Chapter 3 - Advanced Element Formulations

3.1 Shear locking and incompatible mode elements

1. Shear locking phenomenon

(1) FEM techniques for interpolating the displacement field within 2D and 3D meshes were discussed before. In addition, methods for evaluating the area or volume integrals in the *principle of virtual work* were also discussed. These procedures work well for most applications, but there are situations in which the simple element formulations may give inaccurate results.

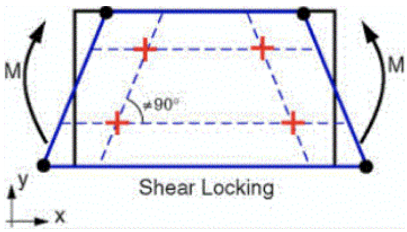
(2) First, we focus on "locking" phenomena. Finite elements are said to lock if they exhibit an unphysically stiff response to deformation. Locking can occur for many different reasons. The most common causes include the following:

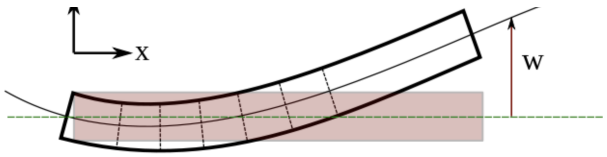
① The governing equations you are trying to solve are poorly conditioned, which leads to an ill-conditioned system of finite element equations.

② The element interpolation functions are unable to approximate accurately the strain distribution in the solid, so the solution converges very slowly as the mesh size is reduced.

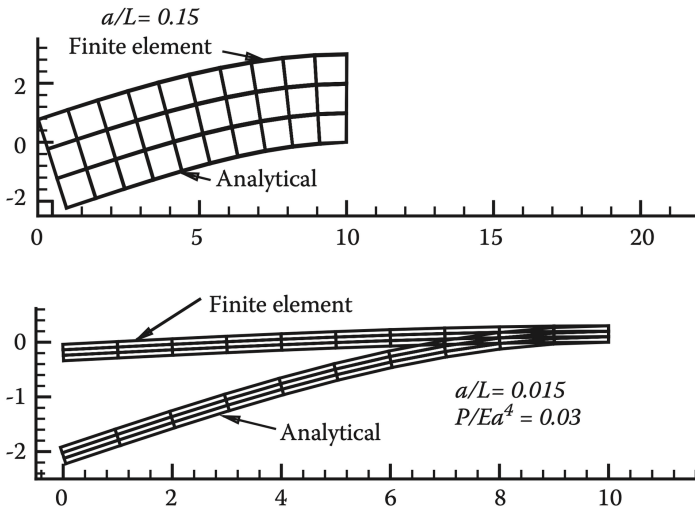
③ In certain element formulations especially in bending problems, displacements and their derivatives are interpolated separately.

(3) In the 2D bending problems of Euler – Bernoulli beam (plates and shells as well), the standard four-noded quadrilateral elements cannot accurately approximate the strain distribution associated with the cross-sections in their planes perpendicular to the neutral plane shown in the figures:





(4) For materials with infinitesimal deformations, such undesired strains existing in the standard elements contributes to the total elastic energy of a system with respect, and therefore naturally enhence the material stiffness. The phenomenon is commonly known as "shear locking" shown below:



(5) The shear locking can be usuallly avoided by using a sufficiently finer mesh. However, finite element analysts sometimes cannot resist the temptation to reduce computational cost by using elongated elements (fewer elements), which may introduce errors.

2. Incompatible mode elements

(1) Hence, *incompatible mode* elements are developed to solve this problem by adding an additional strain distribution to the element to modify the undesired strains existing in standard elements. Such elements are called "incompatible" because the strain is not required to be compatible with the typical displacement interpolation functions. The approach is described as follows:

① The displacement fields in the element are interpolated using the standard scheme, by setting:

$$u_i = \sum_{a=1}^{Ne} N^a(\xi_j) u_i^a \tag{3.1}$$

$$\delta v_i = \sum_{a=1}^{Ne} N^a(\xi_j) \delta v_i^a \tag{3.2}$$

$$x_i = \sum_{a=1}^{Ne} N^a(\xi_j) x_i^a \tag{3.3}$$

where $N_a(\xi_j)$ are the shape functions discussed before, ξ_i are a set of local coordinates in the element, u_i^a, x_i^a denote the displacement values and coordinates of the nodes on the element, and N_e is the number of nodes on the element.

