

# MAT\_ASS\_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

```

```

for (ip=0; ip<2; ip++) {
  for (jp=0; jp<2; jp++) {
    for (kp=0; kp<2; kp++) {

```

```

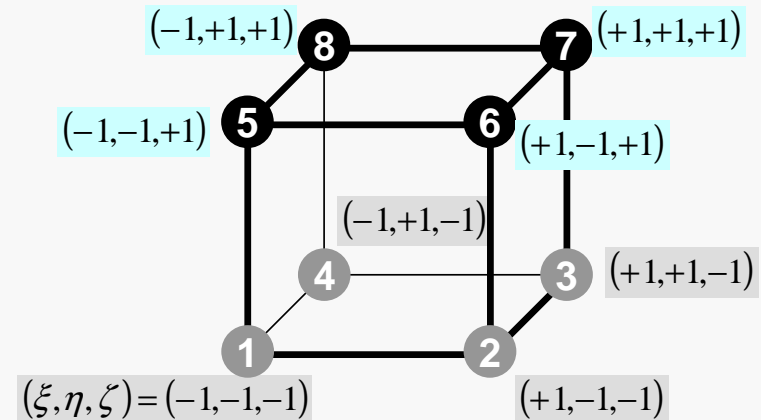
      QP1= 1. e0 + POS[ip] :
      QM1= 1. e0 - POS[ip] :
      EP1= 1. e0 + POS[jp] :
      EM1= 1. e0 - POS[jp] :
      TP1= 1. e0 + POS[kp] :
      TM1= 1. e0 - POS[kp] :

```

```

      SHAPE[ip][jp][kp][0] = 08th * QM1 * EM1 * TM1 ;
      SHAPE[ip][jp][kp][1] = 08th * QP1 * EM1 * TM1 ;
      SHAPE[ip][jp][kp][2] = 08th * QP1 * EP1 * TM1 ;
      SHAPE[ip][jp][kp][3] = 08th * QM1 * EP1 * TM1 ;
      SHAPE[ip][jp][kp][4] = 08th * QM1 * EM1 * TP1 ;
      SHAPE[ip][jp][kp][5] = 08th * QP1 * EM1 * TP1 ;
      SHAPE[ip][jp][kp][6] = 08th * QP1 * EP1 * TP1 ;
      SHAPE[ip][jp][kp][7] = 08th * QM1 * EP1 * TP1 ;

```



# MAT\_ASS\_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

for (ip=0; ip<2; ip++) {
  for (jp=0; jp<2; jp++) {
    for (kp=0; kp<2; kp++) {

      QP1= 1. e0 + POS[ip] :
      QM1= 1. e0 - POS[ip] :
      EP1= 1. e0 + POS[jp] :
      EM1= 1. e0 - POS[jp] :
      TP1= 1. e0 + POS[kp] :
      TM1= 1. e0 - POS[kp] :

      SHAPE[ip][jp][kp][0] = 08th * QM1 * EM1 * TM1 ;
      SHAPE[ip][jp][kp][1] = 08th * QP1 * EM1 * TM1 ;
      SHAPE[ip][jp][kp][2] = 08th * QP1 * EP1 * TM1 ;
      SHAPE[ip][jp][kp][3] = 08th * QM1 * EP1 * TM1 ;
      SHAPE[ip][jp][kp][4] = 08th * QM1 * EM1 * TP1 ;
      SHAPE[ip][jp][kp][5] = 08th * QP1 * EM1 * TP1 ;
      SHAPE[ip][jp][kp][6] = 08th * QP1 * EP1 * TP1 ;
      SHAPE[ip][jp][kp][7] = 08th * QM1 * EP1 * TP1 ;
    }
  }
}

```

$$N_1(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)$$

$$N_2(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$$

$$N_3(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)$$

$$N_4(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$$

$$N_5(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$$

$$N_6(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$

$$N_7(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$$

$$N_8(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

# MAT\_ASS\_MAIN (3/6)

```

PNQ [jp] [kp] [0] = - 08th * EM1 * TM1 ;
PNQ [jp] [kp] [1] = + 08th * EM1 * TM1 ;
PNQ [jp] [kp] [2] = + 08th * EP1 * TM1 ;
PNQ [jp] [kp] [3] = - 08th * EP1 * TM1 ;
PNQ [jp] [kp] [4] = - 08th * EM1 * TP1 ;
PNQ [jp] [kp] [5] = + 08th * EM1 * TP1 ;
PNQ [jp] [kp] [6] = + 08th * EP1 * TP1 ;
PNQ [jp] [kp] [7] = - 08th * EP1 * TP1 ;

```

```

PNE [ip] [kp] [0] = - 08th * QM1 * TM1 ;
PNE [ip] [kp] [1] = - 08th * QP1 * TM1 ;
PNE [ip] [kp] [2] = + 08th * QP1 * TM1 ;
PNE [ip] [kp] [3] = + 08th * QM1 * TM1 ;
PNE [ip] [kp] [4] = - 08th * QM1 * TP1 ;
PNE [ip] [kp] [5] = - 08th * QP1 * TP1 ;
PNE [ip] [kp] [6] = + 08th * QP1 * TP1 ;
PNE [ip] [kp] [7] = + 08th * QM1 * TP1 ;

```

```

PNT [ip] [jp] [0] = - 08th * QM1 * EM1 ;
PNT [ip] [jp] [1] = - 08th * QP1 * EM1 ;
PNT [ip] [jp] [2] = - 08th * QP1 * EP1 ;
PNT [ip] [jp] [3] = - 08th * QM1 * EP1 ;
PNT [ip] [jp] [4] = + 08th * QM1 * EM1 ;
PNT [ip] [jp] [5] = + 08th * QP1 * EM1 ;
PNT [ip] [jp] [6] = + 08th * QP1 * EP1 ;
PNT [ip] [jp] [7] = + 08th * QM1 * EP1 ;

```

```

}
}
}
for ( icel=0; icel < ICELTOT; icel++) {
  CONDO= COND;

```

```

in1=ICELNOD [ icel ] [0] ;
in2=ICELNOD [ icel ] [1] ;
in3=ICELNOD [ icel ] [2] ;
in4=ICELNOD [ icel ] [3] ;
in5=ICELNOD [ icel ] [4] ;
in6=ICELNOD [ icel ] [5] ;
in7=ICELNOD [ icel ] [6] ;
in8=ICELNOD [ icel ] [7] ;

```

$$PNQ(j,k) = \frac{\partial N_l}{\partial \xi} (\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$PNE(i,k) = \frac{\partial N_l}{\partial \eta} (\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$PNT(i,j) = \frac{\partial N_l}{\partial \zeta} (\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$\frac{\partial N_1}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = -\frac{1}{8} (1 - \eta_j) (1 - \zeta_k)$$

$$\frac{\partial N_2}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = +\frac{1}{8} (1 - \eta_j) (1 - \zeta_k)$$

$$\frac{\partial N_3}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = +\frac{1}{8} (1 + \eta_j) (1 - \zeta_k)$$

$$\frac{\partial N_4}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = -\frac{1}{8} (1 + \eta_j) (1 + \zeta_k)$$

First Order Derivative  
of Shape Functions at  
 $(\xi_i, \eta_j, \zeta_k)$

# MAT\_ASS\_MAIN (3/6)

```

PNQ [jp] [kp] [0] = - 08th * EM1 * TM1 ;
PNQ [jp] [kp] [1] = + 08th * EM1 * TM1 ;
PNQ [jp] [kp] [2] = + 08th * EP1 * TM1 ;
PNQ [jp] [kp] [3] = - 08th * EP1 * TM1 ;
PNQ [jp] [kp] [4] = - 08th * EM1 * TP1 ;
PNQ [jp] [kp] [5] = + 08th * EM1 * TP1 ;
PNQ [jp] [kp] [6] = + 08th * EP1 * TP1 ;
PNQ [jp] [kp] [7] = - 08th * EP1 * TP1 ;

```

```

PNE [ip] [kp] [0] = - 08th * QM1 * TM1 ;
PNE [ip] [kp] [1] = - 08th * QP1 * TM1 ;
PNE [ip] [kp] [2] = + 08th * QP1 * TM1 ;
PNE [ip] [kp] [3] = + 08th * QM1 * TM1 ;
PNE [ip] [kp] [4] = - 08th * QM1 * TP1 ;
PNE [ip] [kp] [5] = - 08th * QP1 * TP1 ;
PNE [ip] [kp] [6] = + 08th * QP1 * TP1 ;
PNE [ip] [kp] [7] = + 08th * QM1 * TP1 ;

```

```

PNT [ip] [jp] [0] = - 08th * QM1 * EM1 ;
PNT [ip] [jp] [1] = - 08th * QP1 * EM1 ;
PNT [ip] [jp] [2] = - 08th * QP1 * EP1 ;
PNT [ip] [jp] [3] = - 08th * QM1 * EP1 ;
PNT [ip] [jp] [4] = + 08th * QM1 * EM1 ;
PNT [ip] [jp] [5] = + 08th * QP1 * EM1 ;
PNT [ip] [jp] [6] = + 08th * QP1 * EP1 ;
PNT [ip] [jp] [7] = + 08th * QM1 * EP1 ;

```

```

}
}
}

```

```

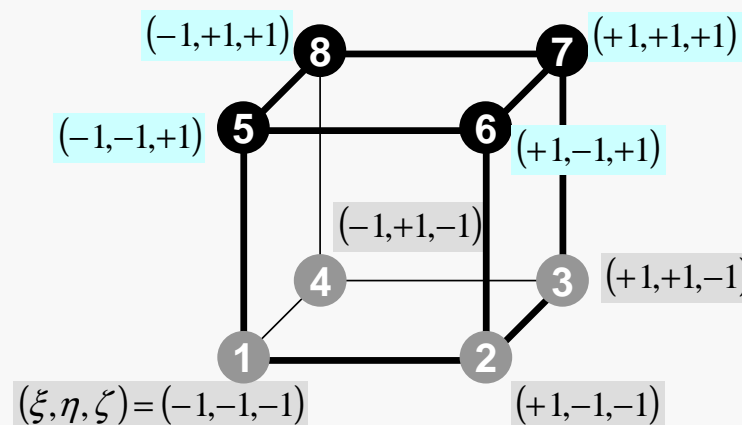
for ( icel=0; icel < ICELTOT; icel++) {
  CONDO= COND;

