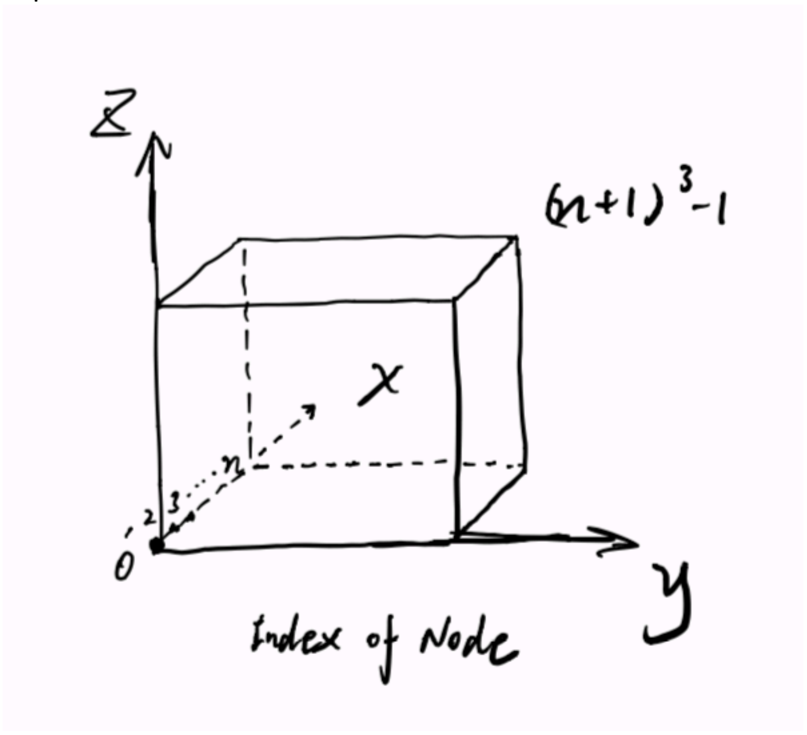
LULESH Serial Code Analysis

sy8wg Shunyu Yao

Nov 14, 2018

1. **Variable Declaration**
   From the beginning of the code to L473, it is basically the **1) declaration, 2) access function, 3) allocation function** of all variables that define a mesh in a LULESH computation. and for all variables, it is again divided into 3 parts: Node-centered variables, Element-centered variables, and all other parameters for the whole mesh such as running time, cycle, or coefficients, etc. Basically, a full list of variables of a mesh are from L322-L430. Some "Boundary Conditions" "on each of 6 hexahedral faces" are defined at L451-L473, but the geometrical meaning is unknown.
   Assuming the size of the mesh is n, then "Node-centered" parameters, are now interpreted as all characters relative to the nodes, usually with respect to the size of n+1, while "Element-centered" parameters may be interpreted as characters between nodes, usually with respect to the size of n.

2. **Main Function Initialization**
   a. From L2662-L2695, the **mesh size** is initialized, with variables assigned with sizes accordingly.
      In serial code, the mesh size is given by the input and the rest of the mesh is generated including values of each node and element such as coordinates, coefficients, energies, velocities, etc.
      The size of variables highly depends on the dimensions involved, so they'll be explained in details later.



   b. From L2699-L2718, the **coordinates of each node** in the cube mesh is initialized.

The values of each node's coordinates are assigned, with a unit increment size of

$$\frac{1.125}{n}$$

n is the size of the mesh.

Note that the index of each node here is represented in one dimension, namely, the last node's index of the whole mesh would be

$$(n+1)^3 - 1$$

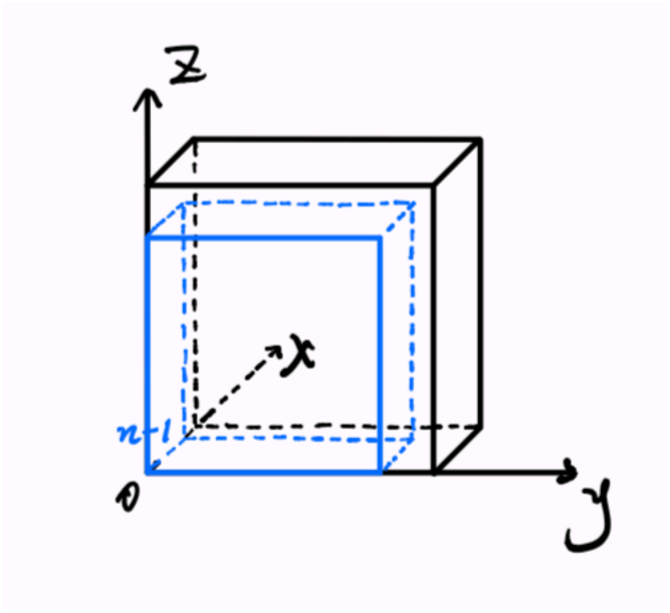
Because the number of nodes are $(n+1)^3$ and in arrays, the index needs minus 1.

c. From L2723-L2743, the **Local Node list (n^3)** is assigned for the Origin-side of n^3 nodes, call it **Element Nodes**, or **Element**.

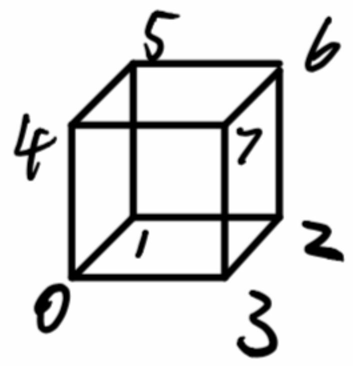
The "Origin-side" means that the nodes that have a list of 8 Local Nodes are the cube of nodes with a size of n^3 that is within the whole cube mesh, and at the $\boldsymbol{O}$ corner of the coordinate system. Thus, it leads to the fact that there are

$$(n+1)^3 - n^3$$

nodes without local nodes, because they are (presumably) at the boundary of the mesh. Each local node is stored as an overall node index in the mesh, and 8 node indexes are stored as an array. A certain node's local node array can be extracted from **nodelist()** with its own index.



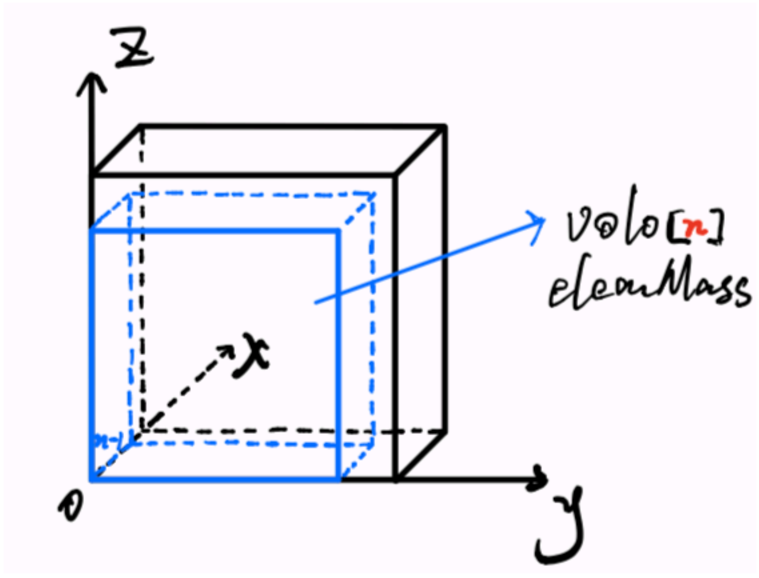
From 0-7, the extracted local node array indicates: **The node itself; the adjoint node of it on X axis; the adjoint node of it on both X and Y axis (diagonal); the adjoint node of it on Y axis; the four other nodes with the same layout on the next node plane above it.**

