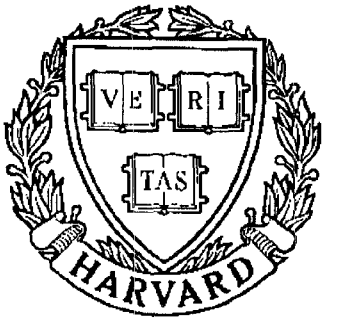
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Solid Modeling of RC Beams
Part I: Data Structures and Algorithms
Part II: Computational Environment

by M.A. Austin and J.L. Preston

TR 91-91r1

To appear in ASCE
Journal of Computing in Civil Engineering

SOLID MODELING OF RC BEAMS
PART 1 : DATA STRUCTURES AND ALGORITHMS

By M.A. Austin, A.M. ASCE[1], and J.L. Preston[2]

ABSTRACT

This paper describes data structures and algorithms used in a solid modeling based methodology for the interactive design and analysis of reinforced concrete (RC) beam structures. Data structures are described for steel reinforcing bar trajectories, and a three-dimensional boundary representation of the concrete beam solid. After the concrete beam model is setup, the envelope of ultimate flexural strength is obtained by slicing the beam solid at selected locations, and extracting the cross section. Algorithms are given for the slicing process, plus the calculation of cross section properties and ultimate flexural strength.

Keywords : Solid Modeling, Computer-Aided Design, Reinforced Concrete.

[1] Assistant Professor, Department of Civil Engineering and Systems Research Center, University of Maryland, College Park, MD 20742, USA.

[2] Graduate Research Assistant, Department of Civil Engineering and Systems Research Center, University of Maryland, College Park, MD 20742, USA.

INTRODUCTION

During the past ten years, the term Computer-Aided Structural Design has come to mean the use of computers to assist the trained engineer as he or she takes the functional specifications of an initial concept, and transforms them into a constructed three-dimensional structure. An engineer's perception of how and when a computer may provide assistance is strongly aligned with the state of computer technology. With rapid advances in the computational and graphical power of engineering workstations currently taking place, the use of workstations to merely visualize a structure, or to provide basic analysis of a completed design may soon be deemed inadequate. Instead, engineers may expect significant computer assistance in (a) the synthesis of preliminary design alternatives, (b) tradeoff analysis of design performance versus costs for design alternatives. (c) strategies of construction, (d) analysis of partially constructed structures, and (e) detailed analysis, checking, and life-time cost analysis of the final design. To ensure that better designs are produced in less time, all of the above tasks (a)-(e) should operate within a single integrated computing environment.

Unfortunately, the technology needed to implement steps (a)-(e) is still in its infancy (Westerberg 1989). The heart of the software integration problem lies in the formulation of: (a) data representations for structures that contain enough information to be useful at multiple levels of abstraction within the design process, and (b) algorithms that can operate on these models for analysis and design evaluation. Subrahmanian et al. (1989) indicate that an emerging theme in current research efforts is that such a model will make extensive use of information and algorithms that remain invariant across range of applications within a problem domain. Within the domain of structural design, notions of `geometry` and

`topology` are the lowest levels of data abstraction, and remain invariant across applications. Algorithms for creating and manipulating instances of these geometric representations fall into the general area of `solid modeling`.

SOLID MODELING

Solid modeling is the process of building and analyzing geometrically complete representations of points, lines, polygons, and three-dimensional solid objects. In a functional sense, a model is said to be geometrically complete if it is possible to answer questions about the geometry of an object (e.g., what is the area of a polygon), or perhaps the relationship among various objects (e.g., compute the intersection of a plane and three-dimensional solid), with algorithmic operations that act on the underlying data representation, and without taxing the inferential capabilities and pattern recognition skills of the human user. For this to be possible, objects must be closed, orientable, non-self-intersecting, bounding, and connected. Each of these conditions may be mathematically stated in terms of the topological sufficiency of the object; for a detailed discussion, see Weiler (1986) or Requicha (1980). It is regretable that the data representations of many engineering objects, including engineering drawings and wireframe models, fail to satisfy the conditions of geometric completeness.

OBJECTIVES AND SCOPE

The long term goals of this research are to use ideas founded in solid modeling as a basis for constructing new algorithms, and developing interactive software for the design and analysis of reinforced concrete structures. The work is motivated by potential applications to the design of reinforced beam structures, concrete frame structures, prestressed concrete

beams and frames, and structures assembled from precast components.

From the beginning, it was evident the preliminary research could be approached in one of two ways. The first possibility was to focus on the formulation of a conceptual methodology for the use of solid modeling in structural assemblies; this problem has been investigated by Powell and Martini (1990), and Zamanian et al. (1991). The interests of the writers focused instead on how solid modeling techniques would work in practice. This led to a second strategy of actually implementing a small-scale software system using techniques from solid modeling.

The problem selected for the prototype implementation was analysis and design of a reinforced concrete beam structure. The authors felt that the design specifications for the prototype system should address four features, namely:

[1] To provide for the interactive generation of solid models of the beams, with arbitrarily shaped cross-sections, possibly containing holes. This was to be implemented by means of interactive drawing and sweeping algorithms.