```

```

  in1=ICELNOD [ icel ] [ 0 ] ;
  in2=ICELNOD [ icel ] [ 1 ] ;
  in3=ICELNOD [ icel ] [ 2 ] ;
  in4=ICELNOD [ icel ] [ 3 ] ;
  in5=ICELNOD [ icel ] [ 4 ] ;
  in6=ICELNOD [ icel ] [ 5 ] ;
  in7=ICELNOD [ icel ] [ 6 ] ;
  in8=ICELNOD [ icel ] [ 7 ] ;

```



# MAT\_ASS\_MAIN (4/6)

```

nodLOCAL [0] = in1 ;
nodLOCAL [1] = in2 ;
nodLOCAL [2] = in3 ;
nodLOCAL [3] = in4 ;
nodLOCAL [4] = in5 ;
nodLOCAL [5] = in6 ;
nodLOCAL [6] = in7 ;
nodLOCAL [7] = in8 ;

```

```

X1=XYZ[in1-1][0] ;
X2=XYZ[in2-1][0] ;
X3=XYZ[in3-1][0] ;
X4=XYZ[in4-1][0] ;
X5=XYZ[in5-1][0] ;
X6=XYZ[in6-1][0] ;
X7=XYZ[in7-1][0] ;
X8=XYZ[in8-1][0] ;

```

```

Y1=XYZ[in1-1][1] ;
Y2=XYZ[in2-1][1] ;
Y3=XYZ[in3-1][1] ;
Y4=XYZ[in4-1][1] ;
Y5=XYZ[in5-1][1] ;
Y6=XYZ[in6-1][1] ;
Y7=XYZ[in7-1][1] ;
Y8=XYZ[in8-1][1] ;

```

```
QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8) ;
```

```

Z1=XYZ[in1-1][2] ;
Z2=XYZ[in2-1][2] ;
Z3=XYZ[in3-1][2] ;
Z4=XYZ[in4-1][2] ;
Z5=XYZ[in5-1][2] ;
Z6=XYZ[in6-1][2] ;
Z7=XYZ[in7-1][2] ;
Z8=XYZ[in8-1][2] ;

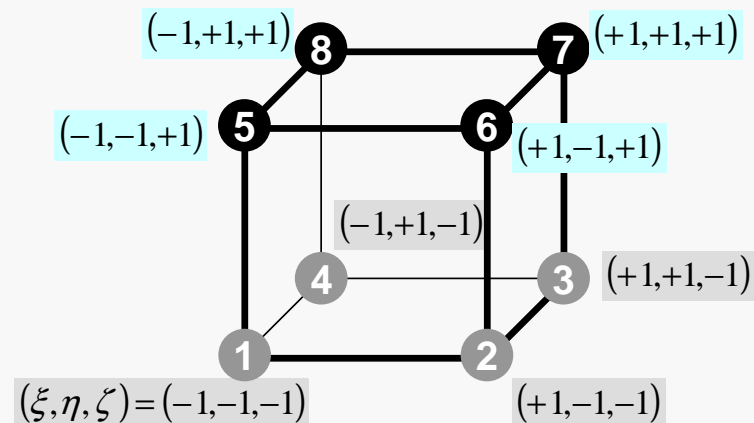
```

```

JACOBI (DETJ, PNQ, PNE, PNT, PNX, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8) ;

```

Node ID (Global)





# MAT\_ASS\_MAIN (4/6)

```
nodLOCAL [0] = in1 ;
nodLOCAL [1] = in2 ;
nodLOCAL [2] = in3 ;
nodLOCAL [3] = in4 ;
nodLOCAL [4] = in5 ;
nodLOCAL [5] = in6 ;
nodLOCAL [6] = in7 ;
nodLOCAL [7] = in8 ;
```

```
X1=XYZ [in1-1] [0] ;
X2=XYZ [in2-1] [0] ;
X3=XYZ [in3-1] [0] ;
X4=XYZ [in4-1] [0] ;
X5=XYZ [in5-1] [0] ;
X6=XYZ [in6-1] [0] ;
X7=XYZ [in7-1] [0] ;
X8=XYZ [in8-1] [0] ;
```

X-Coordinates  
of 8 nodes

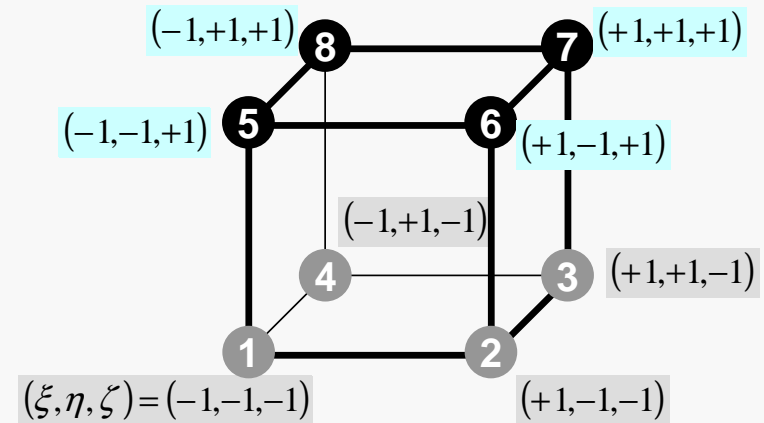
```
Y1=XYZ [in1-1] [1] ;
Y2=XYZ [in2-1] [1] ;
Y3=XYZ [in3-1] [1] ;
Y4=XYZ [in4-1] [1] ;
Y5=XYZ [in5-1] [1] ;
Y6=XYZ [in6-1] [1] ;
Y7=XYZ [in7-1] [1] ;
Y8=XYZ [in8-1] [1] ;
```

Y-Coordinates  
of 8 nodes

```
QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8) ;
```

```
Z1=XYZ [in1-1] [2] ;
Z2=XYZ [in2-1] [2] ;
Z3=XYZ [in3-1] [2] ;
Z4=XYZ [in4-1] [2] ;
Z5=XYZ [in5-1] [2] ;
Z6=XYZ [in6-1] [2] ;
Z7=XYZ [in7-1] [2] ;
Z8=XYZ [in8-1] [2] ;
```

Z-Coordinates  
of 8 nodes



Coordinates:  
Node ID - 1

```
JACOBI (DETJ, PNQ, PNE, PNT, PNQ, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8) ;
```

# MAT\_ASS\_MAIN (4/6)

```

nodLOCAL [0] = in1 ;
nodLOCAL [1] = in2 ;
nodLOCAL [2] = in3 ;
nodLOCAL [3] = in4 ;
nodLOCAL [4] = in5 ;
nodLOCAL [5] = in6 ;
nodLOCAL [6] = in7 ;
nodLOCAL [7] = in8 ;

```

```

X1=XYZ [in1-1] [0] ;
X2=XYZ [in2-1] [0] ;
X3=XYZ [in3-1] [0] ;
X4=XYZ [in4-1] [0] ;
X5=XYZ [in5-1] [0] ;
X6=XYZ [in6-1] [0] ;
X7=XYZ [in7-1] [0] ;
X8=XYZ [in8-1] [0] ;

```

X-Coordinates  
of 8 nodes

```

Y1=XYZ [in1-1] [1] ;
Y2=XYZ [in2-1] [1] ;
Y3=XYZ [in3-1] [1] ;
Y4=XYZ [in4-1] [1] ;
Y5=XYZ [in5-1] [1] ;
Y6=XYZ [in6-1] [1] ;
Y7=XYZ [in7-1] [1] ;
Y8=XYZ [in8-1] [1] ;

```

Y-Coordinates  
of 8 nodes

$QVC = 08th * (X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8) ;$

```

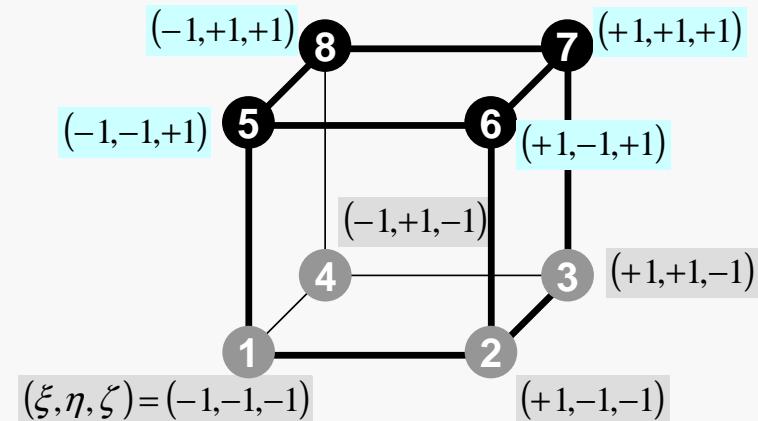
Z1=XYZ [in1-1] [2] ;
Z2=XYZ [in2-1] [2] ;
Z3=XYZ [in3-1] [2] ;
Z4=XYZ [in4-1] [2] ;
Z5=XYZ [in5-1] [2] ;
Z6=XYZ [in6-1] [2] ;
Z7=XYZ [in7-1] [2] ;
Z8=XYZ [in8-1] [2] ;

```

```

JACOBI (DETJ, PNQ, PNE, PNT, PNX, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8) ;

```



Coordinates:  
Node ID - 1

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

Heat Gen. Rate is a function of location  
(cell center:  $x_c, y_c$ )

# MAT\_ASS\_MAIN (4/6)

```

nodLOCAL [0] = in1 ;
nodLOCAL [1] = in2 ;
nodLOCAL [2] = in3 ;
nodLOCAL [3] = in4 ;
nodLOCAL [4] = in5 ;
nodLOCAL [5] = in6 ;
nodLOCAL [6] = in7 ;
nodLOCAL [7] = in8 ;

```

```

X1=XYZ[in1-1][0] ;
X2=XYZ[in2-1][0] ;
X3=XYZ[in3-1][0] ;
X4=XYZ[in4-1][0] ;
X5=XYZ[in5-1][0] ;
X6=XYZ[in6-1][0] ;
X7=XYZ[in7-1][0] ;
X8=XYZ[in8-1][0] ;

