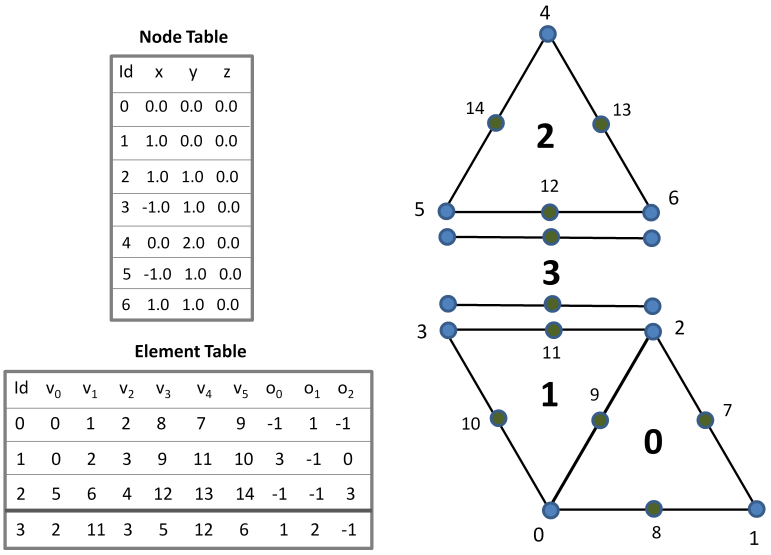
5
Data Structure

In order to implement efficient operations on mesh entities used in the simulation, a simple topological data structure is employed to represent the mesh. Since the GPU memory is limited, the data structure must not be complex so as to provide space for other simulation attributes that indeed require much global memory space. The proposed data structure is used for T3 or T6 meshes, and can be easily extended to support tetrahedral elements. We maintain two tables that describe the mesh. A table of nodes stores the node world-space position (x and y coordinates). A second table is used to represent the elements and adjacency relationships. The elements can by of two types: bulk elements or cohesive elements. Since the number of bulk elements remains unchanged during the entire simulation, we store the cohesive elements immediately after the bulk elements. For each bulk element, we store its nodal incidence; three node indices are used in a T3 mesh and six are used in a T6 mesh. For T6 meshes, the corner node indices are first followed by the mid-side indices. The right hand rule (counter clockwise) is used to define the order of nodal incidence. Another three values represent adjacent element indices that are opposite to each of the corner nodes. For cohesive elements, we store six node indices for a T6 mesh. The first three indices represent the three nodes of the corresponding facet of its adjacent bulk element with the smaller id, following the right hand rule. The next three indices belong to the adjacent facet of the second adjacent bulk element (with the greater id). Another two value are used to represent indices of both bulk elements that are opposite to each cohesive element's facets. Following the pattern, the first bulk element is opposite to the the first facet (and to the first three node indices) of the cohesive element. If a cohesive element is attached to a bulk element's facet, we update the opposite element of that facet to the cohesive element id on the element table. Nodes or elements are represented by their indices in the corresponding table. The last index in the table is not used for cohesive elements. Both node and element tables are stored in the global memory and are updated along the adaptive numerical simulation. Figure 5.1 shows an example of tables used in the data structure.



Node Table

Id	x	y	z
0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	1.0	0.0
3	-1.0	1.0	0.0
4	0.0	2.0	0.0
5	-1.0	1.0	0.0
6	1.0	1.0	0.0

Element Table

Id	v_0	v_1	v_2	v_3	v_4	v_5	o_0	o_1	o_2
0	0	1	2	8	7	9	-1	1	-1
1	0	2	3	9	11	10	3	-1	0
2	5	6	4	12	13	14	-1	-1	3
3	2	11	3	5	12	6	1	2	-1

Figure 5.1: Mesh parameters data structure of a T6 mesh.

