Geometric and Topological Representation of Reinforcement Detail for Reinforced Concrete Framed Structures

By
Raghavan Kunigahalli
Software Developer
Bentley Systems, Inc.
690 Pennsylvania Drive
Exton, PA 19341, USA

Abstract: This paper presents a topological model for reinforcing elements that can be employed to generate equipment-level instructions for automated rebar cage fabricating and placing machines. The model presented in this paper supports extraction of information pertaining to hooks and splices in logarithmic running time during the construction/manufacturing stage. In addition, this model allows constant order insertion and deletion operation during the design and detailing stage. Hence, the described topological structure can be adopted for development of efficient computer-aided-design/computer-aided-construction (CAD/CAC) systems for RC structures. The reinfocement detail consisting of longitudinal and stirrup/tie elements are stored using Edge-Face based structures. Further, contact vertex tables are provided to maintain edge-to-edge and edge-to-contact-vertex relationships that can be utilized for Computer-Aided Process Planning (CAPP) in an automated environment.

1. INTRODUCTION

Reinforced Concrete (RC) framed structures are widely employed for construction of general, commercial, and industrial buildings as well as civil infrastructure facilities such as bridges. Even with such a wide application, issues related to geometric and topological representation, that includes reinforcement detail, have not been investigated. A unique, unambiguous, and complete geometric and topological representation scheme for reinforcement detail will enable development of efficient computer-aided systems for: (1) error-free rebar designing and detailing, (2) computer-aided construction process planning (CAPP), and (3) computer-aided manufacturing (CAM) of rebar cages and pre-cast concrete structural elements.

2. LITERATURE SURVEY

A description of various geometric and topological representation schemes and their characteristics for rigid solids is provided in (Requicha 1980). Classical *winged-edge* representation proposed by Baumgart (1974) represents a face as a sequence of edges comprising it, while the edges are specified as ordered pairs of their constituent vertices. A relational graph structure called *face adjacency graph* (FAG), represents object faces as nodes, whereas, edges and vertices are encoded into links and hyperlinks (Ansaldi et al. 1985). Another face-based model, called *symmetric data structure*, stores face-to-edge and vertex-to-edge relationships along with their inverses (Woo 1985). For

representation of structural objects of mixed dimension, Rossignac and O'Connor (1990) have proposed a scheme based on Selective Geometric Complex (SGC). It consists of a collection of mutually disjointed cells that are connected. Rossignac and Requicha (1991) have extended the domain of Constructive Solid Geometry (CSG) into Constructive Non-Regularized Geometry (CNRG) that support modeling of mixed-dimensional objects. Weiler (1987) has proposed the so-called *radial-edge structure* for structured object representation. A data structure for parametric representation within a Non-Manifold Topological modeling system is described in Chen et. al. (1993).

Shapira (1993) has proposed a Spatial Occupancy Enumeration (SOE) technique, using the *octree* structure to store information on building elements. Austin and Preston (1992) have proposed a data structure to store information pertaining to an individual R.C. beam component. Werner et al. (1993) have proposed a scheme similar to *radial-edge structure* to store geometric and topological information in a R.C. structure.

3. COMPARISON OF THE PROPOSED MODEL WITH EXISTING MODELS

Although *octree* decomposition technique proposed by Shapira (1993) provide easy access to any given point in an object, relationships between various parts of an object are not stored explicitly. Further, boundaries of *voxels* must match and interior of objects must be disjoint. In addition, modeling of elements with curved surfaces, such as reinforcing bars, is difficult and requires large amount of storage space. Proposed data structure by Austin and Preston (1992) employs a variant of the winged-edge data structure called *half-edge structure* to represent the boundary of a R.C. beam component. Reinforcing elements within a beam component, *restricts each individual bar to lie in either a vertical or horizontal plane parallel to the longitudinal axis of the beam component.* Modeling of reinforcing elements does not consider *loop* configurations and *hook* formations for stirrups in a beam component. Further, topological relationships between the longitudinal and stirrup/tie elements have not been incorporated. In addition, the representation of slab and column components in a R.C. framed structure has not been addressed. The proposed scheme by Werner et al. (1993) considers beam, slab, as well as column components. However, issues related to computer-based storage and manipulation of geometric and topological information have not been considered. Further, information related to reinforcement detail has not been incorporated.