```

```

Y1=XYZ[in1-1][1] ;
Y2=XYZ[in2-1][1] ;
Y3=XYZ[in3-1][1] ;
Y4=XYZ[in4-1][1] ;
Y5=XYZ[in5-1][1] ;
Y6=XYZ[in6-1][1] ;
Y7=XYZ[in7-1][1] ;
Y8=XYZ[in8-1][1] ;

```

**QVC= 08th\*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8) ;**

```

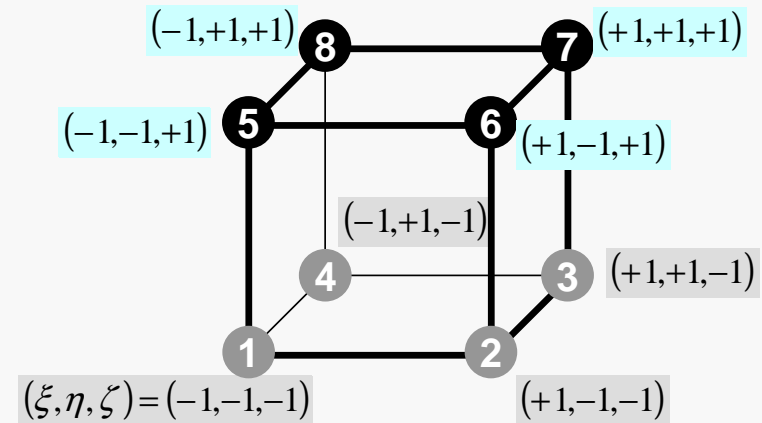
Z1=XYZ[in1-1][2] ;
Z2=XYZ[in2-1][2] ;
Z3=XYZ[in3-1][2] ;
Z4=XYZ[in4-1][2] ;
Z5=XYZ[in5-1][2] ;
Z6=XYZ[in6-1][2] ;
Z7=XYZ[in7-1][2] ;
Z8=XYZ[in8-1][2] ;

```

```

JACOBI (DETJ, PNQ, PNE, PNT, PNx, PNY, PNz,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8) ;

```



**Coordinates:  
Node ID - 1**

$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( \lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( \lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

$$QVC = |x_c + y_c|$$

# MAT\_ASS\_MAIN (4/6)

```
nodLOCAL [0] = in1 ;  
nodLOCAL [1] = in2 ;  
nodLOCAL [2] = in3 ;  
nodLOCAL [3] = in4 ;  
nodLOCAL [4] = in5 ;  
nodLOCAL [5] = in6 ;  
nodLOCAL [6] = in7 ;  
nodLOCAL [7] = in8 ;
```

```
X1=XYZ[in1-1][0] ;  
X2=XYZ[in2-1][0] ;  
X3=XYZ[in3-1][0] ;  
X4=XYZ[in4-1][0] ;  
X5=XYZ[in5-1][0] ;  
X6=XYZ[in6-1][0] ;  
X7=XYZ[in7-1][0] ;  
X8=XYZ[in8-1][0] ;
```

```
Y1=XYZ[in1-1][1] ;  
Y2=XYZ[in2-1][1] ;  
Y3=XYZ[in3-1][1] ;  
Y4=XYZ[in4-1][1] ;  
Y5=XYZ[in5-1][1] ;  
Y6=XYZ[in6-1][1] ;  
Y7=XYZ[in7-1][1] ;  
Y8=XYZ[in8-1][1] ;
```

```
QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8) ;
```

```
Z1=XYZ[in1-1][2] ;  
Z2=XYZ[in2-1][2] ;  
Z3=XYZ[in3-1][2] ;  
Z4=XYZ[in4-1][2] ;  
Z5=XYZ[in5-1][2] ;  
Z6=XYZ[in6-1][2] ;  
Z7=XYZ[in7-1][2] ;  
Z8=XYZ[in8-1][2] ;
```

```
JACOBI (DETJ, PNQ, PNE, PNT, PNX, PNY, PNZ,  
X1, X2, X3, X4, X5, X6, X7, X8,  
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8) ;
```

# JACOBI (1/4)

```

#include <stdio.h>
#include <math.h>
#include "precision.h"
#include "allocate.h"
/**
 *** JACOBI
 ***/
void JACOBI (
    KREAL DETJ[2][2][2],
    KREAL PNX[2][2][8], KREAL PNE[2][2][8], KREAL PNT[2][2][8],
    KREAL PNY[2][2][8], KREAL PNZ[2][2][8],
    KREAL X1, KREAL X2, KREAL X3, KREAL X4, KREAL X5, KREAL X6, KREAL X7, KREAL X8,
    KREAL Y1, KREAL Y2, KREAL Y3, KREAL Y4, KREAL Y5, KREAL Y6, KREAL Y7, KREAL Y8,
    KREAL Z1, KREAL Z2, KREAL Z3, KREAL Z4, KREAL Z5, KREAL Z6, KREAL Z7, KREAL Z8)
{
/**
 calculates JACOBIAN & INVERSE JACOBIAN
 dNi/dx, dNi/dy & dNi/dz
**/

    int ip, jp, kp;
    double dXdQ, dYdQ, dZdQ, dXdE, dYdE, dZdE, dXdT, dYdT, dZdT;
    double coef;
    double a11, a12, a13, a21, a22, a23, a31, a32, a33;

    for (ip=0; ip<2; ip++) {
        for (jp=0; jp<2; jp++) {
            for (kp=0; kp<2; kp++) {
                PNX[ip][jp][kp][0]=0.0;
                PNX[ip][jp][kp][1]=0.0;
                PNX[ip][jp][kp][2]=0.0;
                PNX[ip][jp][kp][3]=0.0;
                PNX[ip][jp][kp][4]=0.0;
                PNX[ip][jp][kp][5]=0.0;
                PNX[ip][jp][kp][6]=0.0;
                PNX[ip][jp][kp][7]=0.0;
            }
        }
    }
}

```

Input

$$\left[ \frac{\partial N_l}{\partial \xi}, \frac{\partial N_l}{\partial \eta}, \frac{\partial N_l}{\partial \zeta} \right], (x_l, y_l, z_l) (l = 1 \sim 8)$$

Output

$$\left[ \frac{\partial N_l}{\partial x}, \frac{\partial N_l}{\partial y}, \frac{\partial N_l}{\partial z} \right], \det|J|$$

**Values at each Gaussian Quad.  
Points: [ip][jp][kp]**

# Partial Diff. on Natural Coord. (1/4)

- According to formulae:

$$\frac{\partial N_i(\xi, \eta, \zeta)}{\partial \xi} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \xi} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \xi}$$

$$\frac{\partial N_i(\xi, \eta, \zeta)}{\partial \eta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \eta} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \eta}$$

$$\frac{\partial N_i(\xi, \eta, \zeta)}{\partial \zeta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \zeta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \zeta} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \zeta}$$

$\left[ \frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} \right]$  can be easily derived according to definitions.

$\left[ \frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} \right]$  are required for computations.

# Partial Diff. on Natural Coord. (2/4)

- In matrix form:

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = [J] \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix}$$

$$[J] = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}$$

$[J]$  : Jacobi matrix, Jacobian

# Partial Diff. on Natural Coord. (3/4)

- Components of Jacobian:

$$J_{11} = \frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^8 N_i x_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \xi} x_i, \quad J_{12} = \frac{\partial y}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^8 N_i y_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \xi} y_i,$$

$$J_{13} = \frac{\partial z}{\partial \xi} = \frac{\partial}{\partial \xi} \left( \sum_{i=1}^8 N_i z_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \xi} z_i$$

$$J_{21} = \frac{\partial x}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^8 N_i x_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \eta} x_i, \quad J_{22} = \frac{\partial y}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^8 N_i y_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \eta} y_i,$$

$$J_{23} = \frac{\partial z}{\partial \eta} = \frac{\partial}{\partial \eta} \left( \sum_{i=1}^8 N_i z_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \eta} z_i$$

$$J_{31} = \frac{\partial x}{\partial \zeta} = \frac{\partial}{\partial \zeta} \left( \sum_{i=1}^8 N_i x_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \zeta} x_i, \quad J_{32} = \frac{\partial y}{\partial \zeta} = \frac{\partial}{\partial \zeta} \left( \sum_{i=1}^8 N_i y_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \zeta} y_i,$$

$$J_{33} = \frac{\partial z}{\partial \zeta} = \frac{\partial}{\partial \zeta} \left( \sum_{i=1}^8 N_i z_i \right) = \sum_{i=1}^8 \frac{\partial N_i}{\partial \zeta} z_i$$



# JACOBI (2/4)

```

PNY [ip] [jp] [kp] [0]=0. 0;
PNY [ip] [jp] [kp] [1]=0. 0;
PNY [ip] [jp] [kp] [2]=0. 0;
PNY [ip] [jp] [kp] [3]=0. 0;
PNY [ip] [jp] [kp] [4]=0. 0;
PNY [ip] [jp] [kp] [5]=0. 0;
PNY [ip] [jp] [kp] [6]=0. 0;
PNY [ip] [jp] [kp] [7]=0. 0;

```

```

PNZ [ip] [jp] [kp] [0]=0. 0;
PNZ [ip] [jp] [kp] [1]=0. 0;
PNZ [ip] [jp] [kp] [2]=0. 0;
PNZ [ip] [jp] [kp] [3]=0. 0;
PNZ [ip] [jp] [kp] [4]=0. 0;
PNZ [ip] [jp] [kp] [5]=0. 0;
PNZ [ip] [jp] [kp] [6]=0. 0;
PNZ [ip] [jp] [kp] [7]=0. 0;