② The Jacobian matrix for the interpolation functions, its determinant, and its inverse are defined in the usual way:

$$\frac{\partial x_i}{\partial \xi_j} = \sum_{a=1}^{Ne} \frac{\partial N^a(\xi)}{\partial \xi_j} x_i^a \tag{3.4}$$

$$|\boldsymbol{J}| = \det(\boldsymbol{J}) \tag{3.5}$$

where the i-th row and j-th column number in the Jacobian matrix $\boldsymbol{J}$ is $\dfrac{\partial x_i}{\partial \xi_j}$, and the i-th row and j-th column number in the inverse matrix of $\boldsymbol{J}^{-1}$ is $\dfrac{\partial \xi_i}{\partial x_j}$.

(2) In the incompatible element, by denoting $J(\mathbf{0}) = |\boldsymbol{J}(\boldsymbol{\xi} = 0)|$ and $J(\boldsymbol{\xi}) = |\boldsymbol{J}(\boldsymbol{\xi})|$, the expression dealing with displacement gradient in such element is assumed by:

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_i}{\partial x_j} \tag{3.6}$$

where

$$\frac{\partial \overline{u}_i}{\partial x_j} = \sum_{a=1}^{Ne} \frac{\partial N^a(\xi)}{\partial \xi_k} \frac{\partial \xi_k}{\partial x_j} u_i^a \tag{3.7}$$

$$\frac{\partial \tilde{u}_i}{\partial x_j} = \tilde{g}_i(\boldsymbol{\xi}) \frac{\partial \xi_k}{\partial x_j} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \tag{3.8}$$

the $\tilde{g}_i(\boldsymbol{\xi})$ is called the *parametric gradient field* function, and such a gradient field is thus assumed to be a linear function of the Gaussian coordinates as the one in the Euler – Bernoulli beam theory, i.e., axial strain in beams as a linear function of distance from the neutral surface:

$$\tilde{g}_i(\boldsymbol{\xi}) = \sum_{k=1}^{P} \alpha_i^{(k)} \xi_k \tag{3.9}$$

$P = \mathbf{2}$ for a 2D problem and $P = \mathbf{3}$ for a 3D problem, $\alpha_i^{(k)}$ are the coefficients of the i-th displacement gradient along the the k-th Gaussian coordinate and to be determined as the degrees of freedom at Gaussian integration points for describing the desired displacement variation field, $J(\mathbf{0})/J(\boldsymbol{\xi})$ is the ratio of area (2D) or volume (3D) mapping from a Gaussian integration point to the center of the element to have zero axial strain in the neutral surface.

(3) The $\alpha_i^{(k)}$ is the set of unknown displacement gradient coefficients in the element, which must be determined as part of the solution. Within an element domain, the assumed displacement gradient in Eq. 3.6 is then substituted into the virtual work equation as usual to obtain the modified formula of element stiffness matrix as follows:

$$\int_{V_e} C_{ijk\ell} \frac{\partial u_k}{\partial x_\ell} \frac{\partial \delta v_i}{\partial x_j} dV - \int_{V_e} b_i \delta v_i \, dV - \int_{\partial_2 V_e} t_i^* \delta v_i \, dA = 0, \quad u_i = u_i^* \text{ on } \partial_1 V_e$$

The external force vector $\boldsymbol{f}$ in the finite element equation of an element as Eq. 2.85 does not have to be modified, since

$$\int_{V_e} b_i \delta v_i \, dV + \int_{\partial_2 V_e} t_i^* \delta v_i \, dA = f_i^a \, \delta v_i^a$$