Designing, rebar detailing, and automated rebar cage manufacturing require efficient topological operators to support activities such as: (1) insertion, deletion, and altering the spacing of both longitudinal and stirrup/tie reinforcing elements and (2) extraction of topological relationships for construction process planning and automated pre-fabrication of reinforcement cages. The representation scheme presented in this paper supports these activities by providing an efficient access to the geometric and topological information pertaining to the reinforcing steel bars in a given structural component of a R.C. framed structure. *Loop* configurations and *hook* formations for stirrup/tie reinforcing bars have been considered while developing the model presented in this paper. Example type definitions for data structures are provided using C-program syntax.

4. DESCRIPTION OF THE PROPOSED MODEL

4.1. Domain Specification

The representation scheme for reinforcement detail presented in this paper takes into account the detailing specifications for: (1) longitudinal straight bars and (2) stirrup/tie

bars stipulated in the American Concrete Institute Detailing Manual (ACI 1994). However, the following situations have not been included within the domain specification of this representation sheme: (1) bent (or cranked) longitudinal bars in beams, (2) spiral ties for circular columns, and (3) orientation of ties for rectangular columns that results in an interior angle other than 90^0, 135^0, or 180^0 at corner and/or hook locations.

4.2. Beam and Column Reinforcement Detail

Different configurations of longitudinal reinforcement, for various cross-sections along the length of a beam or a column component, between any two given structural joints *i* and *j* are possible due to: (1) variations in positive and negative bending moment and (2) splicing of bars. Let us define the portion of a beam or column having the same configuration of longitudinal reinforcement as a *region*. The geometric and topological information regarding reinforcement detail in a given beam or column component is stored in a structure called *Beam_Column_Region*. The information regarding: (1) boundary and reinforcement detail of a beam or a column component and (2) connectivity of the beam or column component to its adjoining structural joints is stored in a structure called *Beam_Column_Component*. C-definitions of *Beam_Column_Component* and *Beam_Column_Region* structures are provided below.

```
typedef    struct  beam_col_comp{
                        int                id;              /* identification */
                    Str_Joint*             i;  /* adjoining structural joints */
                    Str_Joint*             j;
                    Face_Table*            body;       /* B-rep of beam/column*/
          Beam_Column_Region*              reinforcement;  /* reinfocement detail */
                                        } Beam_Column_Component;
typedef    struct  beam_col_region {
               Beam_Column_Component*      parent;       /* parent pointer */
               Long_Circular_List*         long_bars;      /* longitudinal bars */
               Loop_List*                  stirrup_tie_bars;   /* stirrup/tie bars */
          struct  beam_col_region*         sibling;
                                        } Beam_Column_Region;
```

The *Beam_Column_Component* structure contains: (1) beam identification number, (2) pointers to two adjacent structural joints, (3) a pointer to a *Face_Table* structure that stores the boundary represenation (B-rep) of a beam/column component itself, and (4) a pointer to a list of *Beam_Column_Region* structures. If the number of regions in a given beam or column component is k>1, then the region adjacent to the *Str_Joint i* in the *Beam_Column_Component* data structure is considered as the first region and the one adjacent to *Str_Joint j* is considered as the k^{th} region. Parent pointer included in the *Beam_Column_Region* structure enables faster identification of relative location of a region with respect to a complete R.C. framed structure. *Long_Circular_List* and *Loop_List* store geometric and topological information pertaining to, longitudinal reinforcement and stirrup/tie reinforcement, respectively. Structure definitions for these supporting structures are provided in Appendix 1.