```

$$[J] = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}$$

/\*\*

DETERMINANT of the JACOBIAN

\*\*/

```

dXdQ = PNQ[jp][kp][0]*X1 + PNQ[jp][kp][1]*X2
      + PNQ[jp][kp][2]*X3 + PNQ[jp][kp][3]*X4
      + PNQ[jp][kp][4]*X5 + PNQ[jp][kp][5]*X6
      + PNQ[jp][kp][6]*X7 + PNQ[jp][kp][7]*X8;
dYdQ = PNQ[jp][kp][0]*Y1 + PNQ[jp][kp][1]*Y2
      + PNQ[jp][kp][2]*Y3 + PNQ[jp][kp][3]*Y4
      + PNQ[jp][kp][4]*Y5 + PNQ[jp][kp][5]*Y6
      + PNQ[jp][kp][6]*Y7 + PNQ[jp][kp][7]*Y8;
dZdQ = PNQ[jp][kp][0]*Z1 + PNQ[jp][kp][1]*Z2
      + PNQ[jp][kp][2]*Z3 + PNQ[jp][kp][3]*Z4
      + PNQ[jp][kp][4]*Z5 + PNQ[jp][kp][5]*Z6
      + PNQ[jp][kp][6]*Z7 + PNQ[jp][kp][7]*Z8;
dXdE = PNE[ip][kp][0]*X1 + PNE[ip][kp][1]*X2
      + PNE[ip][kp][2]*X3 + PNE[ip][kp][3]*X4
      + PNE[ip][kp][4]*X5 + PNE[ip][kp][5]*X6
      + PNE[ip][kp][6]*X7 + PNE[ip][kp][7]*X8;

```

$$dXdQ = \frac{\partial x}{\partial \xi} = J_{11}$$

$$dYdQ = \frac{\partial y}{\partial \xi} = J_{12}$$

$$dZdQ = \frac{\partial z}{\partial \xi} = J_{13}$$

# JACOBI (3/4)

```

dYdE = PNE[ip][kp][0]*Y1 + PNE[ip][kp][1]*Y2
        + PNE[ip][kp][2]*Y3 + PNE[ip][kp][3]*Y4
        + PNE[ip][kp][4]*Y5 + PNE[ip][kp][5]*Y6
        + PNE[ip][kp][6]*Y7 + PNE[ip][kp][7]*Y8;
dZdE = PNE[ip][kp][0]*Z1 + PNE[ip][kp][1]*Z2
        + PNE[ip][kp][2]*Z3 + PNE[ip][kp][3]*Z4
        + PNE[ip][kp][4]*Z5 + PNE[ip][kp][5]*Z6
        + PNE[ip][kp][6]*Z7 + PNE[ip][kp][7]*Z8;
dXdT = PNT[ip][jp][0]*X1 + PNT[ip][jp][1]*X2
        + PNT[ip][jp][2]*X3 + PNT[ip][jp][3]*X4
        + PNT[ip][jp][4]*X5 + PNT[ip][jp][5]*X6
        + PNT[ip][jp][6]*X7 + PNT[ip][jp][7]*X8;
dYdT = PNT[ip][jp][0]*Y1 + PNT[ip][jp][1]*Y2
        + PNT[ip][jp][2]*Y3 + PNT[ip][jp][3]*Y4
        + PNT[ip][jp][4]*Y5 + PNT[ip][jp][5]*Y6
        + PNT[ip][jp][6]*Y7 + PNT[ip][jp][7]*Y8;
dZdT = PNT[ip][jp][0]*Z1 + PNT[ip][jp][1]*Z2
        + PNT[ip][jp][2]*Z3 + PNT[ip][jp][3]*Z4
        + PNT[ip][jp][4]*Z5 + PNT[ip][jp][5]*Z6
        + PNT[ip][jp][6]*Z7 + PNT[ip][jp][7]*Z8;

```

```

DETJ[ip][jp][kp]= dXdQ*(dYdE*dZdT-dZdE*dYdT) +
                  dYdQ*(dZdE*dXdT-dXdE*dZdT) +
                  dZdQ*(dXdE*dYdT-dYdE*dXdT);

```

```

/**
INVERSE JACOBIAN
**/

```

```

coef=1.0 / DETJ[ip][jp][kp];

```

```

a11= coef * ( dYdE*dZdT - dZdE*dYdT );
a12= coef * ( dZdQ*dYdT - dYdQ*dZdT );
a13= coef * ( dYdQ*dZdE - dZdQ*dYdE );

```

```

a21= coef * ( dZdE*dXdT - dXdE*dZdT );
a22= coef * ( dXdQ*dZdT - dZdQ*dXdT );
a23= coef * ( dZdQ*dXdE - dXdQ*dZdE );

```

```

a31= coef * ( dXdE*dYdT - dYdE*dXdT );
a32= coef * ( dYdQ*dXdT - dXdQ*dYdT );
a33= coef * ( dXdQ*dYdE - dYdQ*dXdE );

```

```

DETJ[ip][jp][kp]=fabs(DETJ[ip][jp][kp]);

```

$$[J] = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}$$

# Partial Diff. on Natural Coord. (4/4)

- Partial differentiation on global coordinate system is introduced as follows (with inverse of Jacobian matrix ( $3 \times 3$ ))

$$\left\{ \begin{array}{c} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{array} \right\} = \left[ \begin{array}{ccc} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{array} \right]^{-1} \left\{ \begin{array}{c} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{array} \right\} = [J]^{-1} \left\{ \begin{array}{c} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{array} \right\}$$

# JACOBI (3/4)

```

dYdE = PNE[ip][kp][0]*Y1 + PNE[ip][kp][1]*Y2
        + PNE[ip][kp][2]*Y3 + PNE[ip][kp][3]*Y4
        + PNE[ip][kp][4]*Y5 + PNE[ip][kp][5]*Y6
        + PNE[ip][kp][6]*Y7 + PNE[ip][kp][7]*Y8;
dZdE = PNE[ip][kp][0]*Z1 + PNE[ip][kp][1]*Z2
        + PNE[ip][kp][2]*Z3 + PNE[ip][kp][3]*Z4
        + PNE[ip][kp][4]*Z5 + PNE[ip][kp][5]*Z6
        + PNE[ip][kp][6]*Z7 + PNE[ip][kp][7]*Z8;
dXdT = PNT[ip][jp][0]*X1 + PNT[ip][jp][1]*X2
        + PNT[ip][jp][2]*X3 + PNT[ip][jp][3]*X4
        + PNT[ip][jp][4]*X5 + PNT[ip][jp][5]*X6
        + PNT[ip][jp][6]*X7 + PNT[ip][jp][7]*X8;
dYdT = PNT[ip][jp][0]*Y1 + PNT[ip][jp][1]*Y2
        + PNT[ip][jp][2]*Y3 + PNT[ip][jp][3]*Y4
        + PNT[ip][jp][4]*Y5 + PNT[ip][jp][5]*Y6
        + PNT[ip][jp][6]*Y7 + PNT[ip][jp][7]*Y8;
dZdT = PNT[ip][jp][0]*Z1 + PNT[ip][jp][1]*Z2
        + PNT[ip][jp][2]*Z3 + PNT[ip][jp][3]*Z4
        + PNT[ip][jp][4]*Z5 + PNT[ip][jp][5]*Z6
        + PNT[ip][jp][6]*Z7 + PNT[ip][jp][7]*Z8;
DETJ[ip][jp][kp]= dXdQ*(dYdE*dZdT-dZdE*dYdT) +
                  dYdQ*(dZdE*dXdT-dXdE*dZdT) +
                  dZdQ*(dXdE*dYdT-dYdE*dXdT);

```

```

/**
INVERSE JACOBIAN
**/

```

```
coef=1.0 / DETJ[ip][jp][kp];
```

```

a11= coef * ( dYdE*dZdT - dZdE*dYdT );
a12= coef * ( dZdQ*dYdT - dYdQ*dZdT );
a13= coef * ( dYdQ*dZdE - dZdQ*dYdE );

```

```

a21= coef * ( dZdE*dXdT - dXdE*dZdT );
a22= coef * ( dXdQ*dZdT - dZdQ*dXdT );
a23= coef * ( dZdQ*dXdE - dXdQ*dZdE );

```

```

a31= coef * ( dXdE*dYdT - dYdE*dXdT );
a32= coef * ( dYdQ*dXdT - dXdQ*dYdT );
a33= coef * ( dXdQ*dYdE - dYdQ*dXdE );

```

```
DETJ[ip][jp][kp]=fabs(DETJ[ip][jp][kp]);
```

$$[J]^{-1} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$



# MAT\_ASS\_MAIN (5/6)

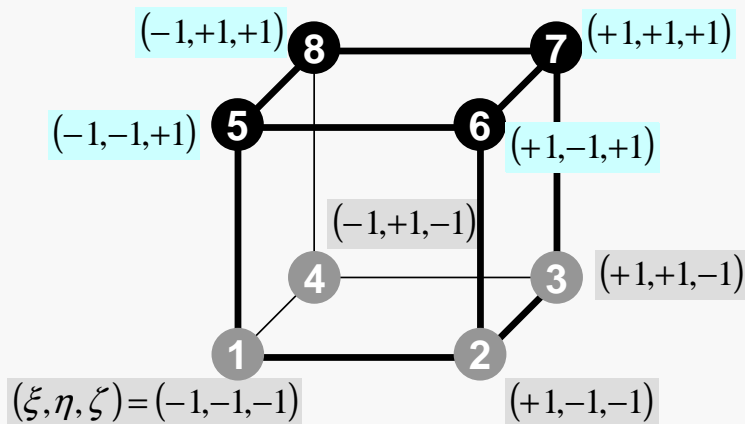
```

/**
CONSTRUCT the GLOBAL MATRIX
**/
for (ie=0; ie<8; ie++) {
  ip=nodLOCAL[ie];

  for (je=0; je<8; je++) {
    jp=nodLOCAL[je];

    kk=-1;
    if ( jp != ip ) {
      iiS=indexLU[ip-1];
      iiE=indexLU[ip ];
      for ( k=iiS; k<iiE; k++) {
        if ( itemLU[k] == jp-1 ) {
          kk=k;
          break;
        }
      }
    }
  }
}