[2] Create an editing environment that allows for the placement of reinforcing bars both within the beam at a three-dimensional level, and within cross-sections at a two-dimensional level. Practical design considerations dictate that reinforcing bars be contained completely inside the concrete solid. The checking of these geometric constraints must be enforceable at both the two- and three-dimensional levels.

[3] Develop interactive software tools that allow the user to freely move between a two-dimensional design and analysis environment where cross-sections are the highest level of objects, and a three-dimensional environment where work is done on complete beams.

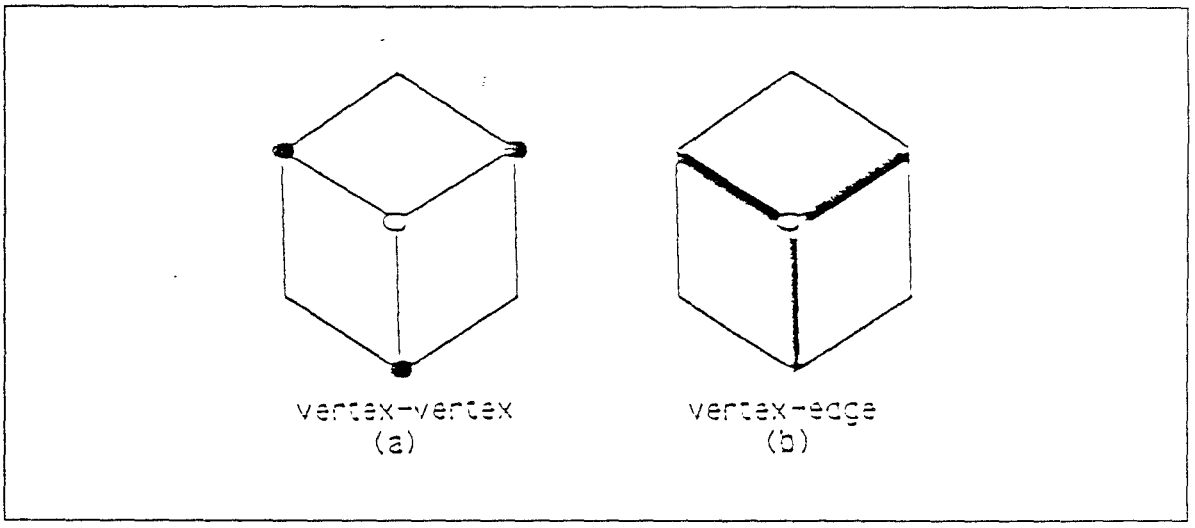


FIG. 1. - Examples of Adjacency Relations

[4] Provide analytical tools to determine the ultimate flexural strength of the beam along its length.

The purpose of this paper is to describe the underlying data structures and algorithms formulated for the prototype system. Details of the software development, and a design example are presented in the companion paper (Preston 1992).

BOUNDARY REPRESENTATIONS AND ADJACENCY RELATIONS

Boundary models represent solids by a set of boundary surface faces, which in turn, are bounded by sets of edges. The information in the boundary represenation is stored as a hierarchical data structure. It links points together to form edges, and then groups edges to form the bounding faces of the solid. The entire organization of the data structure serves to maintain the topological connectivity and coherence of these points; the only true geometric data is the point coordinates.

An important concept in boundary representation models is that of adjacency rela-

tions. Adjacency relations are the conceptual links among the different levels of the model hierarchy, and between neighboring elements at the same level. For example, each edge is adjacent to two faces upward in the hierarchy, to two vertices downward in the hierarchy, and to some number of other edges at the same level. Nine adjacency relationships for boundary models have been identified; they are vertex-vertex, vertex-edge, vertex-face, edge-vertex, edge-edge, edge-face, face-vertex, face-edge, and face-face. For example, the vertex-vertex adjacency relationship - see Fig. 1a - refers to all of the the vertices separated by one edge from a given vertex. Similarly, vertex-edge relationships - see Fig. 1b - refer to the ring of edges that surround a given vertex. Weiler (1986) states "If a topological representation contains enough information to recreate all nine of these adjacency relations without error or ambiguity, it can be considered a sufficient adjacency topology representation." In other words, a data representation must provide for the determination of all of the adjacency relations in order to be geometrically complete.

HALF-EDGE DATA STRUCTURE AND HIERARCHY

The backbone of our solid modeling system is the complex three dimensional assembly called the halfedge data structure (Mantyla 1988). This section describes the halfedge data structure, and how we have modified it for the representation of concrete beams.