As for the first integral in the the virtual work equation related to displacement gradients, it can be written further:

$$\begin{aligned}
&\int_{V_e} C_{ijk\ell} \left(\sum_{a=1}^{Ne} \frac{\partial N^a}{\partial x_j} \delta v_i^a + \sum_{m=1}^{P} \xi_m \frac{\partial \xi_m}{\partial x_j} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \delta \alpha_i^m \right) \left(\sum_{b=1}^{Ne} \frac{\partial N^b}{\partial x_\ell} u_k^b + \sum_{n=1}^{P} \xi_n \frac{\partial \xi_n}{\partial x_\ell} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \alpha_k^n \right) dV = f_i^a \, \delta v_i^a \\
&\left(\int_{V_e} C_{ijk\ell} \frac{\partial N^a}{\partial x_j} \frac{\partial N^b}{\partial x_\ell} dV \, u_k^b + \int_{V_e} C_{ijk\ell} \frac{\partial N^a}{\partial x_j} \xi_n \frac{\partial \xi_n}{\partial x_\ell} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} dV \, \alpha_k^n \right) \delta v_i^a \\
&+ \left(\int_{V_e} C_{ijk\ell} \, \xi_m \frac{\partial \xi_m}{\partial x_j} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \frac{\partial N^b}{\partial x_\ell} dV \, u_k^b + \int_{V_e} C_{ijk\ell} \, \xi_m \frac{\partial \xi_m}{\partial x_j} \xi_n \frac{\partial \xi_n}{\partial x_\ell} \left(\frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \right)^2 dV \, \alpha_k^n \right) \delta \alpha_i^m = f_i^a \, \delta v_i^a
\end{aligned}$$

(4) Expand the real and virtual degree-of-freedom vectors as:

$$\text{(3.10) } \boldsymbol{w} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\alpha} \end{bmatrix}$$

$$\text{(3.11) } \delta \boldsymbol{w} = \begin{bmatrix} \delta \boldsymbol{v} \\ \delta \boldsymbol{\alpha} \end{bmatrix}$$

where $\boldsymbol{u}$ and $\delta \boldsymbol{v}$ are both `(nelnd x ndime)`-by-`1` vector, and $\boldsymbol{\alpha}$ and $\delta \boldsymbol{\alpha}$ are both `(ndime x ndime)`-by-`1` vector, and the corresponding modified external force vector is:

$$\text{(3.12) } \boldsymbol{q} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \end{bmatrix}$$

where $\boldsymbol{f}$ is the original external force vector (`nelnd x ndime` by `1`), and $\mathbf{0}$ is a zero vector (`ndime x ndime` by `1`). Based on the weak form of such governing equations of linear elasticity, we are now finding the solution of $\boldsymbol{w}$ containing all the degrees of freedom which are local to an element and satisfy the governing equations for all virtual velocity fields δv_i satisfying $\delta v_i = 0$ on $\partial_1 R$. The finite element equation of an element can be written in a matrix form as:

$$\text{(3.13) } \boldsymbol{k} \, \boldsymbol{w} = \boldsymbol{q}$$

where

$$\text{(3.14) } \boldsymbol{k} = \begin{bmatrix} \boldsymbol{k}^{uu} & \boldsymbol{k}^{u\alpha} \\ \boldsymbol{k}^{\alpha u} & \boldsymbol{k}^{\alpha\alpha} \end{bmatrix}$$

$$\text{(3.15) } k_{aibk}^{uu} = \int_{V_e} C_{ijk\ell} \frac{\partial N^a}{\partial x_j} \frac{\partial N^b}{\partial x_\ell} dV$$

$$\text{(3.16) } k_{aimk}^{\alpha u} = \int_{V_e} C_{ijk\ell} \frac{\partial N^a}{\partial x_j} \xi_m \frac{\partial \xi_m}{\partial x_\ell} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} dV$$

$$\text{(3.17) } k_{miak}^{u\alpha} = \int_{V_e} C_{ijk\ell} \, \xi_m \frac{\partial \xi_m}{\partial x_j} \frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \frac{\partial N^a}{\partial x_\ell} dV$$

$$\text{(3.18) } k_{mink}^{\alpha\alpha} = \int_{V_e} C_{ijk\ell} \, \xi_m \frac{\partial \xi_m}{\partial x_j} \xi_n \frac{\partial \xi_n}{\partial x_\ell} \left(\frac{J(\mathbf{0})}{J(\boldsymbol{\xi})} \right)^2 dV$$

Note that the $\boldsymbol{k}^{u\alpha}$ is the *transpose* of the $\boldsymbol{k}^{\alpha u}$.