A finite element analysis maintains a set of simulation attributes attached to nodes and elements. We maintain such attributes in global, constant, or texture memory depending on their memory size and dynamics during the simulation. In global memory, we store the attributes that change throughout the entire simulation. The nodes have associated displacements, velocities, accelerations, and forces (internal and cohesive), which are updated at every timestep. Stresses and strains evaluated at the nodes are updated only in a number of timesteps. When facets are checked for possible fractures, the fractured facets in the element attributes store which facets have been fractured. Nodal mass and number of adjacent bulk elements are updated whenever the topology of the mesh changes. Finally, cohesive attributes such as tractions and separations are updated every step for each cohesive element. We store other node and element attributes that remain unchanged during the entire simulation, but require too much memory space, in textures. We cannot store these attributes in constant memory due to its limited memory space. Each element's stiffness and lumped mass matrix and each node's boundary conditions are stored in texture memory. Other attributes that are common to all nodes and elements, such as elastic and fracture material properties, are stored in constant memory. These attributes are stored in one table common to all elements and nodes and occupy little memory space. Since all threads in a warp access the same memory space in constant memory, there will be no bank conflicts.

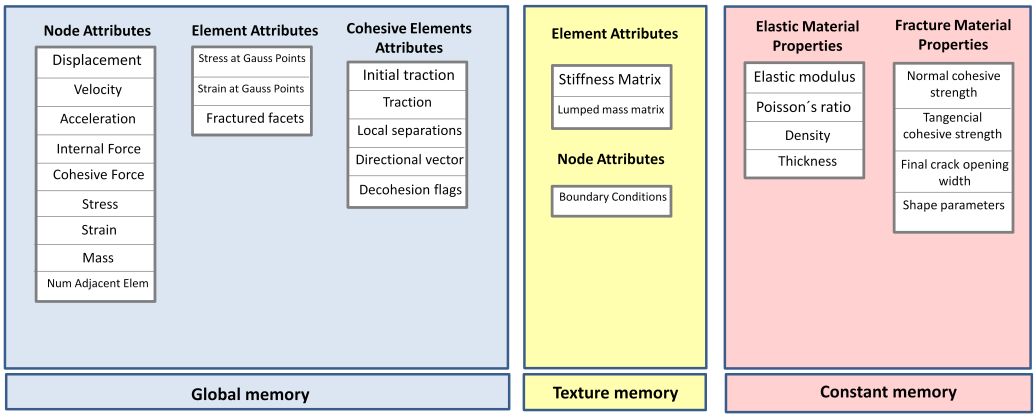


Figure 5.2: Simulation parameters data structure diagram of FEM model. Global memory is used for attributes that change throghout the simulation. Texture memory is used for attributes that are constant during the entire simulation, but occupy too much memory space. Constant memory is used for attributes that are constant during the entire simulation, but are common to all elements and node, therefore requiring few memory space.

5.1 Retrieving adjacency relationship

The node and element tables are enough to perform the previously stated algorithms and do not require too much memory usage. One key adjacency relationship for the insertion of cohesive elements is the set of adjacent element to a given node. With the described data structure, from an element, for each of its incident node, we can easily traverse the set of adjacent elements. Given the first node, we search the other node that precedes it in the order of incidence, and then access the corresponding opposite element and find the order of incidence of the node that had the previous element as its opposite. From both the element and node order, we obtain the next node in the incidence of the element and access its opposite element. From there, we repeat the procedure until we reach the element adjacent to the first one. Figure 5.3 illustrates the traversal algorithm from a given node. Cohesive elements can also be obtained by traversing around a node, since they are also stored in the element table and can be accessed by obtaining the opposite element for a given node.

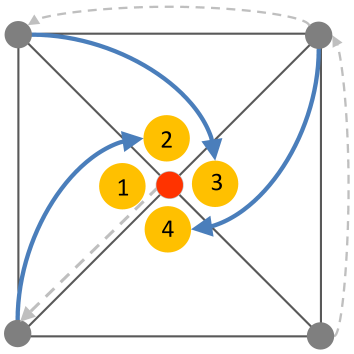


Figure 5.3: Traversal algorithm from a given element node using the proposed data structure.