The name `halfedge` stems from the way the edge between two intersecting faces is stored. Instead of representing the edge with a single line segment going from one point to another, two halfedges are linked together to form a complete edge, as shown in Fig. 2. The edge link positioned between the two halfedges allows neighboring faces to access information on each other. Matching halfedges are oriented in opposite directions. This provides a means of orienting the faces of the solid, so that notions of the inside and outside

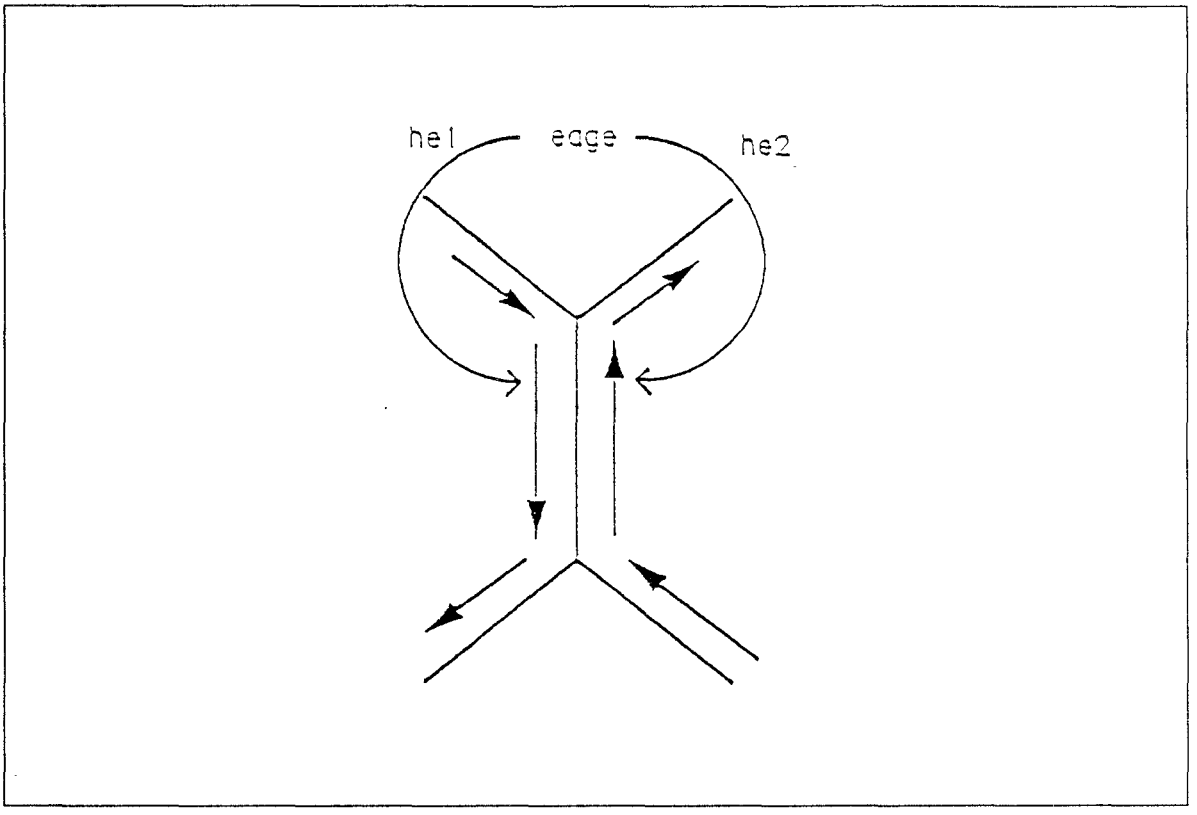


FIG. 2. The Basic Halfedge and Edge

of the solid are clearly defined.

The modified halfedge hierarchy is shown in Fig. 3. It employs eight types of data nodes for the geometrically complete representation of general polyhedral solids. They are `solid`, `face`, `loop`, `halfedge`, `edge`, `vertex`, `point` and `loneface`. The hierarchy allows for the rapid and unambiguous determination of all nine adjacency relations by means of the parent-child information stored at every level of the data structure, and by means of the edge link between corresponding halfedges.

The `solid` node is the root node of the data structure hierarchy; access to all of the geometric information of the model is available through a pointer to the solid node. The