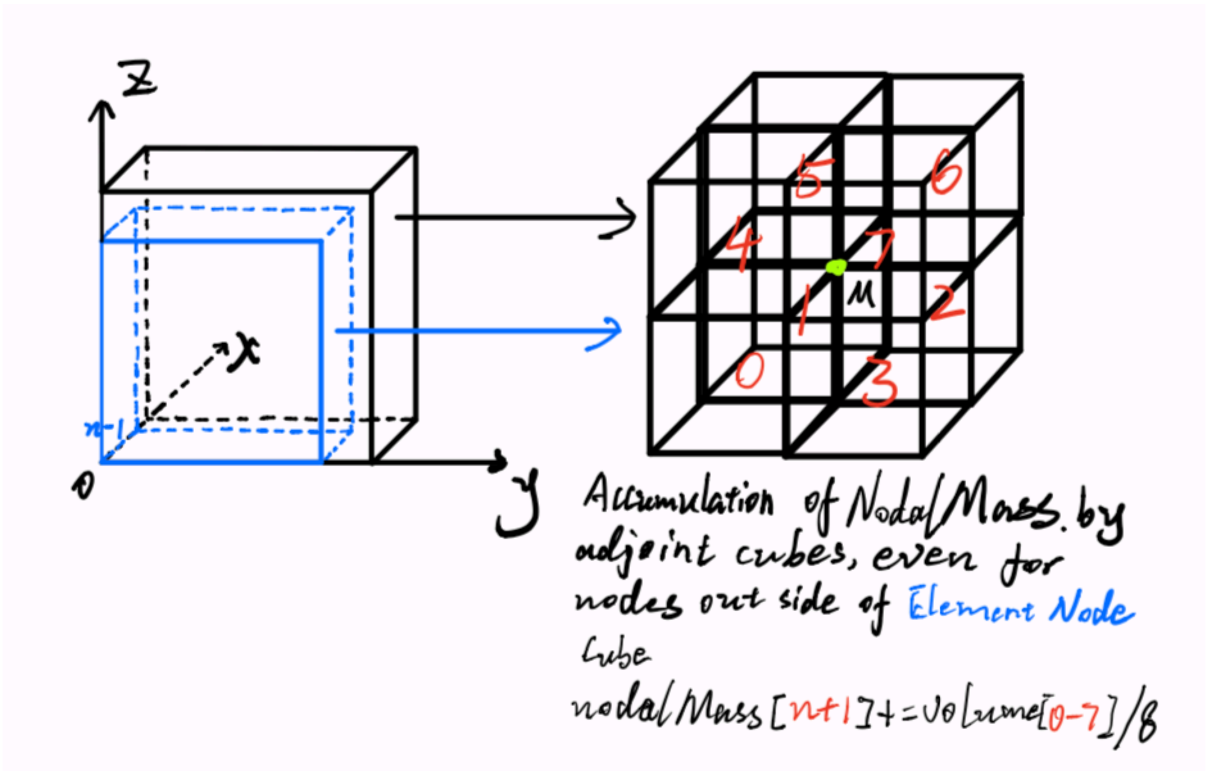


d. At L2747, the **matElemlist (n^3)** index set is initialized, presumably representing sets of materials.
e. From L2751-L2786, a list of **coefficients** and time counters are initialized. Most of them are fixed, not for modification in the latter computation.
f. From L2789-L2806, “Field Data”, the **Reference Volume (n^3)** is computed for the nodes that have local nodes according to its local nodes through **CalcElemVolume()**; **ElemMass(n^3)** is the same with Volume, and **NodalMass($(n+1)^3$)** is computed for all nodes as

$$\frac{volume}{8}$$

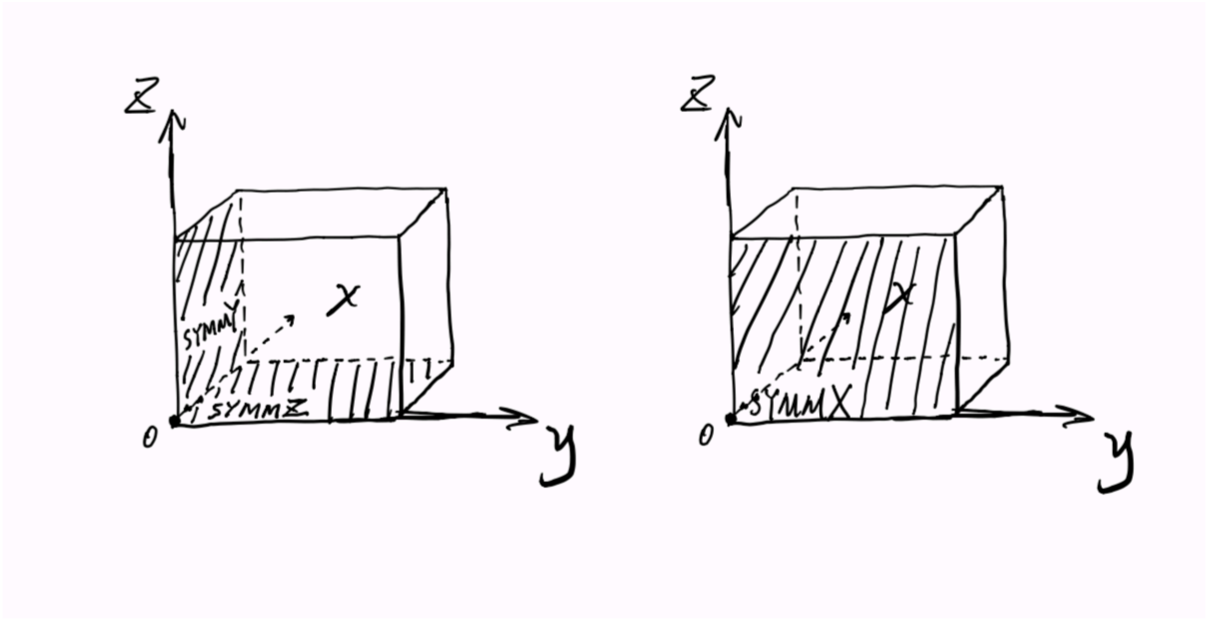
each time a volume is computed. And it accumulates all computation as the node’s mass. Namely, at most, when the node is not at boundary, a node’s mass is determined by 8 volumes computed for 8 element cubes that share the node. (L2804-L2806)




Z
X
n-1
0
Y
5
6
4
7
M
1
2
0
3
Accumulation of Nodal Mass by adjoint cubes, even for nodes out side of Element Node Cube
nodalMass[n+1]+=volume[0-7]/8

g. At L2811, **deposit energy (e(0))** is determined.

h. From L2814-L2822, **symmetric planes of different axis (symmX(), symmY(), symmZ(), each of size n^2)** are defined.
The symmetric plane of an axis is the plane that passes through other two axis.