```



Non-Zero Off-Diagonal Block  
in Global Matix

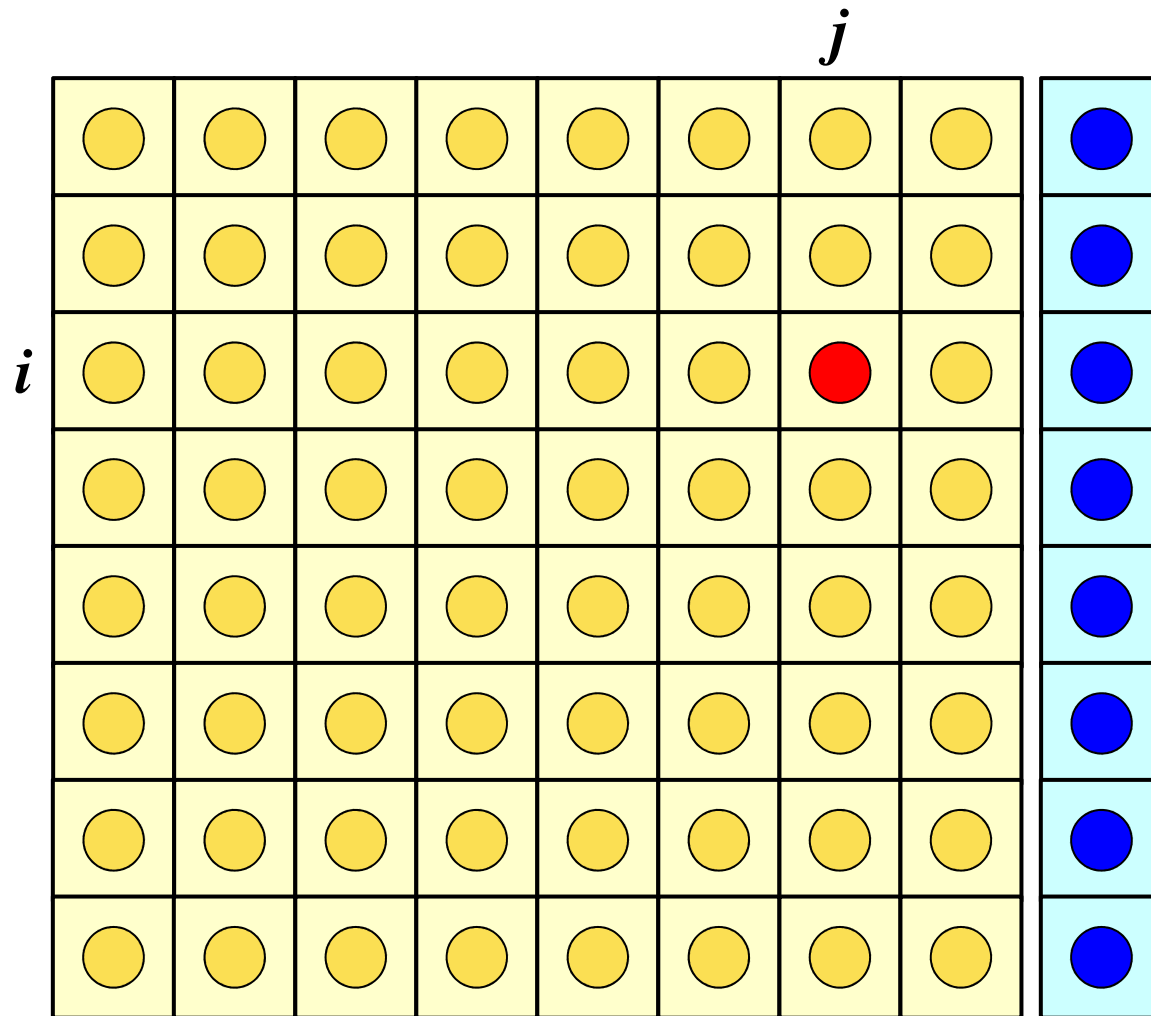
$$A_{ip, jp}$$

kk: address in "itemLU"

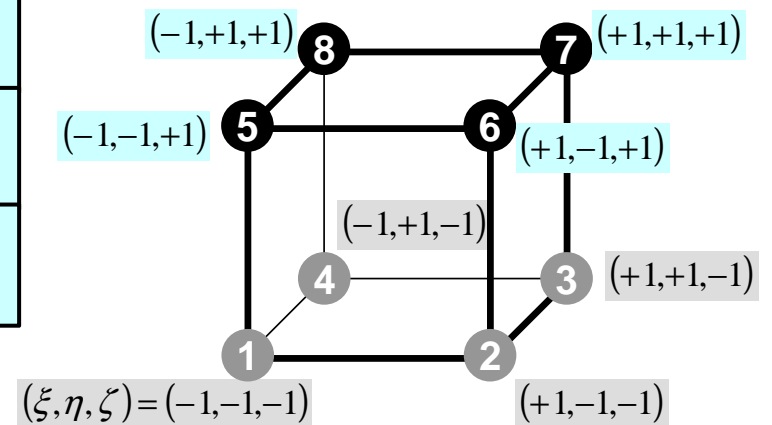
ip= nodLOCAL[ie]  
jp= nodLOCAL[je]

Node ID (ip,jp)  
starting from 1

# Element Matrix: 8x8



$$[k_{ij}] \quad (i, j = 1 \dots 8)$$



# MAT\_ASS\_MAIN (5/6)

```

/**
CONSTRUCT the GLOBAL MATRIX
**/

for (ie=0; ie<8; ie++) {
  ip=nodLOCAL[ie];

  for (je=0; je<8; je++) {
    jp=nodLOCAL[je];

    kk=-1;
    if ( jp != ip ) {
      iiS=indexLU[ip-1];
      iiE=indexLU[ip ];
      for ( k=iiS; k<iiE; k++) {
        if ( itemLU[k] == jp-1 ) {
          kk=k;
          break;
        }
      }
    }
  }
}

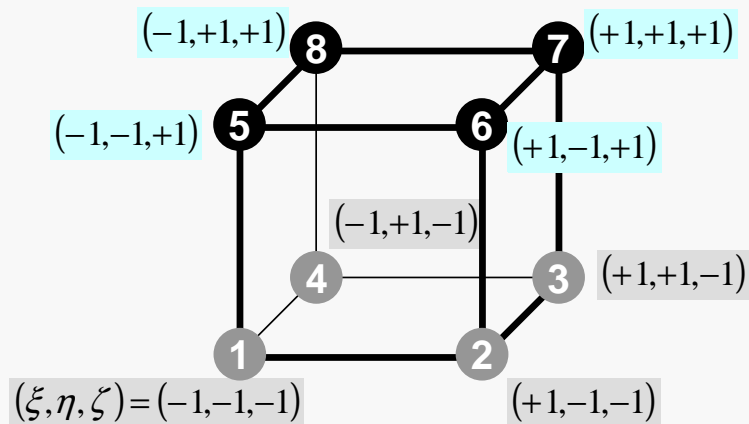
```

Element Matrix ( $i_e \sim j_e$ ): Local ID  
 Global Matrix ( $i_p \sim j_p$ ): Global ID

kk: address in "itemLU" starting from "0"

k: starting from "0"

ip,jp: starting from "1"





# MAT\_ASS\_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0;kpn<2;kpn++) {
  for (jpn=0;jpn<2;jpn++) {
    for (ipn=0;ipn<2;ipn++) {
      coef= fabs (DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;
    }
  }
}

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}
if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}
}

```

$$\int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det|J| d\xi d\eta d\zeta$$

# MAT\_ASS\_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0; kpn<2; kpn++) {
  for (jpn=0; jpn<2; jpn++) {
    for (ipn=0; ipn<2; ipn++) {
      coef= fabs (DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;
    }
  }
}

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}
if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}

```

$$\begin{aligned}
 I &= \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta, \zeta) d\xi d\eta d\zeta \\
 &= \sum_{i=1}^L \sum_{j=1}^M \sum_{k=1}^N W_i \cdot W_j \cdot W_k \cdot f(\xi_i, \eta_j, \zeta_k)
 \end{aligned}$$

$$\int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det|J| d\xi d\eta d\zeta$$

# MAT\_ASS\_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0; kpn<2; kpn++) {
  for (jpn=0; jpn<2; jpn++) {
    for (ipn=0; ipn<2; ipn++) {
      coef= fabs (DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

```

```

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

```

$$\text{coef} = W_i \cdot W_j \cdot W_k \cdot \det|J(\xi_i, \eta_j, \zeta_k)|$$

```

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

```

```

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

```

```

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;

```

```

    }
  }
}

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}
if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}

```

$$\begin{aligned}
 I &= \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta, \zeta) d\xi d\eta d\zeta \\
 &= \sum_{i=1}^L \sum_{j=1}^M \sum_{k=1}^N W_i \cdot W_j \cdot W_k \cdot f(\xi_i, \eta_j, \zeta_k)
 \end{aligned}$$

$$\int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det|J| d\xi d\eta d\zeta$$



# MAT\_ASS\_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0; kpn<2; kpn++) {
  for (jpn=0; jpn<2; jpn++) {
    for (ipn=0; ipn<2; ipn++) {
      coef= fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;
    }
  }
}

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}
if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}

```

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)}$$

$$\{f\}^{(e)} = \int_V \dot{Q}[N]^T dV$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

$$QVC = |x_c + y_c|$$

$$QV0 = \int_V QVOL [N]^T dV$$

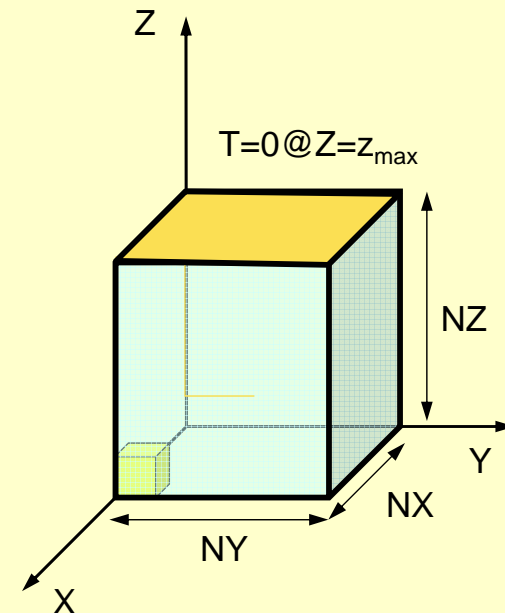
$$\{f\}^{(e)} = QV0 \cdot QVC$$

# MAT\_ASS\_BC: Overview

```
do i= 1, N    Loop for Nodes
  "Mark" nodes where Dirichlet B.C. are applied (IWKX)
enddo
```

```
do i= 1, N    Loop for Nodes
  if (IWKX(i,1).eq.1) then  if "marked" nodes
    corresponding components of RHS (B),
    Diagonal (D) are corrected
    do k= indexLU(i-1)+1, indexLU(i)  Non-Zero Off-Diagonal Nodes
      corresponding comp. of non-zero off-diagonal
      components (AMAT) are corrected
    enddo
  endif
enddo
```

```
do i= 1, N    Loop for Nodes
  do k= indexLU(i-1)+1, indexLU(i)  Non-Zero Off-Diagonal Nodes
    if (IWKX(itemLU(k),1).eq.1) then  if corresponding non-zero
      off-diagonal node is "marked"
      corresponding components of RHS and AMAT are corrected (col.)
    endif
  enddo
enddo
```