(5) What is important to obtain the modified element stiffness matrix is:

$$\text{(3.19) } \boldsymbol{k}^{\alpha u} \boldsymbol{u} + \boldsymbol{k}^{\alpha\alpha} \boldsymbol{\alpha} = \mathbf{0}$$

$$\text{(3.20) } \boldsymbol{\alpha} = -\left(\boldsymbol{k}^{\alpha\alpha} \right)^{-1} \boldsymbol{k}^{\alpha u} \, \boldsymbol{u}$$

By putting $\boldsymbol{\alpha}$ back to

(3.21) $\boldsymbol{k}^{uu}\boldsymbol{u} - \boldsymbol{k}^{u\alpha}\left(\boldsymbol{k}^{\alpha\alpha}\right)^{-1}\boldsymbol{k}^{\alpha u}\boldsymbol{u} = \boldsymbol{f}$

And the modified element stiffness matrix is thus:

(3.22) $\boldsymbol{k}_e = \boldsymbol{k}^{uu} - \boldsymbol{k}^{u\alpha}\left(\boldsymbol{k}^{\alpha\alpha}\right)^{-1}\boldsymbol{k}^{\alpha u}$

where $\boldsymbol{k}^{uu}$ is the original element stiffness matrix, and the last term of the above equation is the addition for modifying element stiffness for satisfying particular deformation behaviors in the Euler – Bernoulli beam theory.

3. Demonstration of finite element analysis code with incompatible mode elements

In the finite element analysis code, all we have to do is to modify the *ElemStif* function:

```
function kel = ElemStif(iel,ndime,nelnd,coor,conn,mate)
  kuu = zeros(ndime*nelnd,ndime*nelnd);
  kau = zeros(ndime*ndime,ndime*nelnd);
  kua = zeros(ndime*nelnd,ndime*ndime);
  kaa = zeros(ndime*ndime,ndime*ndime);
  coorie = zeros(ndime,nelnd);
  xii = zeros(ndime,1);
  dxdxi = zeros(ndime,ndime);
  dNdx = zeros(nelnd,ndime);
  M = numIntegPt(ndime,nelnd);
  xi = IntegPt(ndime,nelnd,M);
  w = integWt(ndime,nelnd,M);
  for a = 1:nelnd
    for i = 1:ndime
      coorie(i,a) = coor(i,conn(a,iel));
    end
  end
  dNdxi = ShpFuncDeri(nelnd,ndime,xii);
  for i = 1:ndime
    for j = 1:ndime
      for a = 1:nelnd
        dxdxi(i,j) = dxdxi(i,j)+coorie(i,a)*dNdxi(a,j);
      end
    end
  end
  jcb0 = det(dxdxi);
  for im = 1:M
    for i = 1:ndime
      xii(i) = xi(i,im);
    end
    dNdxi = ShpFuncDeri(nelnd,ndime,xii);
    dxdxi(:) = 0;
    for i = 1:ndime
      for j = 1:ndime
        for a = 1:nelnd
          dxdxi(i,j) = dxdxi(i,j)+coorie(i,a)*dNdxi(a,j);
        end
      end
    end
    dxidx = inv(dxdxi);
    jcb = det(dxdxi);
    dNdx(:) = 0.;
    for a = 1:nelnd
      for i = 1:ndime
        for j = 1:ndime
```

```
        dNdx(a,i) = dNdx(a,i)+dNdxi(a,j)*dxidx(j,i);
      end
    end
  end
  cmat = MatStif(ndime,mate);
  for a = 1:nelnd
    for i = 1:ndime
      for b = 1:nelnd
        for k = 1:ndime
          ir = ndime*(a-1)+i;
          ic = ndime*(b-1)+k;
          for j = 1:ndime
            for l = 1:ndime
              kuu(ir,ic) = kuu(ir,ic)+cmat(i,j,k,l)*dNdx(a,j)*dNdx(b,l)*w(im)*jcb;
            end
          end
        end
      end
    end
  end
  for a = 1:nelnd
    for i = 1:ndime
      for m = 1:ndime
        for k = 1:ndime
          ir = ndime*(a-1)+i;
          ic = ndime*(m-1)+k;
          for j = 1:ndime
            for l = 1:ndime
              tmp = cmat(i,j,k,l)*dNdx(a,j)*(xii(m)*dxidx(m,l)*jcb0/jcb)*w(im)*jcb;
              kau(ir,ic) = kau(ir,ic)+tmp;
              kua(ic,ir) = kua(ic,ir)+tmp;
            end
          end
        end
      end
    end
  end
  for m = 1:ndime
    for i = 1:ndime
      for n = 1:ndime
        for k = 1:ndime
          ir = ndime*(m-1)+i;
          ic = ndime*(n-1)+k;
          for j = 1:ndime
            for l = 1:ndime
              kaa(ir,ic) = kaa(ir,ic)+...

cmat(i,j,k,l)*xii(m)*dxidx(m,j)*xii(n)*dxidx(n,l)*jcb0/jcb*jcb0/jcb*w(im)*jcb;
            end
          end
        end
      end
    end
  end
```

```
  end
  kel = kuu-kua*(kaa\kau);
end
```