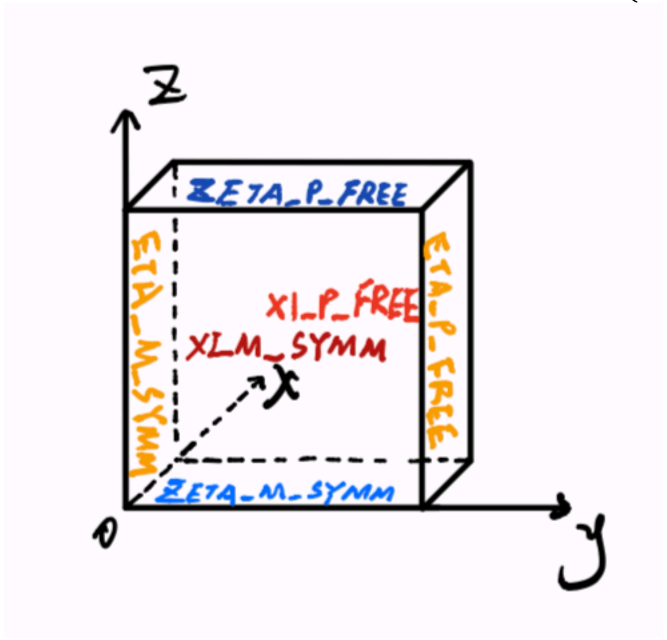
i. From L2827-2854, **lxim, lxip, letam, letap, lzetam, lzetap(n^3)** are set to be the following number arrays:

$$lxim\!: 0; 0, 1, 2, \dots\ [n^3-1-1]$$
$$lxip\!: 1, 2, 3, \dots\ [n^3-1]; [n^3-1]$$
$$letam\!: 0, \dots [n-1]; 0, \dots [n^3-n-1]$$
$$letap\!: [n], \dots [n^3-1]; [n^3-n], \dots [n^3-1]$$
$$lzetam\!: 0, \dots [n^2-1]; 0, \dots [n^3-n^2-1]$$

$$lzetap: [n^2], \dots [n^3 - 1]; [n^3 - n^2], \dots [n^3 - 1]$$

j. From L2859-L2868, initializing Boundary condition info.
First set **elemBC($6n^3$)** as all 0, and then do the "|=" operations with boundary conditions (see L451-L473) on each faces of the mesh cube:

$$x = n \; on \; XI\_P\_FREE(0x008)$$
$$x = 0 \; on \; XI\_M\_SYMM(0x001)$$
$$y = n \; on \; ETA\_P\_FREE(0x080)$$
$$y = 0 \; on \; ETA\_M\_SYMM(0x010)$$
$$z = n \; on \; ZETA\_P\_FREE(0x800)$$
$$z = 0 \; on \; ZETA\_M\_SYMM(0x100)$$



3. **Computation**
From L2877-L2885 is the main computation part of the LULESH solution, which is in the shape of a loop simulating iterations based on timesteps.
    a. **TimeIncrement()** (L477-L523)
    L2878 is the first step of each iteration, setting up time parameters (initializing, increment adjustment) to control computations. There is another time control function CalcTimeConstraintsForElems() at the end of next computation function LagrangeLeapFrog() to wrap up each iteration. TimeIncrement() here is basically doing 3 things: **1) count time and iterations, 2) adjust the unit of increment if necessary, judging by the consequence of last round of computation, 3) Terminating computation if the given time left is little enough, judging by current time increment unit.**
    1) is simple, it first computes the current time left $targetdt$, and after **2)** and **3)** it makes:

$$mesh.time += mesh.deltatime$$
$$mesh.cycle++$$

    2) If fixed coefficient $dtfixed \le 0$ and current computation is not the initial cycle, then it starts to decide whether to upgrade $deltatime$ or not.
        a. Set a candidate for $deltatime$ as $newdt$, initialize it as:

$$newdt = INFINITE \; (very \; big)$$

And

$$olddt = deltatime$$

b. Updating $newdt$ according to $dthydro$ and $dtcourant$, which are two variables computed after an iteration of computing done in CalcTimeConstraintsForElems(). (L486-L492)

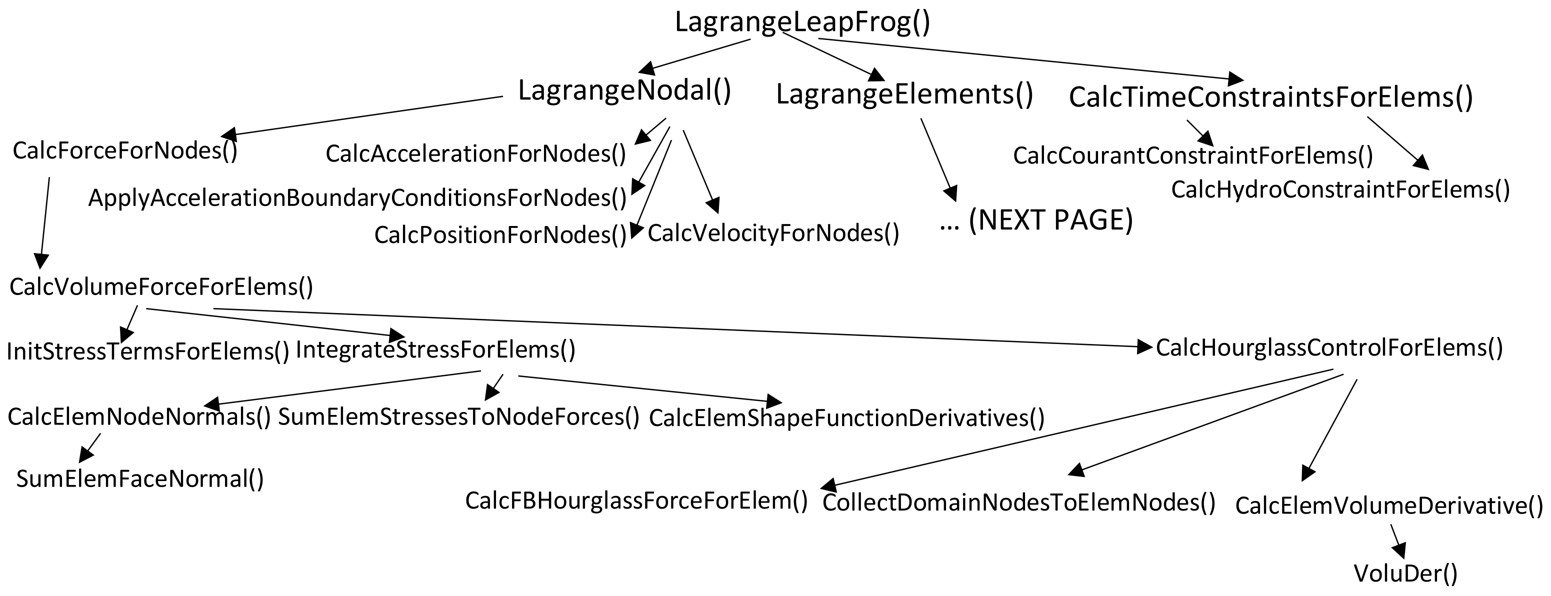
c. Compare the candidate and the old $deltatime$, essentially, if $newdt$ is 1) bigger but under given ratio lower bound (defined in main function initialization), $newdt$ is used; but if 2) bigger than a ratio upper bound, $olddt * ub$ is used; 3) if $olddt$ is bigger, then it is still used. (L494-L502)