# MAT\_ASS\_BC (1/2)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
void MAT_ASS_BC()
{
    int i, j, k, in, ib, ib0, icel;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    int iq1, iq2, iq3, iq4, iq5, iq6, iq7, iq8;
    int iS, iE;
    double STRESS, VAL;

    IWKX=(KINT**) allocate_matrix(sizeof(KINT), N, 2);
    for (i=0; i<N; i++) for (j=0; j<2; j++) IWKX[i][j]=0;

    /**
     * Z=Zmax
     */

    for (in=0; in<N; in++) IWKX[in][0]=0;

    ib0=-1;

    for ( ib0=0; ib0<NODGRPtot; ib0++) {
        if ( strcmp (NODGRP_NAME[ib0].name, "Zmax") == 0 ) break;
    }

    for ( ib=NODGRP_INDEX[ib0]; ib<NODGRP_INDEX[ib0+1]; ib++) {
        in=NODGRP_ITEM[ib];
        IWKX[in-1][0]=1;
    }
}

```

If the node "in" is included in the node group "Zmax"

$$IWKX[in-1][0] = 1$$

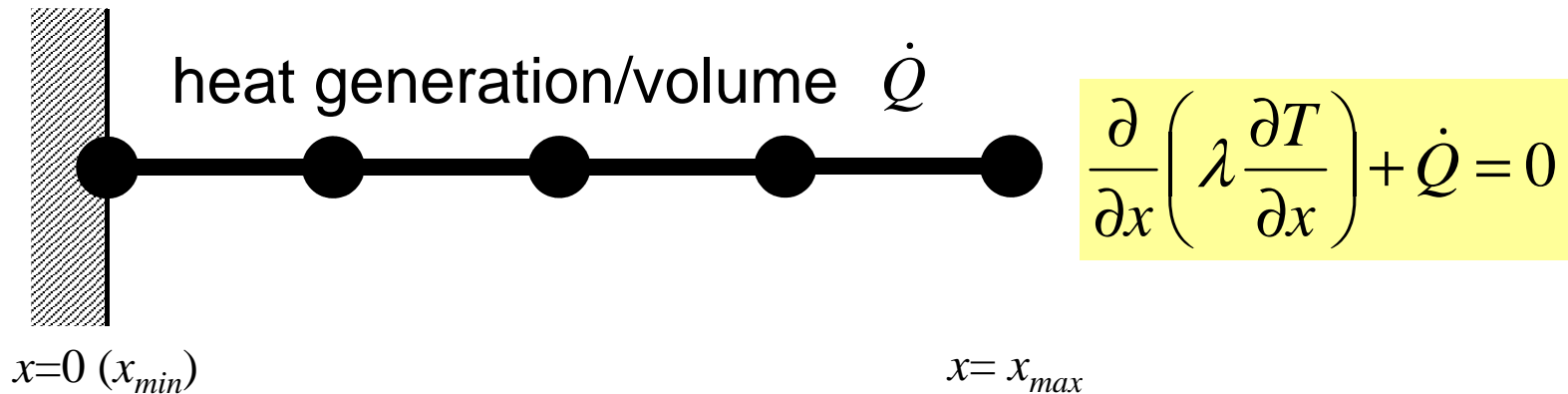
# MAT\_ASS\_BC (2/2)

```
for(in=0; in<N; in++) {
  if( IWKX[in][0] == 1 ) {
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++) {
      AMAT[k]= 0. e0;
    }
  }
}

for(in=0; in<N; in++) {
  for(k=indexLU[in]; k<indexLU[in+1]; k++) {
    if (IWKX[itemLU[k]][0] == 1 ) {
      AMAT[k]= 0. e0;
    }
  }
}
}
```



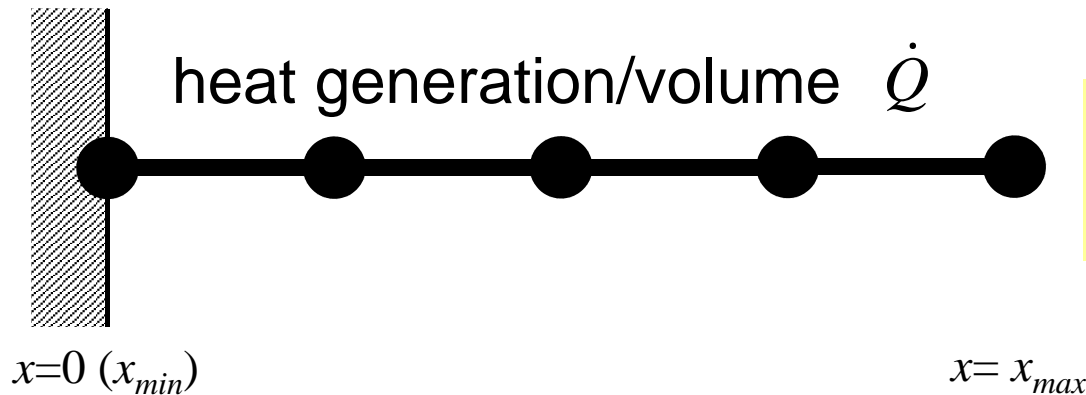
# 1D Steady State Heat Conduction



- **Uniform: Sectional Area:  $A$ , Thermal Conductivity:  $\lambda$**
- Heat Generation Rate/Volume/Time [ $QL^{-3}T^{-1}$ ]  $\dot{Q}$
- Boundary Conditions
  - $x=0$  :  $T=0$  (Fixed Temperature)
  - $x=x_{max}$  :  $\frac{\partial T}{\partial x} = 0$  (Insulated)

# (Linear) Equation at $x=0$

$$T_1 = 0 \text{ (or } T_0 = 0)$$



$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \dot{Q} = 0$$

- **Uniform: Sectional Area:  $A$ , Thermal Conductivity:  $\lambda$**
- Heat Generation Rate/Volume/Time [ $QL^{-3}T^{-1}$ ]  $\dot{Q}$
- Boundary Conditions
  - $x=0$  :  $T=0$  (Fixed Temperature)
  - $x=x_{max}$  :  $\frac{\partial T}{\partial x} = 0$  (Insulated)

# Program: 1d.c (6/6)

## Dirichlet B.C. @ X=0

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= 0.0;

  for (k=0;k<NPLU;k++) {
    if (Item[k]==0) {AMat[k]=0.0;
    }}

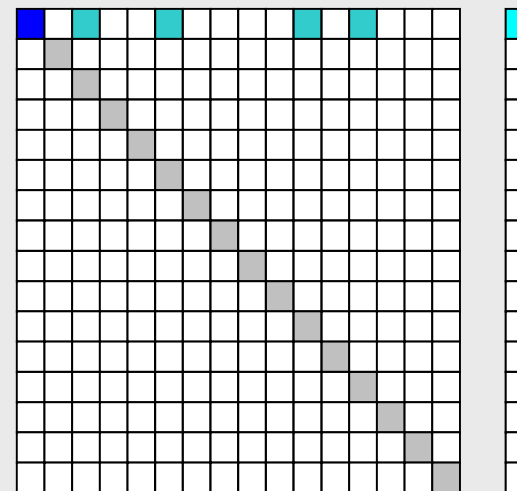
```

$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.



# Program: 1d.c (6/6)

## Dirichlet B.C. @ X=0

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= 0.0;

  for (k=0;k<NPLU;k++) {
    if (Item[k]==0) {AMat[k]=0.0;
    }}

```

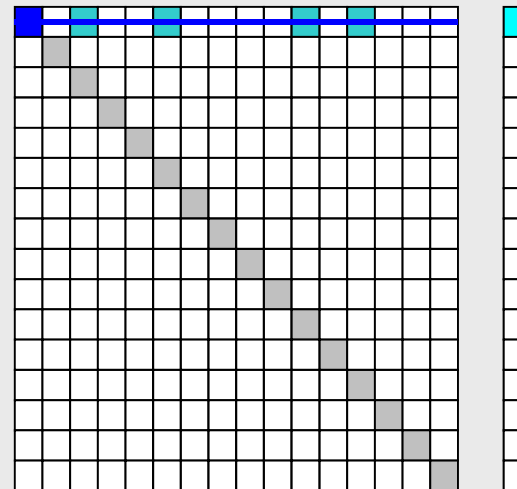
$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.

Erase !



# Program: 1d.c (6/6)

## Dirichlet B.C. @ X=0

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* X=Xmin */
i=0;
jS= Index[i];
AMat[jS]= 0.0;
Diag[i ]= 1.0;
Rhs [i ]= 0.0;

for (k=0;k<NPLU;k++) {
  if (Item[k]==0) {AMat[k]=0.0;
  }}

```

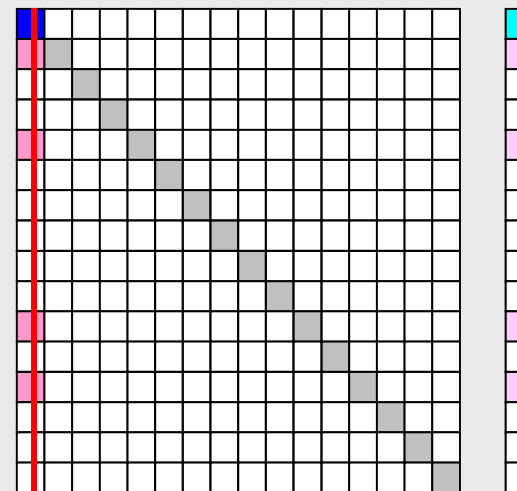
$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.