The shape of the boundary of each longitudinal or stirrup/tie reinforcing bar matches to that of a solid *cylindrical* primitive shown in Figure 1a. A solid cylindrical

primitive consists of: (1) 3 faces among which one is longitudinal and two are end faces, (2) 3 edges out of which one is *identified* (edge *a* in Figure 1a), and (3) two vertices that bound the three edges. To capture topological relationships between a longitudinal bar and a stirrup/tie bar, the longitudinal face of a longitudinal bar is partitioned into 8 equal parts. This results in 8 *boundary-edges* along the longitudinal face, depicted by numbered points along the boundary of an end-face, of a longitudinal bar as shown in Figure 1b. The following four cases, depicted in Figure 1c, of topological relationships between a longitudinal bar and a stirrup/tie bar are considered in the domain of this topological representation scheme: (1) tangential contact between a longitudinal bar and a stirrup/tie bar which results in one edge-to-edge contact vertex, (2) 90^0 bend of a stirrup/tie bar at a corner or a hook location which results in three edge-to-edge contact vertices, (3) 135^0 bend of a stirrup/tie bar at a hook location which results in four edge-to-edge contact vertices, and (4) 180^0 bend of a stirrup/tie bar at a hook location which results in five edge-to-edge contact vertices.

4.2.1. Longitudinal Reinforcement Detail

The geometric and topological information regarding an individual reinforcing bar in a given region *i* is stored using an Edge-Face structure. A typical definition of an Edge-Face structure for a longitudinal bar is provided below.

```
typedef   struct  e_f_long{
          Beam_column_Region*     parent;
          Bar_Type                bar_type;   /* enumerated type */
          int                     bar_number;  /* between corner bars */
          double                  long_face;     /* face information */
          double                  end_face;
          Edge_Type*              interior_edge; /* edge information */
   struct e_f_long*               Boundary_Edge_Type[8] self_pointers;
                                  /*to store edge-to-edge relationships */
          Splicing_Rebar*         spliced_bar;
                                        /* spliced bar information */
                        } Edge_Face_Longitudinal;
```

The *Edge_Face_Longitudinal* structure contains a pointer to the parent region and an enumerated type to uniquely identify individual longitudinal bar in a given region. Let A, B, C, and D denote the four corner longitudinal bars in a given region *i*. Further, let there be *p* bars between A and B, *q* bars between B and C, *r* bars between C and D, and *s* bars between D and A. An enumerated type {A, B, C, D, AB, BC, CD, DA} along with integer values (bar numbers) in the range of 1 to *p*, 1 to *q*, 1 to *r*, and 1 to *s* are utilized to identify a particular longitudinal bar. Geometric information on individual longitudinal bar in a given region is stored using two floating point numbers and a pointer to an *Edge_Type* data structure. The *Boundary_Edge_Type* is an array of 8 elements corresponding to each of the 8 boundary-edges, depicted in Figure 1c, along the longitudinal face of a longitudinal bar. This array stores *self-pointers* and facilitates maintaining edge-to-edge topological relationships between a longitudinal bar and stirrup/tie bars in a given region *i*. The exact methodology to store such relationships will

be described in the next section that addresses the modeling scheme for the stirrup/tie reinforcement detail. If there is a splicing of a longitudinal bar, the *Splicing_Rebar* structure stores geometric and topological information pertaining to the lap-spliced bars.