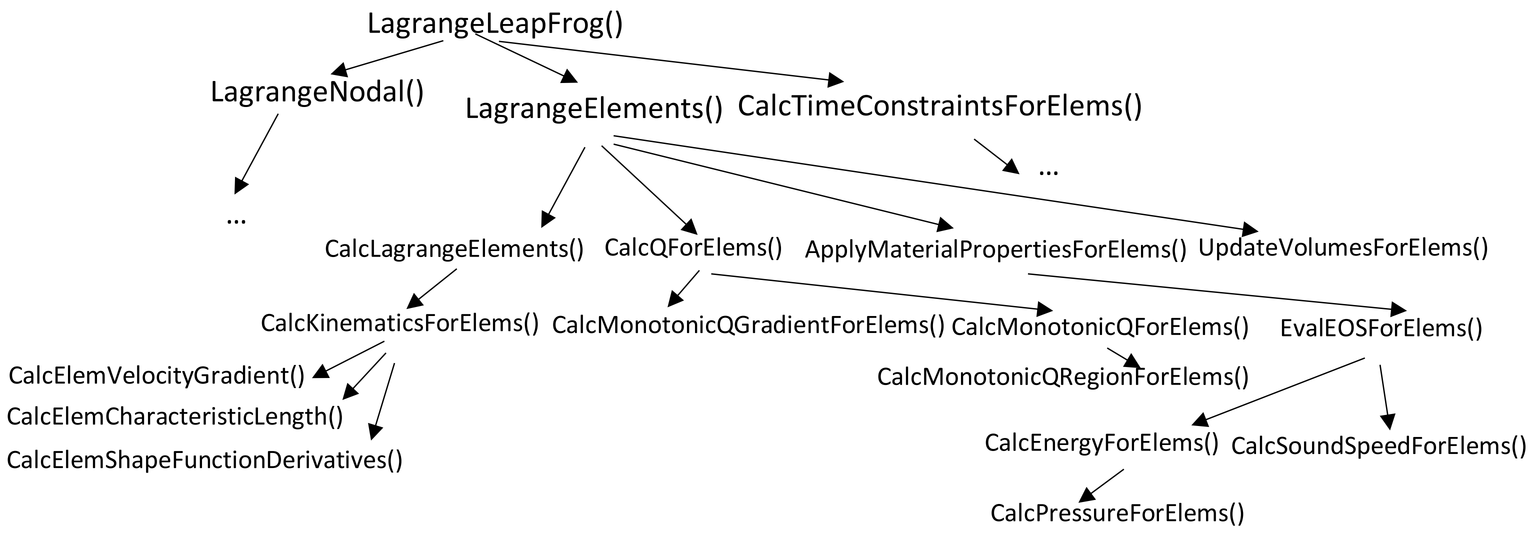
d. Finally, if the decision made above is bigger than an ultimate maximum limit, then use the limit as new $deltatime$. (L504-L508)

3) If the time left is under $\frac{4deltatime}{3}$, then end the computation. (L511-L518)

b. **LagrangeLeapFrog()**

This function does all the computation. A function call tree is presented, without counting in the local code part of each function call. Generally, there are two big chunk of computation, **Nodal part** and **Element part**.


LagrangeLeapFrog()
LagrangeNodal()
LagrangeElements()
CalcTimeConstraintsForElems()
CalcForceForNodes()
CalcAccelerationForNodes()
ApplyAccelerationBoundaryConditionsForNodes()
CalcPositionForNodes()
CalcVelocityForNodes()
... (NEXT PAGE)
CalcCourantConstraintForElems()
CalcHydroConstraintForElems()
CalcVolumeForceForElems()
InitStressTermsForElems()
IntegrateStressForElems()
CalcHourglassControlForElems()
CalcElemNodeNormals()
SumElemStressesToNodeForces()
CalcElemShapeFunctionDerivatives()
SumElemFaceNormal()
CalcFBHourglassForceForElem()
CollectDomainNodesToElemNodes()
CalcElemVolumeDerivative()
VoluDer()



c. **LagrangeNodal()**

Computes the **nodal part** of the computation. (L1490-L1509)

i. **CalcForceForNodes()**

Computes the force (**fx, fy, fz**) on each node. It initializes the force cube by assigning 0 as fx, fy, fz of each point. (L1412-L1427)

ii. **CalcVolumeForceForElems()**

Computes the force on each node ($(n+1)^3$) according to the stress on each element (n^3) (**IntegrateStressForElems()**) and hourglass force on each node(($(n+1)^3$) (**CalcHourglassControlForElems()**) (L1378-L1410).

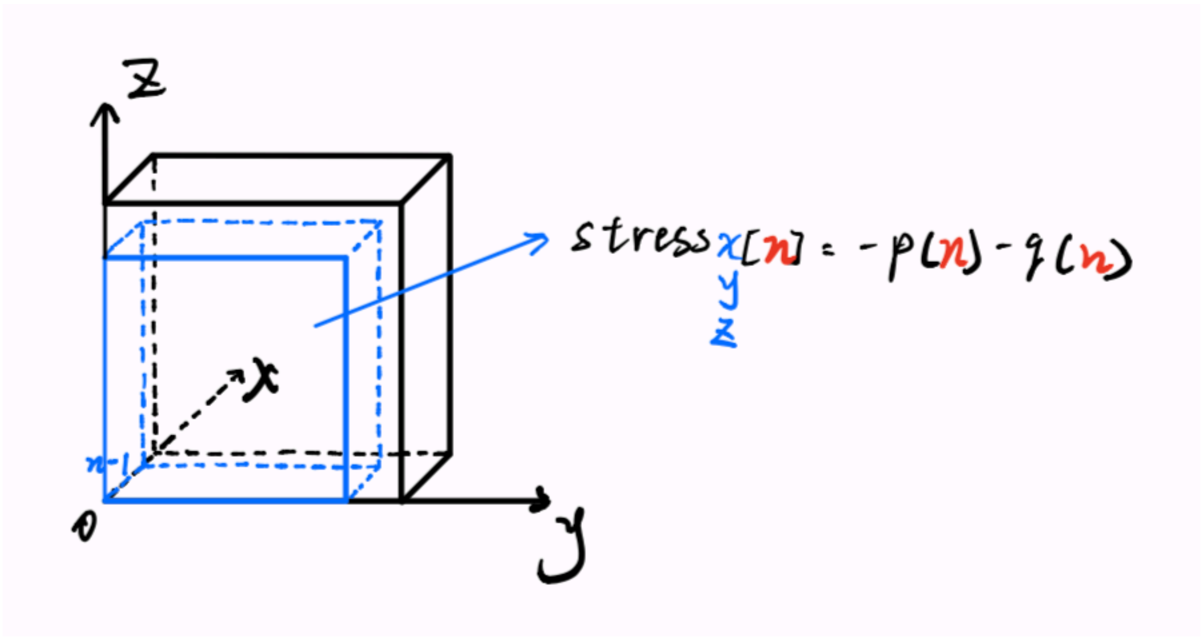
Two different kinds of forces are all computed individually for each group of local nodes first, then apply to each of the node in **nodelist**, ones that have local nodes, and because one node in the mesh can be the local node of other nodes, so forces are accumulated on one single node, this accumulation happens not only in the computation for one force, but also between different forces. Here, the overall calculation of different force on a single node is accumulated as the final result.