3.2 Interactive and scientific visualization of finite element analysis results

1. ParaView

   From the official website of ParaView (https://www.paraview.org/download/), the features of ParaView are:

   (1) ParaView is an open-source, multi-platform application designed to visualize data sets of varying sizes from small to very large.

   (2) The goals of the ParaView project include developing an open-source, multi-platform visualization application that supports distributed computational models to process large data sets.

   (3) ParaView has an open, flexible, and intuitive user interface.

   (4) ParaView is built on an extensible architecture based on open standards.

   (5) Please check the Documentation including *"ParaViewGettingStarted"* and *"ParaViewTutorial"* PDF files provided on the official website of ParaView.

2. Extrapolation element strains and stresses to nodal values

   (1) FEM successfully complete certain quantities such as strains and stresses with the best precision at the Gaussian integration points of elements.

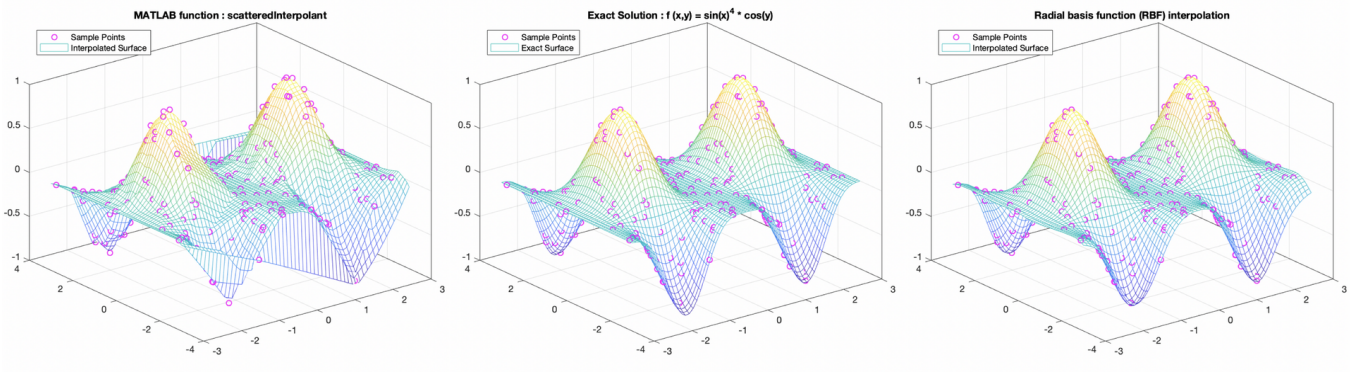
   (2) In scientific visualization, it is more preferable to have accurate evaluation of strain and stress distributions at nodes of meshes.