The *Edge_Face_Longitudinal* structures of longitudinal bars in a given region *i* are stored in a circular list called *Long_Circular_List*. A specification for: (1) ordering of the *Long_Circular_List* and 8 boundary-edges of a longitudinal bar and (2) labeling of longitudinal bars using the enumerated elements of Bar_Type is imperative to ensure an *unambiguous* representation. Let us consider a *counter-clockwise* traversal of the list of beam components enclosing a slab. During such a traversal, regions pertaining to beam components that are adjacent to the bottom and right edges of the slab will have bar labeled *A* located at the *bottom-left* corner and bars *B*, *C*, and *D* following in a *counter-clockwise* direction. The *Long_Circular_List* in this case is also ordered *counter-clockwise*. On the other hand, regions pertaining to beam components that are adjacent to the top and left edges of the slab will have bar labeled *A* located at *bottom-right* corner and the *Long_Circular_List* ordered *clockwise*. Similarly, ordering of 8 boundary-edges along the longitudinal face of each longitudinal bar in a given region employs the following scheme: (1) ordering for the longitudinal bars belonging to beam components adjacent to bottom and right edges of the slab will begin at the boundaries that correspond to the *positive* X and Y-coordinate axes, respectively, and proceeds in a *counter-clockwise* direction and (2) ordering for the longitudinal bars belonging to beam components adjacent to top and left edges of the slab will begin at the boundaries that correspond to the *negative* X and Y-coordinate axes, respectively, and proceeds in a *clockwise* direction.

4.2.2. Stirrup/Tie Reinforcement Detail

There can be more than one loop of stirrup/ties to resist the shear force at a given cross-section of a beam or column component. Further, due to variations in the shear force, the spacing of loops, and in some cases the configurations of the loops of stirrups/tie bars themselves, may vary along the length of a given *region*. The *Loop_List* in the *Beam_Column_Region* data structure allows variable spacing and configurations for loops of stirrup/tie bars while maintaining the geometric and topological information pertaining to stirrup/tie reinforcement detail. A typical C-definition for a *Loop-List* data structure is provided in Appendix I. *Edge_Face_Stirrup_Ties* structure present in the *Loop_Configuration* structure of a *Loop_List* stores information pertaining to a stirrup/tie bar. A typical definition for *Edge_Face_Stirrup_Ties* structure is provided below.

```
typedef   struct e_f_stirrup_tie   {
              double                  long_face_info;     /* face information */
              double                  end_face_info;
              Edge_type*              edge_info;  /* identified edge */
              Hook_Location*          hook_info;  /* hook information */
              Stirrup_Tie_Hoop*       hoop_type;  /* for hoop type stirrups */
                                      } Edge_Face_Stirrup_Ties;
```

The *Edge_Face_Stirrup_Ties* structure stores geometric information pertaining to the vertices and faces using floating point numbers. The information regarding the *identified* edge of a stirrup/tie reinforcing bar is stored using a pointer to an Edge_Type

structure. The *Hook_Location* structure stores the edge-to-edge and edge-to-contact-vertex topological relationships between a stirrup/tie bar and a longitudinal bar at the location of a *standard hook*. A typical C-definition for the *Hook_Location* data structure is provided in the Appendix I. It consists of two arrays, of 5 elements each, one for the initial hook and the other for the terminal hook of a stirrup/tie bar. In case of a 90^0 hook, last two elements of the array will contain NULL pointers and in case of a 135^0 hook, last element of the array contains a NULL pointer. Topological information pertaining to a *hoop* type stirrup/tie bar is stored in the *Stirrup_Tie_Hoop* structure. A typical definition of the *Stirrup_Tie_Hoop* data structure is provided below.

```
typedef   struct stirrup_tie_hoop {
  Topological_Relationships*    Corner_Location[3];    corner_info;
                                        /* corner information */
        Topological_List    one_two; /* intermediate bars */
        Topological_List    two_three;
        Topological_List    three_four;
        Topological_List    four_one;
                              } Stirrup_Tie_Hoop;
```

The *Corner_Location* type in the *Stirrup_Tie_Hoop* data structure is an array of 3 elements that stores topological relationships occurring due to case(ii) of Figure 1c. The variables *one_two*, *two_three*, *three_four*, and *four_one* store topological relationships, of case (i) of Figure 1c, that occur between a corner and a hook location or two corner locations. Typical definitions for the secondary structures are provided in Appendix I.