One thing notable is that, **Jacobian Determinant**, in **IntegrateStressForElems()**, once its computed, it's never used, and even erased and recomputed later in **CalcHourglassControlForElems()**. To be explained later.

iii. **InitStressTermsForElems(Index_t numElem, Real_t *sigxx, Real_t *sigyy, Real_t *sigzz)**

Initializes **stress** on 3 dimensions of each element (n^3), assigning them all as:

$$-p(i) - q(i)$$

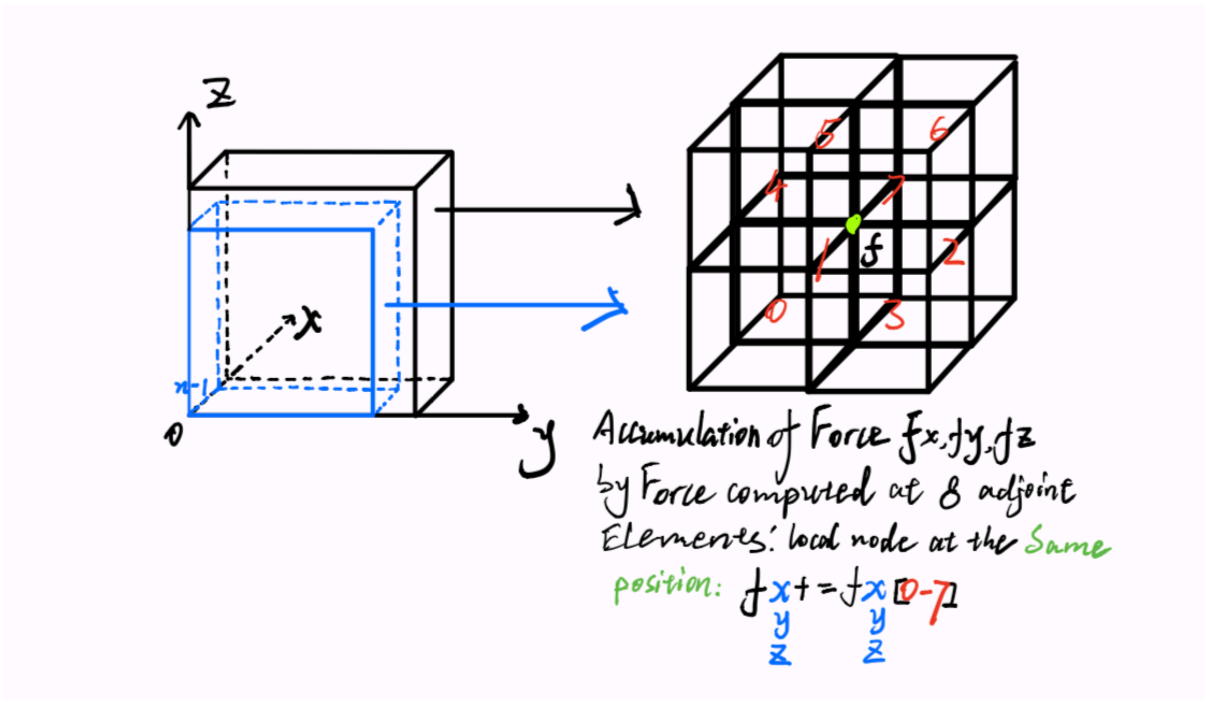


with respect to pressure and q of each element. $sigxx, sigyy, sigzz$ are **outputs**. (L525-L535)

iv. **IntegrateStressForElems (Index_t numElem, Real_t *sigxx, Real_t *sigyy, Real_t *sigzz, Real_t *determ)**

Extracting the local nodes for each element node, and for each element node (n^3):

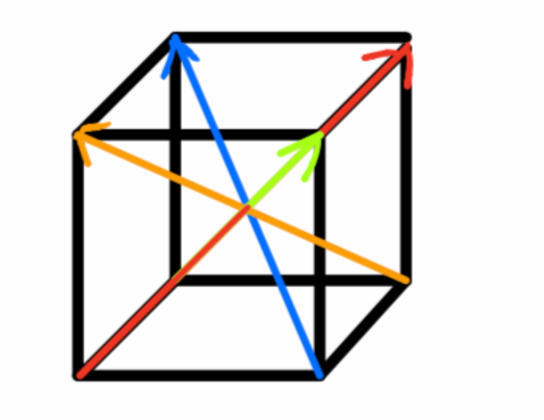
calculate the **derivatives** (**CalcElemShapeFunctionDerivatives()**); calculate the **normal for each faces** of the local nodes cube (**CalcElemNodeNormals()**); calculate (sum) the **stress force**(**SumElemStressesToNodeForces()**).(L771-L805). Finally, accumulates all the stress force on each node (**fx, fy, fz**). (L809-L817) Note that even though it is said to be computing the **stress force** only for each **element node(**n^3**)**, the force is still for all the **nodes(**$(n+1)^3$**)**.



z
x
y
0
n-1
5
6
4
7
1
f
2
0
3
Accumulation of Force fx,fy,fz
by Force computed at 8 adjoint
Elements' local node at the Same
position: fx+=fx[0-7]
y y
z z

v. **CalcElemShapeFunctionDerivatives(const Real_t* const x, const Real_t* const y, const Real_t* const z, Real_t b[][8], Real_t* const volume)**

Calculates 9 factors out of 4 diagonal vectors pointing bottom-up of a local node cube, 3 factors (XI, ETA, ZETA) of each dimension, each factor determined by all 4 vectors;(L566-L576)



Calculates 9 cofactors, 3 (XI, ETA, ZETA) for each dimension, each depending on other two dimensions' different two factors, for example,

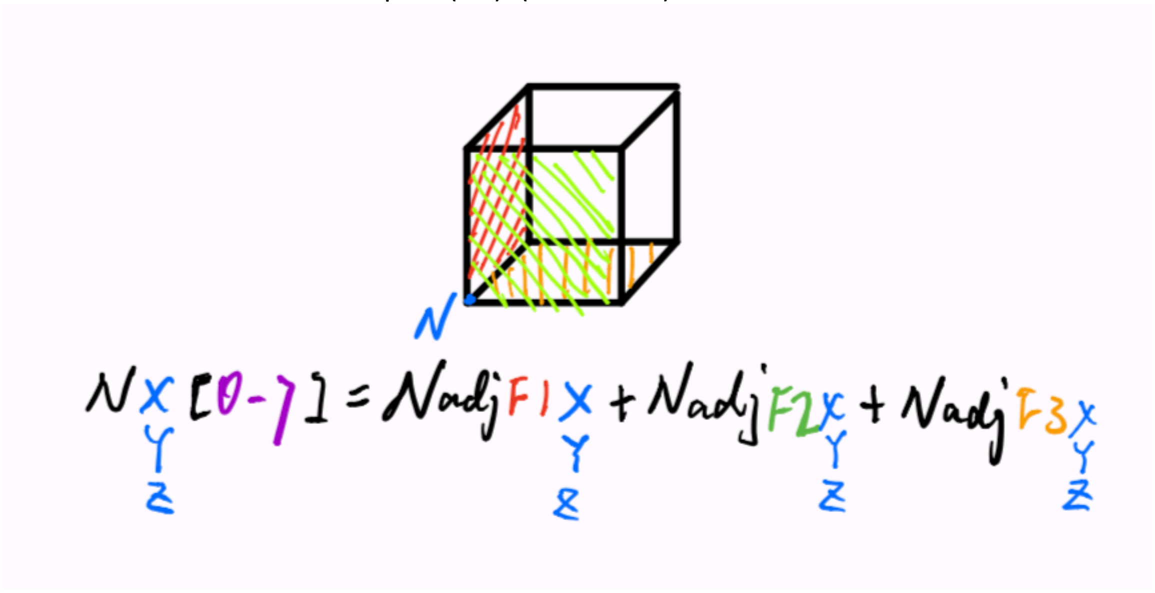
$$cXIx = (ETAy * ZETAz) - (ETAz * ZETAy);\text{(L579-L589)}$$