   (3) Here, a general and efficient method called *"Radial basis function interpolation"* is suggested to performing interpolation (also including extrapolation) on a 2D or 3D field data set of scattered computation results to have the best field esitmation on desired points.

   (4) MATLAB code demonstration:

```
n = 250;
x = -3 + 6*rand(n,1);
y = -3 + 6*rand(n,1);
v = sin(x).^4 .* cos(y);
F = scatteredInterpolant(x,y,v);
[xq,yq] = meshgrid(-3:0.1:3);
F.Method = 'linear';
F.ExtrapolationMethod = 'linear';
vq = F(xq,yq);
figure;
plot3(x,y,v,'mo');
hold on;
mesh(xq,yq,vq);
title('MATLAB function: scatteredInterpolant');
legend('Sample Points','Interpolated Surface','Location','NorthWest');
hold off;
figure;
plot3(x,y,v,'mo');
hold on;
mesh(xq,yq,sin(xq).^4 .* cos(yq));
title('Exact Solution: f(x,y) = sin(x)^4 * cos(y)');
legend('Sample Points','Exact Surface','Location','NorthWest');
rbfop=rbfcreate([x(:)'; y(:)'], v(:)','RBFFunction','multiquadric','Stats', 'on');
rbfcheck(rbfop);
vqrbf = rbfinterp([xq(:)'; yq(:)'],rbfop);
figure;
plot3(x,y,v,'mo');
hold on;
mesh(xq,yq,reshape(vqrbf, size(xq)));
title('Radial basis function (RBF) interpolation');
```

```
legend('Sample Points','Interpolated Surface','Location','NorthWest');
hold off;
```



(5) Hence, the nodal stresses and strains can be properly interpolated or extrapolated based on the the scattered stress and strain values at the Gaussian integration points of elements.

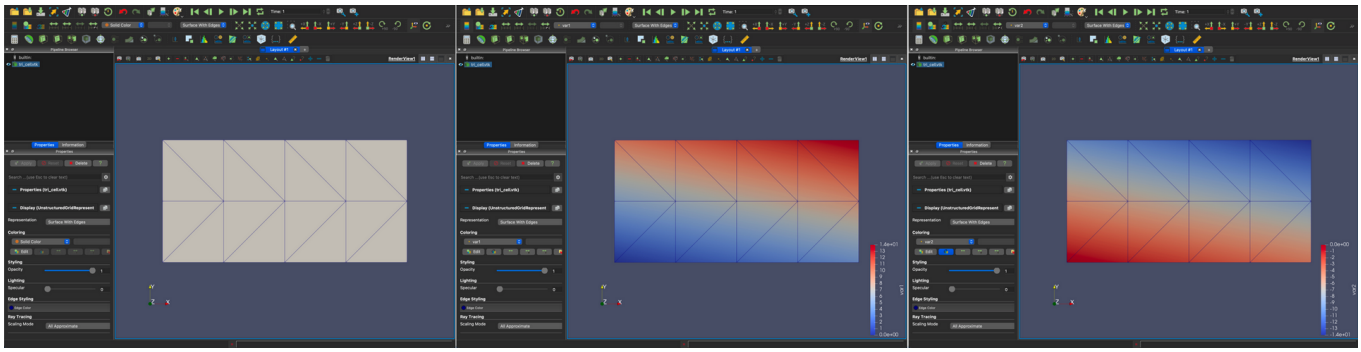
3. VTK files of meshes for ParaView visualization

(1) The VTK files of meshes for ParaView visualization are recommended to be written in the ASCII format as follows:

① 2D meshes of triangular elements (file : *.vtk)

```
# vtk DataFile Version 2.0
Generated Volume Mesh
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 15 float
0.00 0.00 0.00
1.00 0.00 0.00
2.00 0.00 0.00
3.00 0.00 0.00
4.00 0.00 0.00
0.00 1.00 0.00
1.00 1.00 0.00
2.00 1.00 0.00
3.00 1.00 0.00
4.00 1.00 0.00
0.00 2.00 0.00
1.00 2.00 0.00
2.00 2.00 0.00
3.00 2.00 0.00
4.00 2.00 0.00
CELLS 16 64
3 0 6 5
3 0 1 6
3 1 7 6
3 1 2 7
3 2 8 7
3 2 3 8
3 3 9 8
3 3 4 9
3 5 6 10
3 6 11 10
3 6 7 11
3 7 12 11
3 7 8 12
3 8 13 12
3 8 9 13
3 9 14 13
```

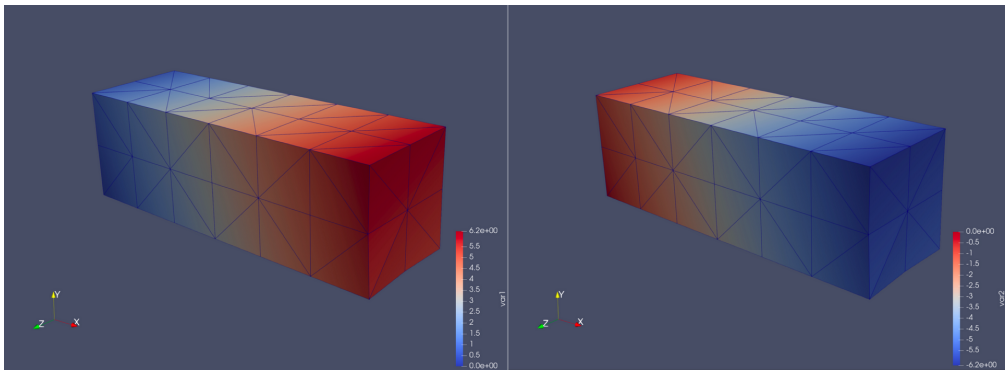
```
CELL_TYPES 16
5
5
5
5
5
5
5
5
5
5
5
5
5
5
5
5
POINT_DATA 15
SCALARS var1 float
LOOKUP_TABLE default
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
SCALARS var2 float
LOOKUP_TABLE default
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-6.0
-7.0
-8.0
-9.0
-10.0
-11.0
-12.0
-13.0
-14.0
```



② 3D meshes of tetrahedral elements (file : *.vtk)

```
# vtk DataFile Version 2.0
Generated Volume Mesh
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 63 float
0 0 0
0 0 0.500000000000000
0 0 1.000000000000000
0 0.500000000000000 0
0 0.500000000000000 0.500000000000000
0 0.500000000000000 1.000000000000000
0 1.000000000000000 0
0 1.000000000000000 0.500000000000000
0 1.000000000077169 1.000000000077169
0.500001250264599 0 0
0.499996249206202 0 0.500000000000000
. . .
3.000000000077169 1.000000000077169 0
3.000000000000000 1.000000000000000 0.499998749735401
3.000000000000000 1.000000000000000 1.000000000000000
CELLS 144 720
4 9 12 3 4
4 9 4 3 0
4 9 12 4 13
4 9 1 4 0
4 9 4 1 13
4 9 1 10 13
4 14 4 1 2
4 14 1 4 13
4 14 1 11 2
4 14 10 1 13
4 14 5 4 2
4 14 1 10 11
4 21 28 30 31
4 21 18 27 28
4 21 22 19 28
. . .
4 41 40 37 28
4 41 38 29 28
CELL_TYPES 144
10
10
10
```