### Elimination and Erase



Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix (in this case just erase off-diagonal components)

1D

# if $T_1 \neq 0$

```
/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/
```

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

```
/* X=Xmin */
i=0;
jS= Index[i];
AMat[jS]= 0.0;
Diag[i ]= 1.0;
Rhs [i ]= PHImin;
```

$$Diag_j \phi_j + \sum_{k=Index[j]}^{Index[j+1]-1} Amat_k \phi_{Item[k]} = Rhs_j$$

```
for (j=1; i<N; i++) {
  for (k=Index[j]; k<Index[j+1]; k++) {
    if (Item[k]==0) {
      Rhs [j]= Rhs[j] - Amat[k]*PHImin;
      AMat[k]= 0.0;
    }
  }
}
```

if  $T_1 \neq 0$

```
/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/
```

```
/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= PHImin;
```

```
  for (j=1; i<N; i++) {
    for (k=Index[j]; k<Index[j+1]; k++) {
      if (Item[k]==0) {
        Rhs [j]= Rhs[j] - Amat[k]*PHImin;
        AMat[k]= 0.0;
      }
    }
  }
```

$$\text{Diag}_j \phi_j + \sum_{k=\text{Index}[j], k \neq k_s}^{\text{Index}[j+1]-1} \text{Amat}_k \phi_{\text{Item}[k]}$$

$$= \text{Rhs}_j - \text{Amat}_{k_s} \phi_{\text{Item}[k_s]}$$

$$= \text{Rhs}_j - \text{Amat}_{k_s} \phi_{\min} \quad \text{where } \text{Item}[k_s] = 0$$

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

# MAT\_ASS\_BC (2/2)

```

for(in=0; in<N; in++) {
  if( IWKX[in][0] == 1 ) {
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++) {
      AMAT[k]= 0. e0;
    }
  }
}

```

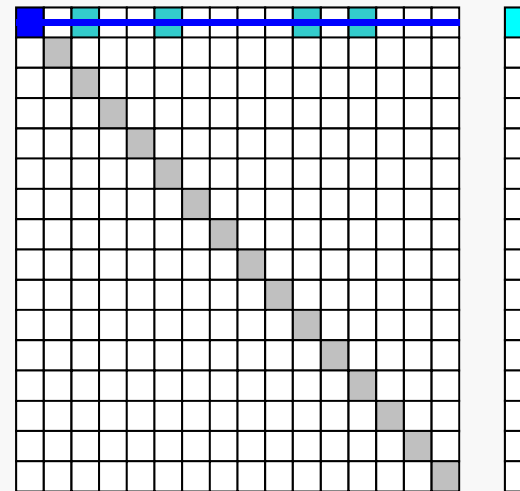
Boundary Nodes: IWKX[in-1][0]=1

```

for(in=0; in<N; in++) {
  for(k=indexLU[in]; k<indexLU[in+1]; k++) {
    if ( IWKX[itemLU[k]][0] == 1 ) {
      AMAT[k]= 0. e0;
    }
  }
}
}

```

Erase !!



Same as 1D case



# MAT\_ASS\_BC (2/2)

```

for(in=0; in<N; in++) {
  if( IWKX[in][0] == 1 ) {
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++) {
      AMAT[k]= 0. e0;
    }
  }
}

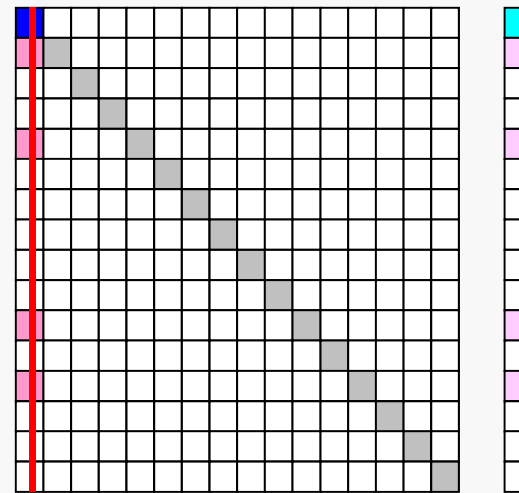
```

```

for(in=0; in<N; in++) {
  for(k=indexLU[in]; k<indexLU[in+1]; k++) {
    if (IWKX[itemLU[k]][0] == 1 ) {
      AMAT[k]= 0. e0;
    }
  }
}

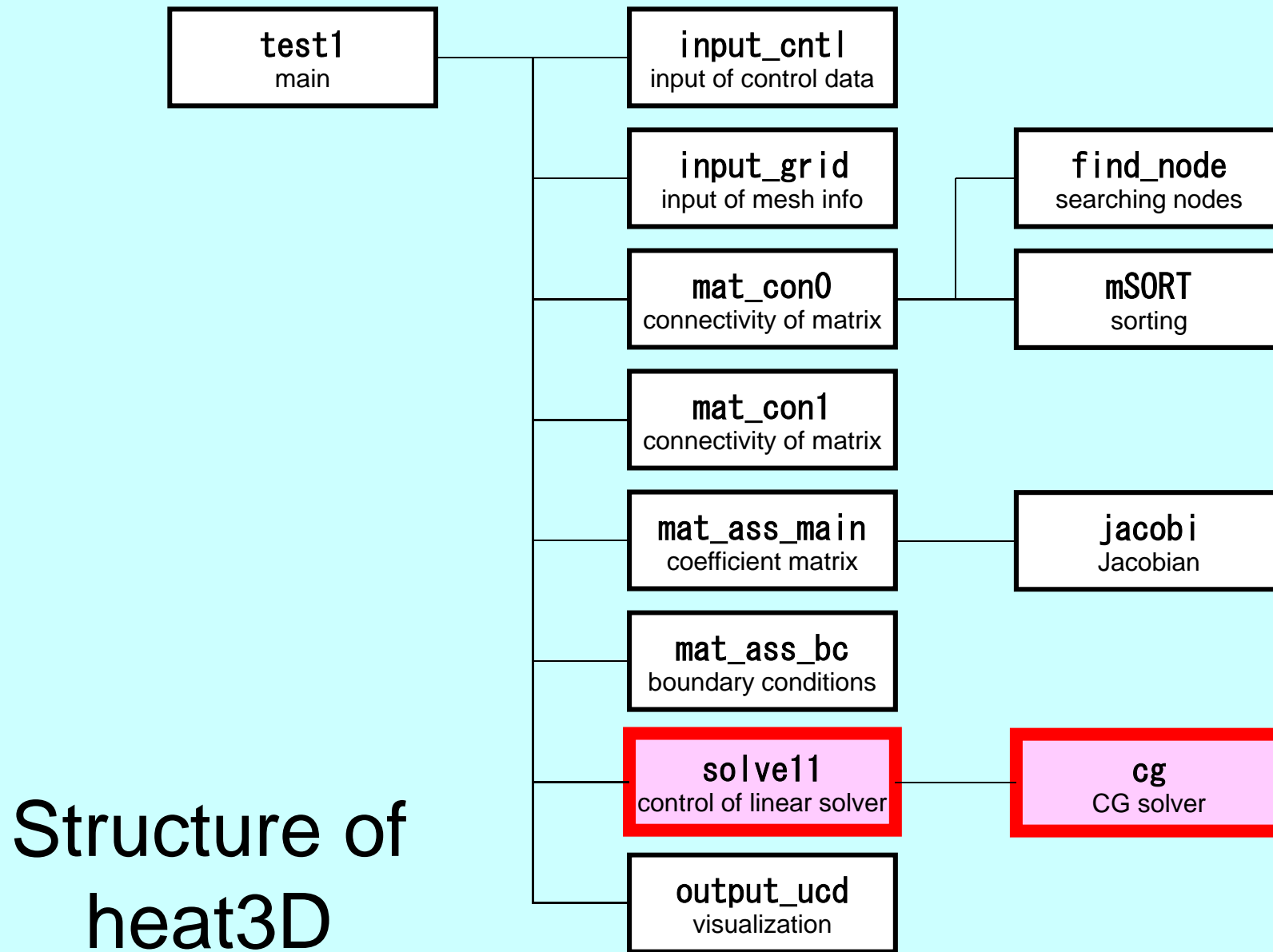
```

Boundary Nodes:  $IWKX[in-1][0]=1$



Elimination and Erase

Same as 1D case



# Main Part

```
/**
    program heat3D
**/
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
// #include "solver11.h"
extern void INPUT_CNTL ();
extern void INPUT_GRID ();
extern void MAT_CONO ();
extern void MAT_CON1 ();
extern void MAT_ASS_MAIN ();
extern void MAT_ASS_BC ();
extern void SOLVE11 ();
extern void OUTPUT_UCD ();
int main()
{
    INPUT_CNTL ();
    INPUT_GRID ();

    MAT_CONO ();
    MAT_CON1 ();

    MAT_ASS_MAIN ();
    MAT_ASS_BC ();

    SOLVE11 ();

    OUTPUT_UCD ();
}
```

# SOLVE11

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void CG();
void SOLVE11()
{
    int i, j, k, ii, L;

    int ERROR, ICFLAG=0;
    CHAR_LENGTH BUF;

    /**
     +-----+
     | PARAMETERS |
     +-----+
    **/
    ITER      = pfemIarray[0];      Number of Iterations for CG
    RESID     = pfemRarray[0];      Convergence Criteria for CG

    /**
     +-----+
     | ITERATIVE solver |
     +-----+
    **/
    CG (N, NPLU, D, AMAT, indexLU, itemLU, B, X, RESID, ITER, &ERROR);
    ITERactual= ITER;
}

```

# Preconditioned CG Solver

## Diagonal Scaling/Point Jacobi Preconditioning