Calculates partials (derivatives) for different local node in 3 dimensions, an array of 3*8 is output. Each derivative is solely determined by cofactors in the same dimension; For same dimension, derivative of a local node is symmetric to the one of its diagonal local node, for example:

$$b[0][0] = -cXIx - cETAx - cZETAx;$$
$$b[0][6] = -b[0][0];\text{(L595-L620)}$$

Calculates Jacobian Determinant (volume), which is determined only by ZETA factors & cofactors.(L623)
volume and **b[][]** are the outputs.

vi. **CalcElemNodeNormals(Real_t pfx[8], Real_t pfy[8], Real_t pfx[8], const Real_t x[8], Real_t y[8], Real_t z[8])**
Initializes the input derivative array for each local node in 3D (which is confusing, because the derivatives calculated in **CalcElemShapeFunctionDerivatives** are not used, and now it is made place for the **Normal** output);
Computes **normals** for each face of the input local node cube. The coordinates of each 4 local nodes representing a face of the cube are given as input for the **SumElemFaceNormal()** to compute the normal on each local node. 6 faces are all separately computed. The **pfx/y/z** are outputs (3D). (L662-L717)


N
N X Y Z [0-7] = Nadj F1 X Y Z + Nadj F2 X Y Z + Nadj F3 X Y Z

vii. **SumElemFaceNormal (Real_t *normalX0, Real_t *normalY0, Real_t *normalZ0, Real_t *normalX1, Real_t *normalY1, Real_t *normalZ1, Real_t *normalX2, Real_t *normalY2, Real_t *normalZ2, Real_t *normalX3, Real_t *normalY3, Real_t *normalZ3, const Real_t x0, const Real_t y0, const Real_t z0, const Real_t x1, const Real_t y1, const Real_t z1, const Real_t x2, const Real_t x2, const Real_t y2, const Real_t z2, const Real_t x3, const Real_t y3, const Real_t z3)**
6 **bisects** X0, X1, Y0, Y1, Z0, Z1 are computed according to the coordinates in the same dimension, for example:

$$bisectX0 = 0.5 * (x3 + x2 - x1 - x0);$$
$$bisectX1 = 0.5 * (x2 + x1 - x3 - x0);$$

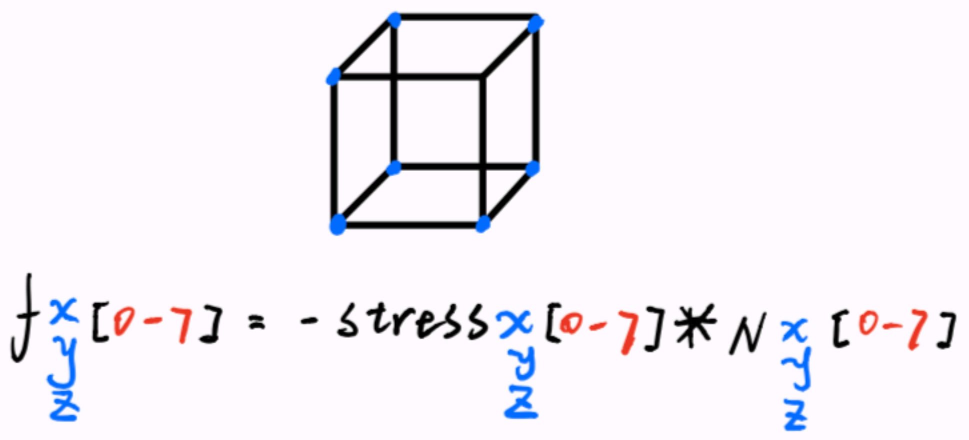
And **areaX, areaY, areaZ** are computed through the 4 bisects from different two dimensions:

$$areaX = 0.25 * (bisectY0 * bisectZ1 - bisectY1 * bisectZ0);$$

output normal through **normal[X-Z][0-3]**, and every 4 normals of each dimension are added with **area[X-Y]**, and accumulates because each local node will be accumulatively computed 3 times (3 faces share one vertex node). (L626-L660)

viii. **SumElemStressesToNodeForces(const Real_t B[][8], const Real_t stress_xx, const Real_t stress_yy, const Real_t stress_zz, Real_t* const fx, Real_t* const fy, Real_t* const fz)**

Multiply the **stress** computed for each local node with **derivative** (**normals**) on each local node in 3D, and output **fx/y/z,** the negative value of the multiplication as the force of each local node.(L719-L769)

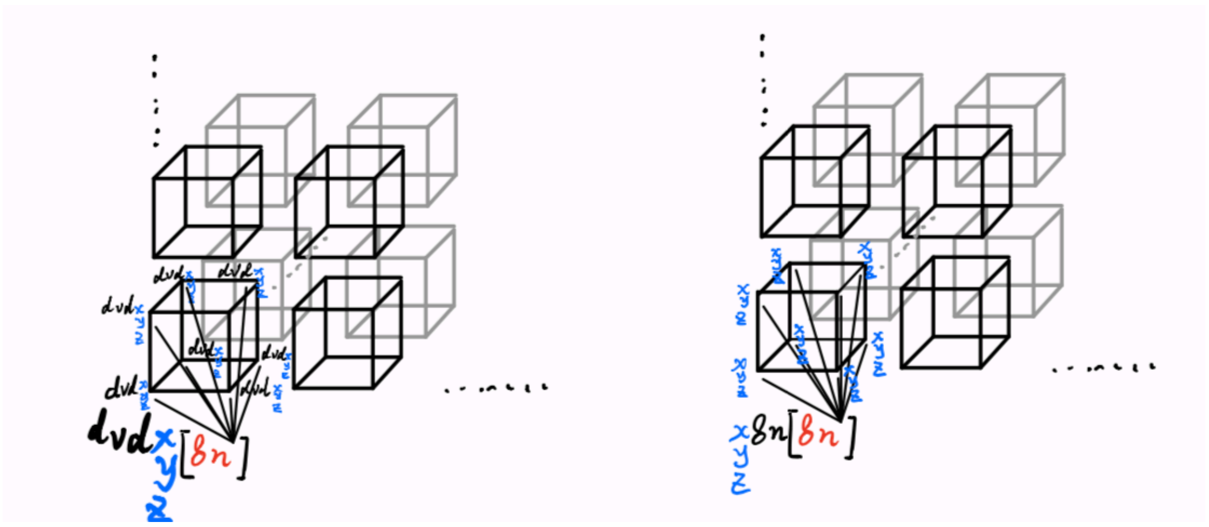


ix. **CalcHourglassControlForElems(Real_t determ[], Real_t hourg, Real_t *x8n, Real_t *y8n, Real_t *z8n, Real_t *dvdx, Real_t *dvdy, Real_t *dvdz)**

Calculates the precondition for computation of **Hourglass force** and compute it for each local node of an element node and accumulatively added them together on each node of the whole mesh. The input parameters are **determ[]**, the **volume** for each Element node calculated during **CalcElemShapeFunctionDerivatives** when computing **stress force**; and **hourglass coefficient**.