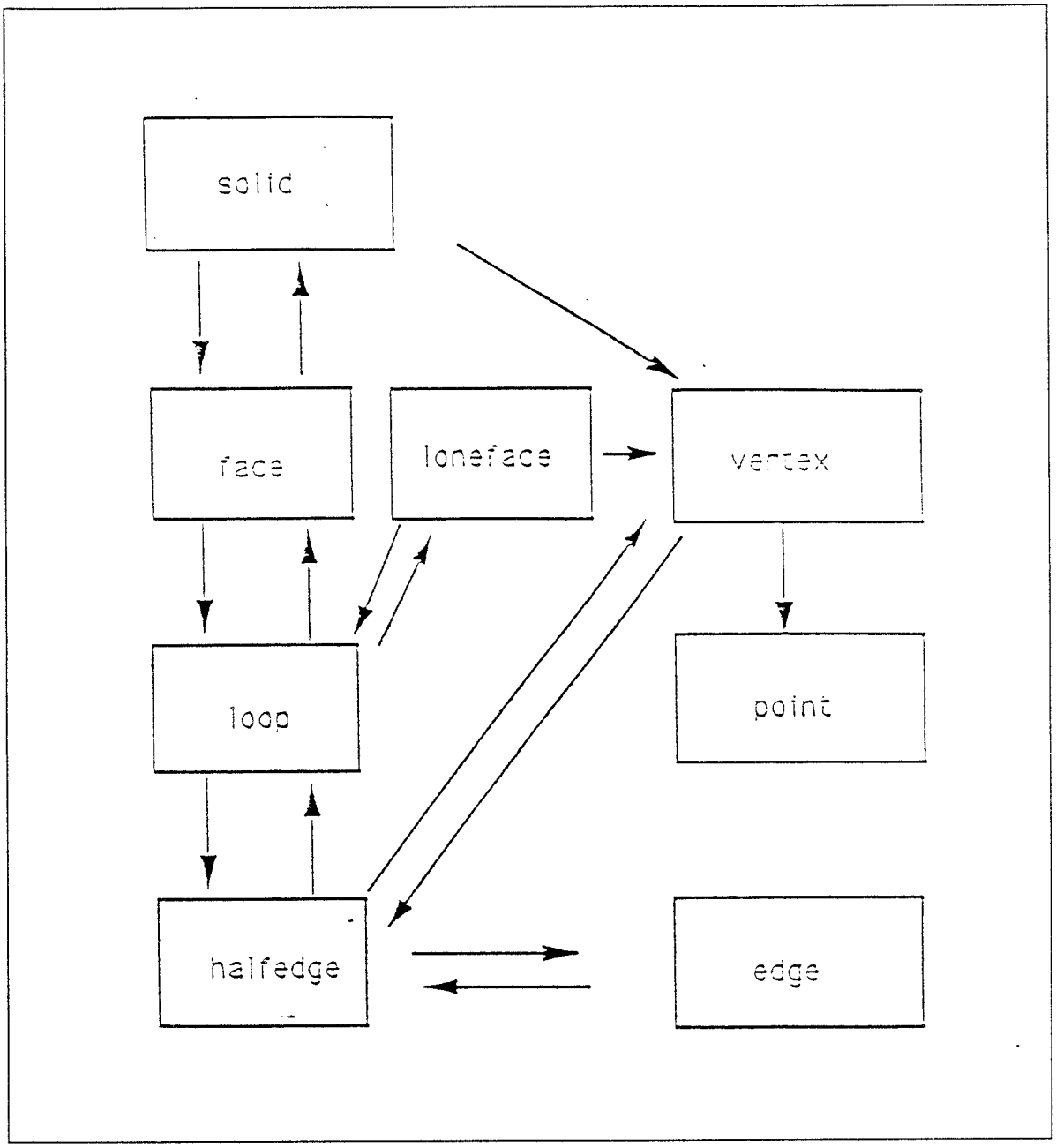


FIG. 3. - Halfedge and Loneface Hierarchy

fundamental data fields of the solid node are the list of faces that bound the solid, and the unordered list of vertices that are boundaries of the edges of the solid. The same vertex nodes are also accessible through the face hierarchy. Each `face` contains a complete list

```
struct vertex{                    /* vertex data structure          */
       short           vertexno; /* ID number for the vertex       */
       struct halfedge *vedge;   /* Halfedges emanating from vertex */
       struct point    *vdata;   /* Coordinate data                */
       struct vertex   *nextv;   /* The next vertex of the sverts  */
       struct vertex   *prevv;   /* The previous vertex            */
       };

struct point{                     /* point data structure           */
       float x, y, z, w;          /* x,y,z and w coordinates        */
       };
```

Table 1. - Vertex and Point Data Structures

of loops that bound it, pointers to the parent solid and unique `outer` loop, the next and previous faces in the list, and a `point` data structure defining the equation of the face plane. While the halfedges of the outer loop form a clockwise ring, all inner loops on the face are ordered counterclockwise. This orientation convention is used to indicate the `outside` of the model. The loop intermediary positioned between the halfedge and the face allows for the easy representation and manipulation of objects that have faces containing holes.

Table 1 gives the `C` definition of the vertex data structure. Each vertex node contains a pointer to its parent halfedge, pointers to the previous and next vertices in the list, an identification number, and a pointer to a point structure which holds the actual coordinate data stored in the form of four-dimensional homogeneous coordinates. The homogeneous coordinate system stores the `(x,y,z)` coordinates of a point in $\mathbb{R}^3$, plus a fourth coordinate `w`; it is a scaling factor, usually equal to one when the data represents an actual point. There are three important reasons for using homogeneous coordinates. First, they provide for more compact graphical display and manipulation of geometric data than is possible using a purely three-dimensional approach. This advantage stems from ability of homogeneous coordinates to represent all coordinate transformations as the matrix product of individual

four-by-four translation, rotation, and scaling matrices. Second, the homogeneous point structure may be used to store the equation of a plane. While the first three coordinates store a unit vector normal to the plane, the fourth coordinate stores the distance from the plane to the origin measured along the normal vector. Third, the signed distance of a point to the plane is simply given by the four-dimensional dot product between the point and the face equation. This feature is used extensively in our algorithm for extracting the cross-sections from beams, and ensuring that reinforcing bars remain within the beam solid.

Finally, the design specifications of our system required that users be provided with mechanisms to freely switch between three-dimensional operations on complete beam models, and two-dimensional operations on cross-sections. In principle, the halfedge data structure could be used for two-dimensional figures. Each figure would be modeled as a laminar solid of zero thickness, and with a front and a back face. The disadvantage of this approach is that all operations performed on the geometry of the cross-section would need to take account of these geometric features. In an effort to reduce the complication burden, a new `loneface` was added to the halfedge data hierarchy. It provides an alternative root node for accessing vertices, loops, and halfedges. The `loneface` is similar to the face node of the original data structure. However, because it stands alone without a parent solid, it has its own list of vertices to store the coordinates of the points of the face. It does not have an edge because all of the geometric information and adjacency relationships are available directly from the loops and halfedges. The remainder of the loop-halfedge-vertex hierarchy remains the same for a `loneface` as for a complete solid.