4.2.3. Slab Reinforcement Detail

Reinforcement detail pertaining to a slab component, designed using one-way and two-way slab theories, consists of longitudinal reinforcing bars to resist: (1) positive bending moment, (2) negative bending moment, and (3) torsion at the four corners. The reinforcement for positive bending moment and torsion are typically provided in two layers of bars, namely, upper and lower, that are placed along the two principal orthogonal directions x and y. There exists a boundary edge-to-edge contact between a given longitudinal bar and every other longitudinal bars placed in the other (orthogonal) direction. Negative bending moment reinforcement for a slab normally results in edge-to-edge contacts with longitudinal bars near the top-faces of beams enclosing the slab. The geometric and topological information pertaining to a reinforcing bar in a slab component is stored in an *Edge_Face_Slab* structure.

```
typedef struct e_f_slab   {
        Bar_Type                bar_type;           /* enumerated type */
        Bar_Direction           bar_direct;         /* enumerated type */
        double                  long_face_info;     /* face information */
        double                  end_face_info;
        Edge_Type*              interior_edge;   /* edge information */
struct e_f_slab* Boundary_Edge_Type_Slab[2]      self_pointer;
                    /* self pointers to to support edge-to-edge relationships */
        Slab_Topological_Relationships*   slab_topo;
                                        /* reinforcement topology */
                    } Edge_Face_Slab;
```

The *Bar_Type* in the *Edge_Face_Slab* structure is an enumerated type {UPPER, LOWER, NEGATIVE} that enables identification of appropriate topological relationships that need to be stored. The direction of a given longitudinal bar is stored using the *Bar_Direction* enumerated type defined as {X-DIRECTION, Y-DIRECTION}. The geometric information on faces and the interior edge is stored using floating point numbers and a pointer to an *Edge_Type* structure. In case of topological relationships between two reinforcing bars of a slab component, only two types of boundary edge-to-edge relationships occur: (1) upper to lower bar contact that results in a relationship between boundary edge_types 3 and 7 of Figure 1b and (2) lower to upper bar contact that results again in a relationship between boundary edge_types 3 and 7 shown in Figure 1b. Hence, Boundary_Edge_Type_Slab in the Edge_Face_Slab data structure consists of an array of only two elements that contain self-pointers. The topological information regarding reinforcing elements is stored in a *Slab_Topological_Relationships* data structure. A typical definition of *Slab_Topological_Relationships* structure is provided below.

```
typedef struct  slab_type {
            Reinforcement_Type          r_type;     /* enumerated type */
        union     {
          struct  pos_torsion    {
            Edge_Face_Slab**            edge_to_edge;
            Contact_Vertex_Table*       slab_contact_vertex;
                                } Positive_BM_And_Torsional;
          struct  negative       {
            Edge_Face_Longitudinal**    edge_to_edge;
            Contact_Vertex_Table*       slab_beam_contact_vertex;
                                } Negative_BM;
                    } Reinforcement_Variant;
                            } Slab_Topological_Relationships;
```

The Reinforcement_Type in the *Slab_Topological_Relationships* structure is an enumerated type {POSITIVE, TORSIONAL, NEGATIVE} provided to identify the variant structures. The *Edge_Face_Slab* structures of longitudinal bars in a slab are arranged as lists ordered in the two principal directions X and Y. Two such lists in orthogonal directions, that are confined within the boundary of a slab component, give rise to a rectangular grid structure. Thus, positive bending moment and torsional reinforcement in a slab component results in a total of five grid structures. Negative bending moment reinforcement for a slab component forms four lists, two *X-lists* and two *Y-lists*, of *Edge_Face_Slab* structures. C type definitions for *Slab_Grid_Reinf_List*, *Slab_Torsional_List*, and *Edge_Face_Slab_List* are provided in Appendix I. A type definition of the data structure for a slab component is provided below.

```
typedef struct  slab {
                Face_Table*                 body;
                Slab_Grid_Reinf_List*       positive_BM;
                Slab_Torsional_List*        torsion;
                Edge_Face_Slab_List*        negative_BM;
                    } Slab_Component;
```