For each Element node (n^3):

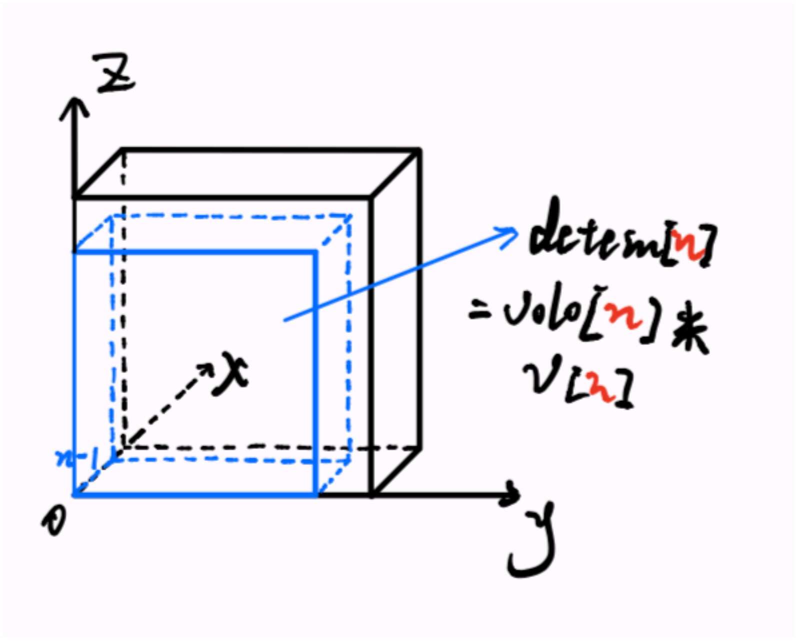
it takes out its local nodes and put its coordinates in 3D (**CollectDomainNodesToElemNodes**, into **x1, y1, z1**)(L1340);
Calculates Element Volume **Derivative** (**CalcElemVolumeDerivative**, into **pfx, pfy, pfz**)(L1342);
Unfolding derivatives and coordinates of local nodes of each element (got from previous 2 steps), putting all the derivatives and coordinates of every local node into 6 arrays, 3 for derivatives, 3 for coordinates, each 1 of which of for 1 dimension (into **dvdx/y/z(**$8n^3$**), x8n, y8n, z8n(**$8n^3$**)**)(L1345-L1354);



Erase and recalculate the **derem[](n^3)** (Which is confusing because the passed-in **determ[]**, the volume calculated in **CalcElemShapeFunctionDerivatives** is not used, and till here, it means the two outputs of **IntegrateStressForElems** when calculating **stress force** are not used) for each element node by:

$$\boldsymbol{determ}[i] = mesh.volo(i) * mesh.v(i)$$

$mesh.volo(i)$ is **reference volume** calculated at main function by **CalcElemVolume**, and $mesh.v(i)$ **relative volume(n^3)**, which is calculated in the latter part of previous iteration of the whole computation loop, the initial **relative volume** is 1.0 for each element node.(L1356)

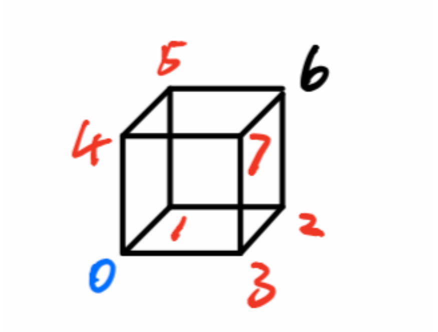


After checking the **hourglass efficient** initialized in main function is positive, compute **FB Hourglass Force** for each element node using **determ[], hourglass efficient, coordinates of all local nodes of all element nodes, derivative for all local nodes of all element nodes** (**CalcFBHourglassForceForElems**)(L1364-L1366).

x. **CollectDomainNodesToElemNodes (const Index_t* elemToNode, Real_t elemX[8], Real_t elemY[8], Real_t elemZ[8])**
Take the input element node and unfold its local nodes's coordinates into arrays in 3D, after the loop in which this function is called, 3 arrays of coordinates of all local nodes of all element nodes are generated. The **elemX/Y/Z** is the output coordinates. (L819-L861)

xi. **CalcElemVolumeDerivative (Real_t dvdx[8], Real_t dvdy[8], Real_t dvdz[8], const Real_t x[8], const Real_t y[8], const Real_t z[8])**
Calculates the **volume derivatives** of the passed-in element cube. Each vertex on the element cube has a **volume derivative,** after the loop in which this function is called, 3 arrays of derivatives of all elements in 3D are generated. and each of them are determined by the coordinates of vertexes except its diagonal vertex, so 6 nodes' coordinates are passed into **VoluDer to** calculate one derivative, and totally 8 **VoluDer** are called. **dvdx/y/z** are the output **volume derivatives**. (L893-L933)

xii. **VoluDer(const Real_t x0, const Real_t x1, const Real_t x2, const Real_t x3, const Real_t x4, const Real_t x5, const Real_t y0, const Real_t y1, const Real_t y2, const Real_t y3, const Real_t y4, const Real_t y5, const Real_t z0, const Real_t z1, const Real_t z2, const Real_t z3, const Real_t z4, const Real_t z5, Real_t* dvdx, Real_t* dvdy, Real_t* dvdz)**
The volume derivative is in 3D, each dimension determined by 2 other dimension's coordinates of the 6 parameter nodes:



$$dvdx = (y1 + y2) * (z0 + z1) - (y0 + y1) * (z1 + z2) + (y0 + y4) * (z3 + z4) - (y3 + y4) * (z0 + z4) - (y2 + y5) * (z3 + z5) + (y3 + y5) * (z2 + z5);$$

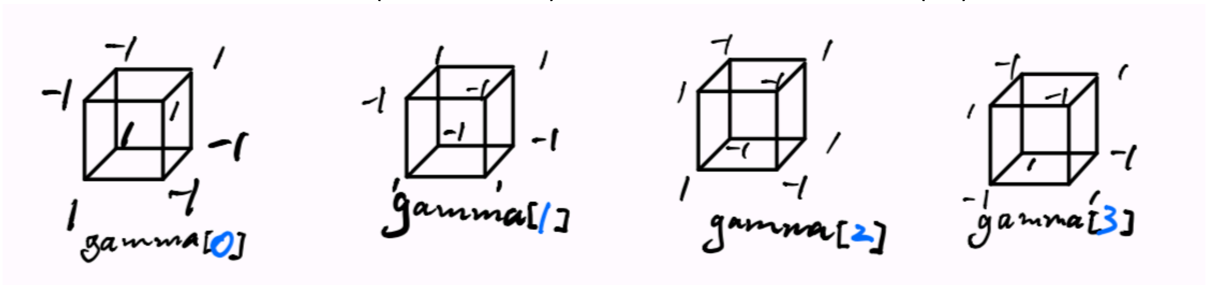
$$dvdy = -(x1 + x2) * (z0 + z1) + (x0 + x1) * (z1 + z2) - (x0 + x4) * (z3 + z4) + (x3 + x4) * (z0 + z4) + (x2 + x5) * (z3 + z5) - (x3 + x5) * (z2 + z5);$$

$$dvdz = -(y1 + y2) * (x0 + x1) + (y0 + y1) * (x1 + x2) - (y0 + y4) * (x3 + x4) + (y3 + y4) * (x0 + x4) + (y2 + y5) * (x3 + x5) - (y3 + y5) * (x2 + x5);$$