Key to the Naming of Euler Operators			
M	make	K	kill
J	join	S	split
V	vertex	E	edge
R	ring	H	hole
F	face	S	solid

Table 2 - Key to the Naming of Euler Operators

THE EULER OPERATORS

Boundary models are constructed using a set of powerful, low-level operators known as the Euler operators. They were originally introduced by Baumgart (1972) as a means of providing for a complete and general method of assembling geometrically complete polyhedral boundary models of arbitrary complexity. By convention, the Euler operators are referred to by mnemonic names based on their functionality. A list of operator components is given in Table 2. The 5 primary operators and their acronyms are: (a) Make Vertex Face Solid (MVFS), (b) Make Edge Vertex (MEV), (c) Make Edge Face (MEF), (d) Kill Edge Make Ring (KEMR), and (e) Kill Face Make Ring Hole (KFMRH). It is important to note that in Baumgart's naming convention, what has heretofore been referred to as a `loop` will be referred to as a `ring` in the naming of the Euler operators. Moreover, for each of `M` operator there exists a corresponding `K` operator to perform the inverse operation; for example, the KEV operator kills an edge, its corresponding halfedges, and a vertex.

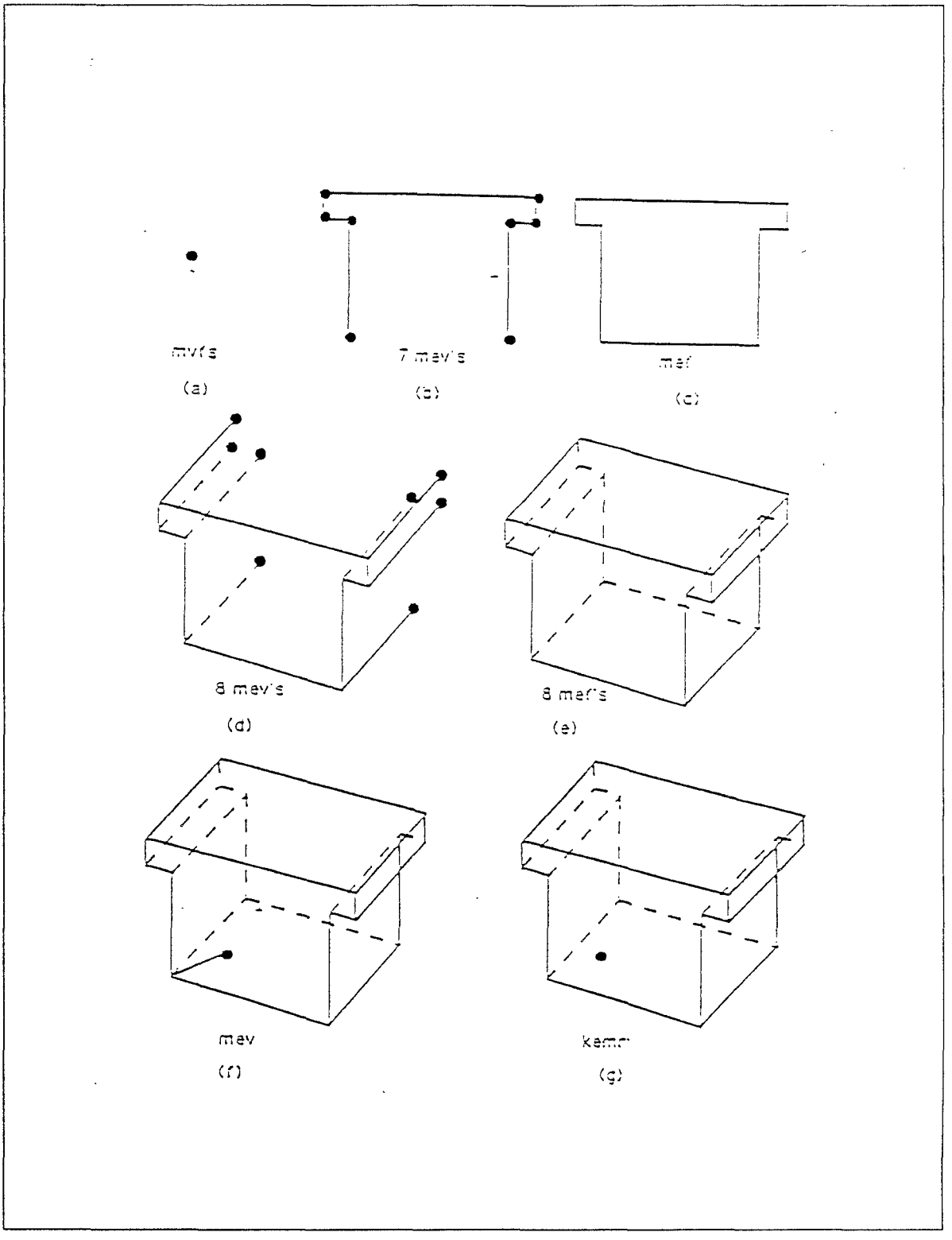


FIG. 4. - Generating a T-Beam using Euler Operators