5. STORAGE AND TRAVERSAL PERFORMANCE

A review of the presented model indicates that geometric information, which requires *binary* representation of floating point numbers, is stored only once. For instance, geometric information pertaining to longitudinal and stirrup/tie reinforcing bars is stored only once in the respective *Edge-Face* data structures. The topological information is stored by employing pointers and structures of pointers to and from the data structures containing the basic geometric information. Insertion or deletion of a reinforcing bar requires *creating* and *freeing* of an edge-face structure and *traversal* of *Long_Circular_List* and/or *Loop_List*. Since the size of these two lists are limited to the number of bars within a given region, insertion or deletion of a reinforcing bar can be performed in constant order time *O(1)*. Further, storing of structural joints by a sorted order of their identification numbers will ensure determination of edge-to-edge and edge-to-contact-vertex relationships among the reinforcing bars in *O(log n)* time complexity.

6. CONCLUSION

A novel approach to represent geometric and topological information pertaining to reinforcement detail in a RC framed structural component is presented. The representation scheme supports geometric queries related to: (1) spacing of longitudinal elements in a given cross-section, (2) spacing of stirrups along a given beam section between two structural joints, (3) extraction of topological relationships between longitudinal and stirrup/tie elements, and (4) extraction of topological information pertaining to torsional and bending moment reinforcement in a structural slab component. Representation schemes that support such queries are imperative to develop efficient automated systems for computer-aided rebar pre-fabrication and erection.

APPENDIX I -- Type Definitions for Supporting Data Structures

```
typedef   struct e_f_splice {
            Edge_Face_Longitudinal*      splicing_bar;
            Edge_Face_Longitudinal**     contact_location;
                                  } Splicing_Rebar;
typedef   struct loop_config  {
            Edge_Face_Stirrup_Ties*  loop;
            Edge_Face_Stirrup_Ties*  next;
                                    } Loop_Configuration;
typedef   struct loop_list  {
            Beam_Column_Region*    parent_region;
            Loops_Configuration*  current;
            Loops_Configuration*  next;
                                  } Loop_List;
typedef   struct contact_vert  {
                Point_Type* contact_vertex;
          struct contact_vert* next;
                                    } Contact_Vertex_Table;
typedef   struct topo_relation {
          Edge_Face_Longitudinal**    edge_to_edge;
          Contact_Vertex_Table*     contact_vertex;
                                    } Topological_Relationships;
```

```
typedef   struct topo_list   {
                Topological_Relationships*  current;
                Topological_Relationships*  next;
                                    } Topological_List;
typedef   struct hook   {
                Topological_Relationships*  hook_initial[5];
                Topological_Relationships*  hook_terminal[5];
                                } Hook_Location;
typedef   struct e_f_slab_list {
                Edge_Face_Slab*  reinforcement;
                Edge_Face_Slab*  next;
                                        } Edge_Face_Slab_List;
typedef   struct grid_list   {
                Edge_Face_Slab_List   upper;
                Edge_Face_Slab_List   lower;
                                    } Slab_Grid_Reinf_List;
typedef   struct tor_reinf   {
                Slab_Grid_Reinf_List* bottom_left;
                Slab_Grid_Reinf_List* bottom_right;
                Slab_Grid_Reinf_List* top_right;
                Slab_Grid_Reinf_List* top_left;
                                    } Slab_Torsional_List;
```