```

Compute  $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$ 
for i= 1, 2, ...
  solve  $[\mathbf{M}]\mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$ 
   $\rho_{i-1} = \mathbf{r}^{(i-1)} \mathbf{z}^{(i-1)}$ 
  if i=1
     $\mathbf{p}^{(1)} = \mathbf{z}^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $\mathbf{p}^{(i)} = \mathbf{z}^{(i-1)} + \beta_{i-1} \mathbf{p}^{(i-1)}$ 
  endif
   $\mathbf{q}^{(i)} = [\mathbf{A}]\mathbf{p}^{(i)}$ 
   $\alpha_i = \rho_{i-1} / \mathbf{p}^{(i)} \mathbf{q}^{(i)}$ 
   $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$ 
   $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)}$ 
  check convergence  $|\mathbf{r}|$ 
end

```

$$[\mathbf{M}] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

# Diagonal Scaling, Point-Jacobi

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- **solve**  $[M] \mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$  is very easy.
- Provides fast convergence for simple problems.

# CG Solver (1/6)

```
#include <stdio.h>
#include <math.h>
#include "precision.h"
#include "allocate.h"
extern FILE *fp_log;

void CG (
    KINT N, KINT NPLU, KREAL D[],
    KREAL AMAT[], KINT indexLU[], KINT itemLU[],
    KREAL B[], KREAL X[], KREAL RESID, KINT ITER, KINT *ERROR)
{
    int i, j, k;
    int ieL, isL, ieU, isU;
    double WVAL;
    double BNRM20, BNRM2, DNRM20, DNRM2;
    double S1_TIME, E1_TIME;
    double ALPHA, BETA;
    double C1, C10, RHO, RH00, RH01;
    int    iterPRE;

    KREAL **WW;

    KINT R=0, Z=1, Q=1, P=2, DD=3;
    KINT MAXIT;
    KREAL TOL;
```

# CG Solver (1/6)

```

#include <stdio.h>
#include <math.h>
# WW[i][0] = WW[i][R] => {r}
# WW[i][1] = WW[i][Z] => {z}
e WW[i][1] = WW[i][Q] => {q}
v WW[i][2] = WW[i][P] => {p}
WW[i][3] = WW[i][DD] => 1/{D}
{
  int i, j, k;
  int ieL, isL, ieU, isU;
  double WVAL;
  double BNRM20, BNRM2, DNRM20, DNRM2;
  double S1_TIME, E1_TIME;
  double ALPHA, BETA;
  double C1, C10, RH0, RH00, RH01;
  int iterPRE;

  KREAL **WW;

  KINT R=0, Z=1, Q=1, P=2, DD=3;
  KINT MAXIT;
  KREAL TOL;

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if i=1
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```



# CG Solver (1/6)

```

WW=(KREAL**) allocate_matrix(sizeof(KREAL), 4, N);

MAXIT = ITER;
TOL   = RESID;

for (i=0; i<N; i++) {
    X[i]=0.0;
}
for (i=0; i<N; i++) for (j=0; j<4; j++) WW[j][i]=0.0;
/**
+-----+
| {r0} = {b} - [A]{xini} |
+-----+
**/
for (j=0; j<N; j++) {
    WW[DD][j] = 1.0/D[j];
    WVAL = B[j] - D[j]*X[j];

    for (k=indexLU[j]; k<indexLU[j+1]; k++) {
        i = itemLU[k];
        WVAL += -AMAT[k]*X[i];
    }
    WW[R][j] = WVAL;
}

```

$$\begin{aligned}
 WW[i][0] &= WW[i][R] \Rightarrow \{r\} \\
 WW[i][1] &= WW[i][Z] \Rightarrow \{z\} \\
 WW[i][1] &= WW[i][Q] \Rightarrow \{q\} \\
 WW[i][2] &= WW[i][P] \Rightarrow \{p\} \\
 WW[i][3] &= WW[i][DD] \Rightarrow 1/\{D\}
 \end{aligned}$$

Reciprocal numbers (倒数) of diagonal components are stored in  $WW[DD][i]$ . Computational cost for division is usually expensive.

# CG Solver (2/6)

```

WW=(KREAL**) allocate_matrix(sizeof(KREAL), 4, N);

MAXIT = ITER;
TOL   = RESID;

for (i=0; i<N; i++) {
    X[i]=0.0;
}
for (i=0; i<N; i++) for (j=0; j<4; j++) WW[j][i]=0
/**
+-----+
| {r0} = {b} - [A]{xini} |
+-----+
**/
for (j=0; j<N; j++) {
    WW[DD][j] = 1.0/D[j];
    WVAL = B[j] - D[j]*X[j];

    for (k=indexLU[j]; k<indexLU[j+1]; k++) {
        i = itemLU[k];
        WVAL += -AMAT[k]*X[i];
    }
    WW[R][j] = WVAL;
}

```

**Compute  $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$**

```

for i = 1, 2, ...
    solve [M]z(i-1) = r(i-1)
    ρi-1 = r(i-1) z(i-1)
    if i = 1
        p(1) = z(0)
    else
        βi-1 = ρi-1/ρi-2
        p(i) = z(i-1) + βi-1 p(i-1)
    endif
    q(i) = [A]p(i)
    αi = ρi-1/p(i) q(i)
    x(i) = x(i-1) + αip(i)
    r(i) = r(i-1) - αiq(i)
    check convergence |r|
end

```

# CG Solver (3/6)

```
BNRM2= 0. e0;
for (i=0; i<N; i++) {
  BNRM2+= B[i]*B[i];
}
```

```
BNRM2= BNRM2;
```

```
if (BNRM2 == 0. e0) BNRM2= 1. e0;
```

```
ITER = 0;
```

```
for( ITER=1; ITER<= MAXIT; ITER++) {
```

```
/**
```

```
***** Conjugate Gradient Iteration
```

```
**/
```

```
/**
```

```
+-----+
| {z} = [Minv] {r} |
+-----+
```

```
**/
```

```
for (i=0; i<N; i++) {
  WW[Z][i]= WW[DD][i]*WW[R][i];
}
```

**$BNRM2 = |b|^2$   
for convergence criteria  
of CG solvers**

# CG Solver (3/6)

```

    BNRM20= 0. e0;
  for (i=0; i<N; i++) {
    BNRM20+= B[i]*B[i];
  }

  BNRM2= BNRM20;

  if (BNRM2 == 0. e0) BNRM2= 1. e0;

  ITER = 0;

  for( ITER=1; ITER<= MAXIT; ITER++) {
/**
*****
**/

/**
+-----+
| {z}= [Minv] {r} |
+-----+
**/
    for (i=0; i<N; i++) {
      WW[Z][i]= WW[DD][i]*WW[R][i];
    }

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if  $i=1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

# CG Solver (4/6)

```

/**
+-----+
| {RHO}= {r} {z} |
+-----+
**/
RH00= 0. e0;

for(i=0;i<N;i++){
  RH00+= WW[R][i]*WW[Z][i];
}
RHO= RH00;
/**
+-----+
| {p} = {z} if      ITER=1 |
| BETA= RHO / RH01 otherwise |
+-----+
**/
if( ITER == 1 ){
  for(i=0;i<N;i++){
    WW[P][i]=WW[Z][i];
  }
}else{
  BETA= RHO / RH01;
  for(i=0;i<N;i++){
    WW[P][i]=WW[Z][i] + BETA*WW[P][i];
  }
}

```

Compute  $r^{(0)} = b - [A]x^{(0)}$   
 for  $i = 1, 2, \dots$   
 solve  $[M]z^{(i-1)} = r^{(i-1)}$   
 $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$   
 if  $i=1$   
 $p^{(1)} = z^{(0)}$   
 else  
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$   
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$   
 endif  
 $q^{(i)} = [A]p^{(i)}$   
 $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$   
 $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$   
 $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$   
 check convergence  $|r|$   
 end

# CG Solver (5/6)

```

/**
+-----+
| {q} = [A] {p} |
+-----+
**/
for ( j=0; j<N; j++) {
    WVAL= D[j] * WW[P][j];
    for (k=indexLU[j]; k<indexLU[j+1]; k++) {
        i=itemLU[k];
        WVAL+= AMAT[k] * WW[P][i];
    }
    WW[Q][j]=WVAL;
}

/**
+-----+
| ALPHA= RHO / {p} {q} |
+-----+
**/
C10= 0. e0;
for (i=0; i<N; i++) {
    C10+=WW[P][i]*WW[Q][i];
}
C1=C10;

ALPHA= RHO / C1;

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for for  $i = 1, 2, \dots$ 
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if  $i=1$ 
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence  $|r|$ 
end

```

# CG Solver (6/6)

```

/**
+-----+
| {x} = {x} + ALPHA*{p} |
| {r} = {r} - ALPHA*{q} |
+-----+
**/
for (i=0; i<N; i++) {
    X [i] += ALPHA *WW[P][i];
    WW[R][i] += -ALPHA *WW[Q][i];
}

DNRM2= 0. e0;
for (i=0; i<N; i++) {
    DNRM2+=WW[R][i]*WW[R][i];
}
DNRM2= DNRM2;
RESID= sqrt(DNRM2/BNRM2);

if ( RESID <= TOL ) break;
if ( ITER == MAXIT ) *ERROR= -300;

RH01 = RHO ;
}

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if  $i=1$ 
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence  $|r|$ 
end

```

# CG Solver (6/6)

```

/**
+-----+
| {x} = {x} + ALPHA*{p} |
| {r} = {r} - ALPHA*{q} |
+-----+
**/
for (i=0; i<N; i++) {
  X [i] += ALPHA *WW[P][i];
  WW[R][i] += -ALPHA *WW[Q][i];
}

DNRM2= 0. e0;
for (i=0; i<N; i++) {
  DNRM2+=WW[R][i]*WW[R][i];
}
DNRM2= DNRM2;
RESID= sqrt(DNRM2/BNRM2);

if ( RESID <= TOL ) break;
if ( ITER == MAXIT ) *ERROR= -300;

RH01 = RHO ;
}

```

Compute  $r^{(0)} = b - [A]x^{(0)}$

for  $i = 1, 2, \dots$

  solve  $[M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if  $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

**check convergence |r|**

end

$$\text{Resid} = \sqrt{\frac{\text{DNorm2}}{\text{BNorm2}}} = \frac{|r|}{|b|} = \frac{|Ax - b|}{|b|} \leq \text{Tol}$$