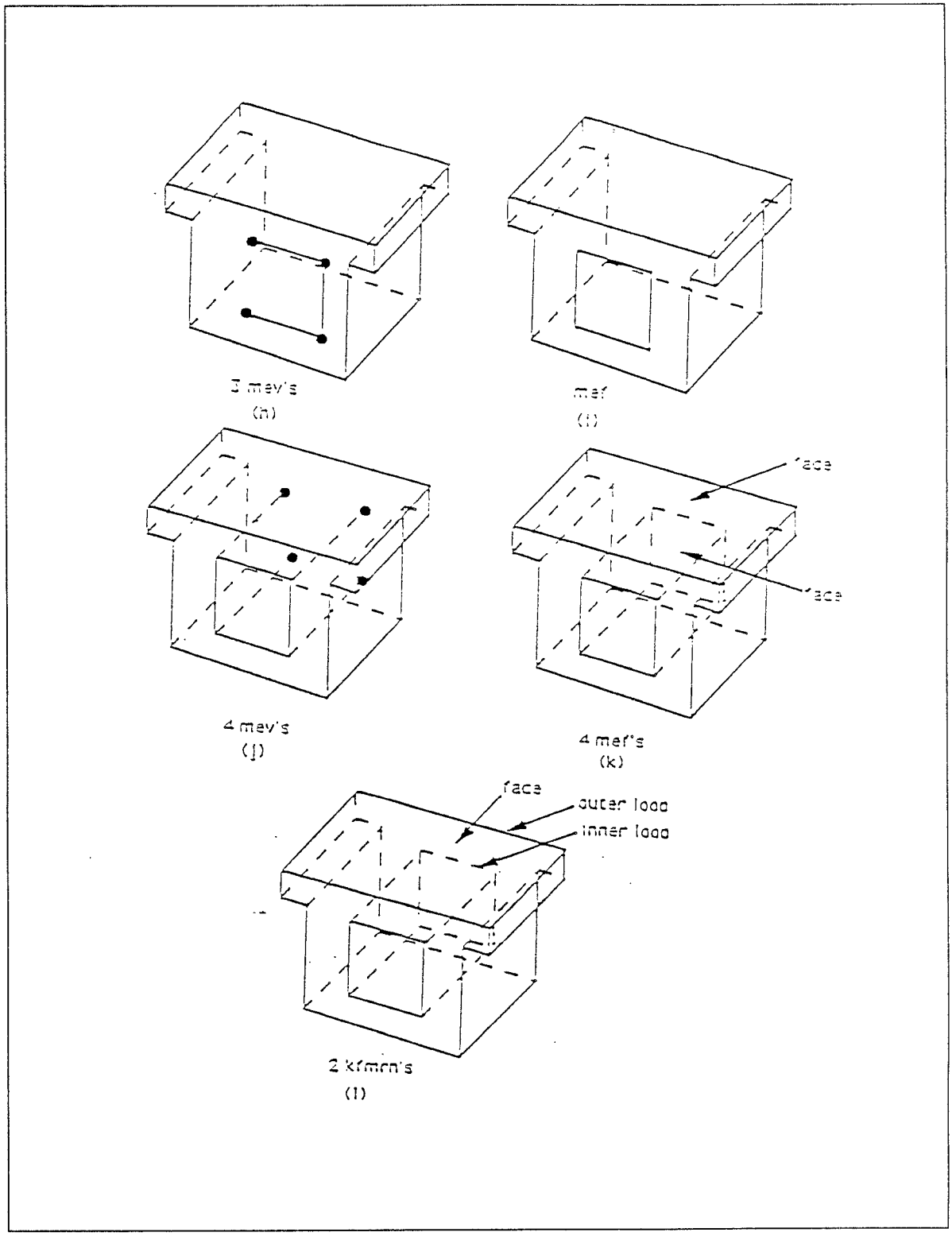


FIG. 4. - Generating a T-Beam using Euler Operators (cont.)

EXAMPLE : GENERATION OF A SIMPLE T-SHAPED BOX GIRDER

The functionality of the Euler Operators is demonstrated by working through the step-by-step process of generating the simple Tee-Shaped Box Girder shown in Figs. 4(a-l).

[a] The first step is to generate a primitive one-vertex solid with the MVFS operator, as shown in Fig. 4a. The result is a solid with only one face and one vertex. It has no volume, no edges, and no surface area, and as such serves as a primitive instance of the boundary model, with the hierarchy in place to be extended by other operators.

[b] The MEV operator adds an edge and a vertex to an existing model. Fig. 4b shows the result of seven successive MEV's to generate all of the vertices of the initial face and all but one of its bounding edges. At this point, the solid still has only one face with zero area.

[c] The MEF operator adds a new edge and a new face to the model. The first and last vertices of the face are connected with an MEF operator (see Fig. 4c). This turns the solid with one face into a laminar solid with two faces but no thickness.

[d] The model is extended into the third dimension by performing a MEV operation at each vertex of the new face. The result is shown in Fig. 4d.