7. REFERENCES

1. ACI, "ACI Detailing Manual -1994," Publication SP 66(94), American Concrete Institute, Detroit, MI, 1988.
2. Ansaldi, S., Floriani, L. D., and Folcidieno, B., "Geometric Modeling for Solid Objects by Using a Face Adjacency Representation," *ACM/SIGGRAPH Computer Graphics*, Vol. 19, No. 3, pp. 131-139, 1985.
3. Austin, M. A. and Preston, J. L., "Solid Modeling of R. C. Beams: I. Data Structures and Algorithms," *ASCE Journal of Computing in Civil Engineering*,, Vol. 6, No. 4, pp. 389-403, 1992.
4. Baumgart, B., Geometric Modeling for Computer Vision, Ph.D. Thesis, Computer Science Department, Stanford Universitý, 1974.
5. Chen, J., Gursoz, E. L., and Prinz, F. B., "Integration of Parametric Geometry and Non-Manifold Topology in Geometric Modeling," *2nd ACM Symposium on Solid Modeling and Applications*, Montreal, Canada, pp. 53-64, 1993.
6. Requicha, A. A. G., "Representation for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, Vol. 12, No. 4, pp. 439-465, 1980.
7. Requicha, A. A. G. and Rossignac, J. R., "Solid Modeling and Beyond," *IEEE Computer Graphics and Applications*, September, pp. 31-44, 1992.
8. Rossignac, J. R. and M. A. O'Connor, "SGC: A Dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries," *Geometric Modeling for Product Engineering*, Wozny et al. (ed), North-Holland, pp. 145-180, 1990.
9. Rossignac, J. R. and Requicha, A. A. G., "Constructive Non-Regularized Geometry," *Computer-Aided Design*, Vol. 23, No. 1, pp. 21-32, 1991.

10. Weiler, K., "The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling," *Geometric Modeling for CAD Applications*, Wozny et al. (ed.), North-Holland, pp. 1-//, 1986.
11. Woo, T. C., "A Combinatorial Analysis of Boundary Data Structure Schemata," *IEEE Computer Graphics and Applications*, March, pp. 19-27, 1985.

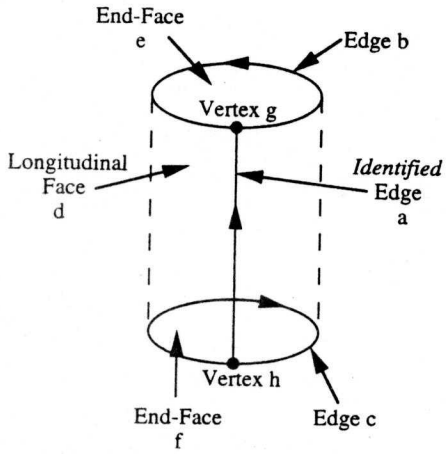

Figure 1a: A Cylindrical Primitive

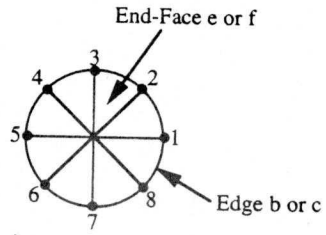

Figure 1b: Partition of Longitudinal Face of a Longitudinal Bar

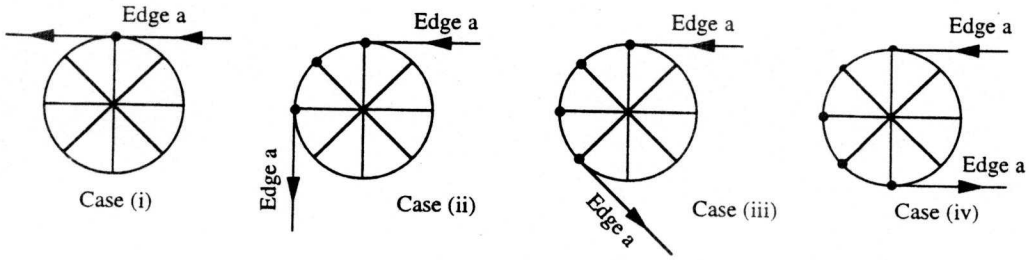

Figure 1c: Four Possible Cases of Topological Relationships