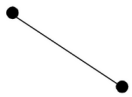
```
10
10
10
10
10
10
10
. . .
10
10
POINT_DATA 63
SCALARS var1 float
LOOKUP_TABLE default
0
0.100000000000000
0.200000000000000
0.300000000000000
0.400000000000000
0.500000000000000
0.600000000000000
0.700000000000000
0.800000000000000
0.900000000000000
1.000000000000000
1.100000000000000
1.200000000000000
1.300000000000000
. . .
6.100000000000001
6.200000000000000
SCALARS var2 float
LOOKUP_TABLE default
0
-0.100000000000000
-0.200000000000000
-0.300000000000000
-0.400000000000000
-0.500000000000000
-0.600000000000000
-0.700000000000000
-0.800000000000000
-0.900000000000000
-1.000000000000000
-1.100000000000000
-1.200000000000000
-1.300000000000000
-1.400000000000000
-1.500000000000000
-1.600000000000000
. . .
-6.100000000000001
-6.200000000000000
```



(2) Linear cell types in VTK files. Use the keyword *CELL_TYPES* to manipulate cell types:

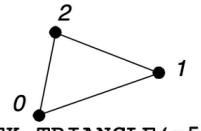
VTK_VERTEX (=1)

VTK_POLY_VERTEX (=2)

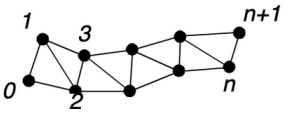


VTK_LINE (=3)

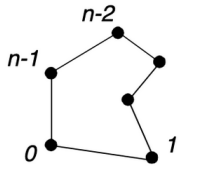
VTK_POLY_LINE (=4)



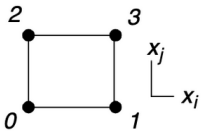
VTK_TRIANGLE(=5)



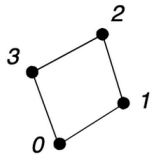
VTK_TRIANGLE_STRIP (=6)



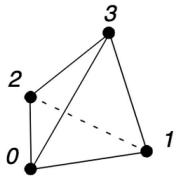
VTK_POLYGON (=7)



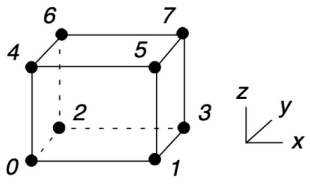
VTK_PIXEL (=8)



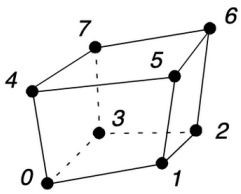
VTK_QUAD (=9)



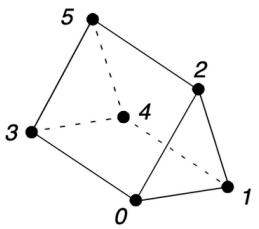
VTK_TETRA (=10)



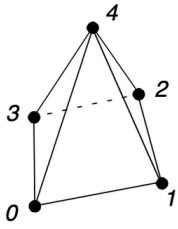
VTK_VOXEL (=11)



VTK_HEXAHEDRON (=12)

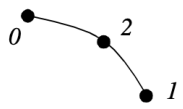


VTK_WEDGE (=13)

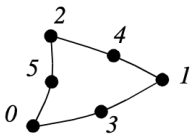


VTK_PYRAMID (=14)

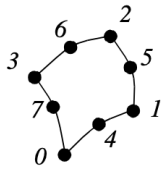
(3) Non-linear cell types in VTK files. Use the keyword *CELL_TYPES* to manipulate cell types:



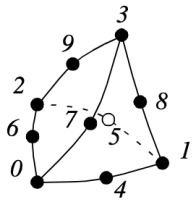
VTK_QUADRATIC_EDGE
(=21)



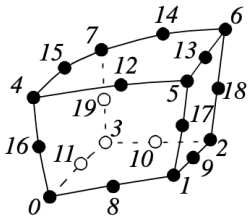
VTK_QUADRATIC_TRIANGLE
(=22)



VTK_QUADRATIC_QUAD
(=23)



VTK_QUADRATIC_TETRA
(=24)



VTK_QUADRATIC_HEXAHEDRON
(=25)