[e] The model becomes a fully bounded solid after the application of eight MEF operators to form the side, top, and bottom faces; see the T-beam in Fig. 4e.

[f] A hole is cut through the beam by first generating a hole in one of the end faces. The end face hole is formed by extending an edge into the face with an MEV operation (see Fig. 4f).

[g] The KEMR operator eliminates temporary artifact edges that occur during the generation of the hole in the beam end face. The KEMR operation forms a primitive

one-vertex inner loop on the face (see Fig. 4g).

[h] The sides of the new loop are created with three MEV operations (see Fig. 4h).

[i] Next, the new loop is closed, and a new face is formed on the other side of it with an MEF operation (see Fig. 4i).

[j] The new loop is extended along the length of the beam with four MEV operations, in just the same way the original face was extended (see Fig. 4j).

[k] The new edges are closed off. New interior faces are created with four MEF operations, as shown in Fig. 4k.

[l] The KFMRH operator turns the "back" face of the hole into an inner loop of the back face of the beam, leaving the complete model of the T-shaped box-girder shown in Figure 4l.

It is important to note that the combined effect of steps (d) and (e) is to sweep out a plane figure along its normal axis to form a three-dimensional solid. This is the conceptual basis behind the sweeping operation used extensively in our system to generate solid models of beams from cross-sections.

DATA STRUCTURES FOR REINFORCING BARS

The implementation and modeling of three-dimensional reinforcing bar trajectories proved to be more difficult than initially expected. While it is desirable to create a prototype system that provides for the specification of almost arbitrarily shaped reinforcing bar trajectories, in practice, the complexity of required editing operations rapidly increases with system flexibility. Since the purpose of implementing the prototype system was to demonstrate the feasibility of a design methodology, a compromise solution was adopted. The compromise restricts each individual bar to lie in either a vertical or horizontal plane

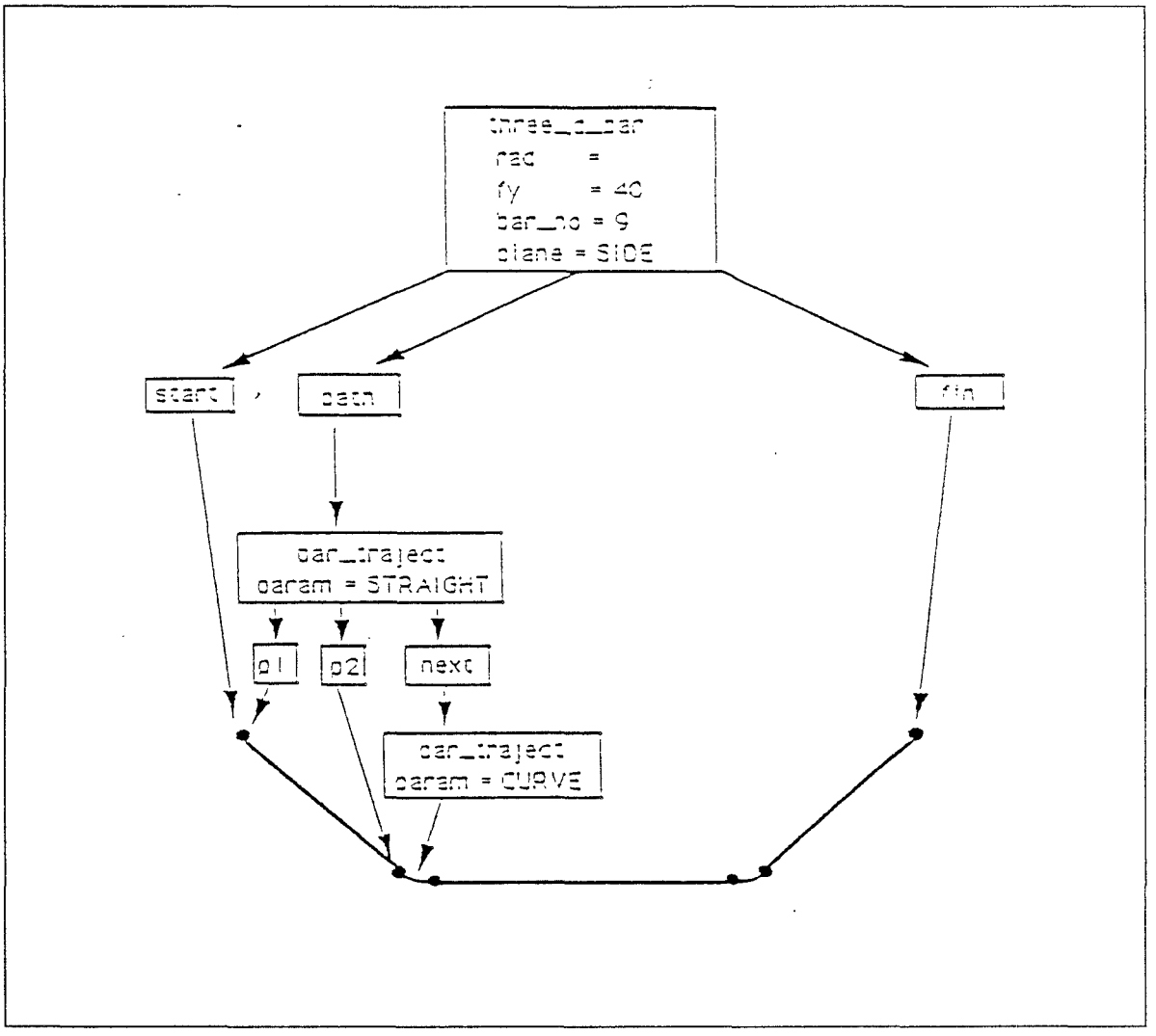


FIG. 5. - The 3-D-Bar Data Structure

parallel to the longitudinal axis of the beam. This allowed us to provide relatively simple point-and-click editing operations for the generation of straight and bent bar segments.