And output the value of each derivative divided by 12. (L863-L891)

xiii. **CalcFBHourglassForceForElems (Real_t determ[], Real_t hgcoef)**
Compute Hourglass Force for all element nodes. Same as **stress force,** even though it is said to be for **element nodes(n^3).**, note that finally

the **hourglass force** is still for every **node(**$(n+1)^3$**)** in the mesh (accumulated on **fx, fy, fz**).
It first constructs 4 **gamma cubes** with 1 or -1 factor on each vertex(L1142-L1173) and for each element node(n^3):

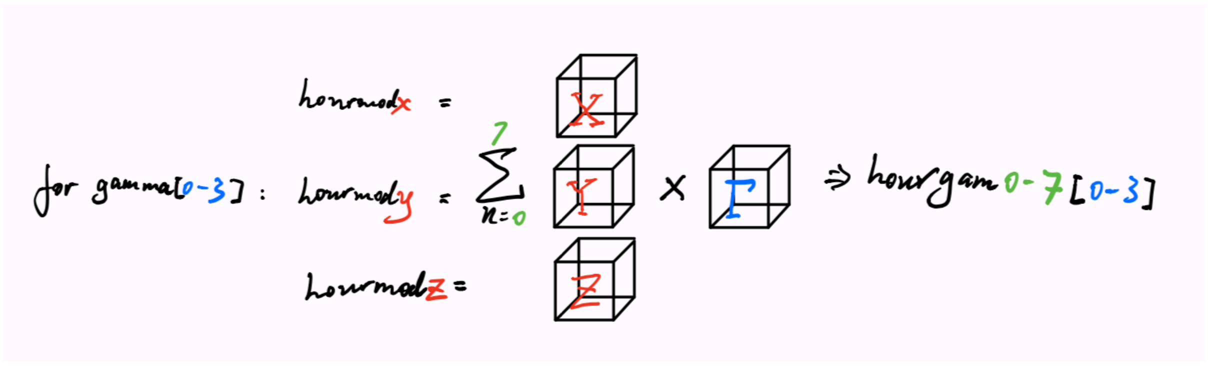


Define
$\boldsymbol{volinv} = 1/\boldsymbol{determ}[]$;(L1182)
for each gamma cube(4):
Compute **hourmodx/y/z** the hourglass node in 3D, with each dimension's **hourmod (**take x as example) defined as the sum of products of coordinate and gamma factor**:**

$$\boldsymbol{hourmodx} = \sum_{i=0}^{7} x_i * gamma_i\,, (i\ is\ the\ cube\ position);\text{(L1186-L1202)}$$

And Compute **Hourglass Mode(8)** for each position of the cube:

$$\boldsymbol{hourgam}_i = gamma_i - \boldsymbol{volinv} * (\boldsymbol{dvdx_i} * \boldsymbol{hourmodx} + \boldsymbol{dvdy_i} * \boldsymbol{hourmody} + \boldsymbol{dvdz_i} * \boldsymbol{hourmodz}), (i\ is\ the\ cube\ position);\text{(L1204-1234)}$$



Extract the element's **sound speed,** which is calculated in the **element part** of the whole computation of last iteration. Initial value is 0;(L1240)
Extract the element's **mass**, which is calculated through CalcElemVolume during initialization of main function;(L1241)
Calculate the cubic root of **derterm[](volume13)**;(L1242)
Calculated the **coefficient of current element** by:

$$\boldsymbol{coefficient} = -\boldsymbol{hourg}(the\ passed - in\ hourglass\ coefficient) * \mathbf{0.01} * \mathbf{sound\ speed} * \boldsymbol{mass} / \boldsymbol{volume}\mathbf{13};\text{(L1280)}$$

Compute the **hourglass force** in 3D for current element through **coefficient, hourglass mode(8),** the **velocity(**$(n+1)^3$**) of each local node(8)**(Calculated later in the **Nodal Part**, with an initial value of 0)(CalcElemFBHourglassForce), and accumulate them onto each node.(L1282-L1317)

xiv. **CalcElemFBHourglassForce (Real_t *xd, Real_t *yd, Real_t *zd, Real_t *hourgam0, Real_t *hourgam1, Real_t *hourgam2, Real_t *hourgam3, Real_t *hourgam4, Real_t *hourgam5, Real_t *hourgam6, Real_t *hourgam7, Real_t coefficient, Real_t *hgfx, Real_t *hgfy, Real_t *hgfz)**

For each dimesion(3):

For each gamma cube(4):

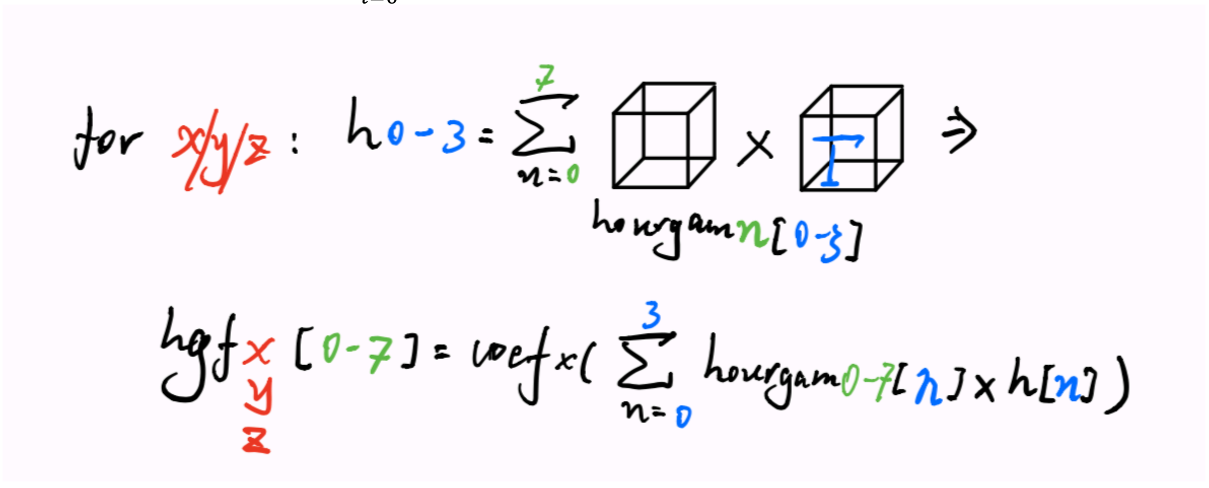
Multi-plus each vertex's **velocity** and **hourglass mode,** to get 4 **h** (L947-L969, L1003-L1025, L1060-L1082):

$$\boldsymbol{h}[\mathbf{0}-\mathbf{3}] = \sum_{i=0}^{7} \boldsymbol{xd}_i * \boldsymbol{hourgam}_i[\mathbf{0}-\mathbf{3}], (i \text{ is the cube position});$$

For each local node(8):

Multi-plus **hourglass modes** from each gamma cube with each **h** as the **hourglass force** for this element on this local node, and times the **coefficient** (L971-L1001, L1048-L1058, L1085-L1115):

$$\boldsymbol{hgfx}[\mathbf{0}-\mathbf{7}] = \boldsymbol{coef} * (\sum_{i=0}^{3} \boldsymbol{xd}_{0-7} * \boldsymbol{hourgam}_{\mathbf{0}-\mathbf{7}}[\boldsymbol{i}]), (i \text{ is the gamma cube index}).$$


for x/y/z : $h_{0-3} = \sum_{n=0}^{7}$ × ⇒
$hourgam_n[0-3]$
$hgf_{x,y,z}[0-7] = coef \times (\sum_{n=0}^{3} hourgam_{0-7}[n] \times h[n])$