Reinforcing bar trajectories are described with the two-level data structure hierarchy shown in Fig. 5. The top level is the `three_d_bar` data structure containing parametric information about the bar, such as its size, pointers to the starting and finishing coordinate locations, and a flag indicating whether the bar lies in a vertical or horizontal plane. It

```
struct cross_section {
    struct loneface  *geom;          /* geometry of oriented section            */
    struct two_d_bar *bar_list;      /* bars in the cross-section               */
    float  top_block, bot_block;     /* location of top,bottom stress blocks    */
    float             cx, cy;        /* location of section centroid            */
    struct track      **tracks;      /* data structure to modify cross-section  */
    int               num_tracks;    /* number of tracks                        */
    struct cross_section *next_sect; /* next cross-section in the list          */
};

struct two_d_bar{
     int           num;              /* bar number               */
     float         rad;              /* radius of the bar        */
     struct point  cent;             /* location of the center   */
     float         fy;               /* yield strength of the bar */
     float         sig;              /* theoretical stress       */
     float         developed;        /* development length factor */
     struct two_d_bar *nextb;        /* next bar in the list     */
};
```

Table 3. - Cross Section and Two-Dimensional Data Structures

also contains a pointer to the second level of the bar trajectory data structure, which is a linked list of simple straight or curved bar segments defining the complete path of the bar inside the beam. Each element in the trajectory list contains the starting and ending points of the trajectory and a flag indicating whether the trajectory is a curved or straight segment.

CROSS-SECTION DATA STRUCTURE

Solid modeling approaches to engineering analysis begin with the extraction of appropriate details for each design task from a more general model, and then create a temporary model for analysis purposes. For the prototype implementation of the RC modeler, a fast algorithm was needed for the user, or program itself, to slice the beam and extract cross-sections at any point along the longitudinal axis of the beam.

Table 3 summarizes the C language definitions for the cross-section and two-dimensional reinforcing bar data structures. The two main fields in the cross section data structure are: (a) a pointer to a `loneface` data structure holding the geometric form of the cross-section, and (b) a pointer to a linked list of two-dimensional reinforcing bars. Data fields (location of the top and bottom stress blocks, centroid of the section, and the nominal strength of the concrete) are also provided for the real-time calculation and storage of ultimate section capacity, graphical display, and generation of neighboring cross-sections along the length of the beam.

CROSS-SECTIONING ALGORITHM

An algorithm formulated by Margalit (1988) is used as a basis for slicing the beam segment and extracting a cross section for the first time. The step-by-step procedure is:

[1] Initialize a `point` data structure which contains the equation of a cross-sectional cutting plane at the desired location.

[2] For each face of the solid model, dynamically allocate memory for a temporary working array of vertices. Compute the intersection of all the halfedges in the solid model with the cutting plane; this is an easy calculation because the signed distance of a point to the cutting plane is simply given by the four-dimensional dot-product between the vector defining each point, and the vector defining the cutting plane. For each halfedge-cutting plane intersection, create a new vertex and place it in the working array of vertices.

[3] When a new vertex is created at step [2], also create links to the two endpoints of the edge upon which it lies. These links are stored in a list of special data structures

called `tracks`, and subsequently form a line upon which the vertex can `ride` in the extraction of further cross sections from the beam segment. Details are given below.

[4] Determine the direction of the intersection line between each face and the cutting plane. This is given by the cross-product of the normal vector of each face with the normal vector of the cutting plane. Sort the face/cutting plane intersection vertices according to the direction criteria. All of the bounding edges of the cross-section are now implicit within the arrays of vertices.

[5] Starting with the temporary working vertex array for the first face, generate a primitive `loneface` with one loop at the starting vertex. Add an edge linking this vertex to the second vertex in the array, and then remove these two vertices from the array. The starting vertex is saved so that the loop can be closed later.

[6] Move to the vertex array corresponding to the neighboring face, and find the implicit edge which begins at the ending location of the last edge created. Add a new edge from this vertex to the next vertex in the new array, and remove these two vertices from the working vertex array. New edges are added to the loop of cross section until the next vertex is identical to the original starting vertex of the loop, whence the loop is closed. The process is repeated to generate another loop until there are no more implicit edges remaining.

[7] Sort through the loops of the new cross-section to find the outer loop. If the cross section contains only one loop, then the problem is already solved. Otherwise, the algorithm identifies the loop contain vertices outside all of the remaining loops. The orientation of the loops are verified so that the outer loop is oriented clockwise and the inner loop is oriented counter-clockwise.

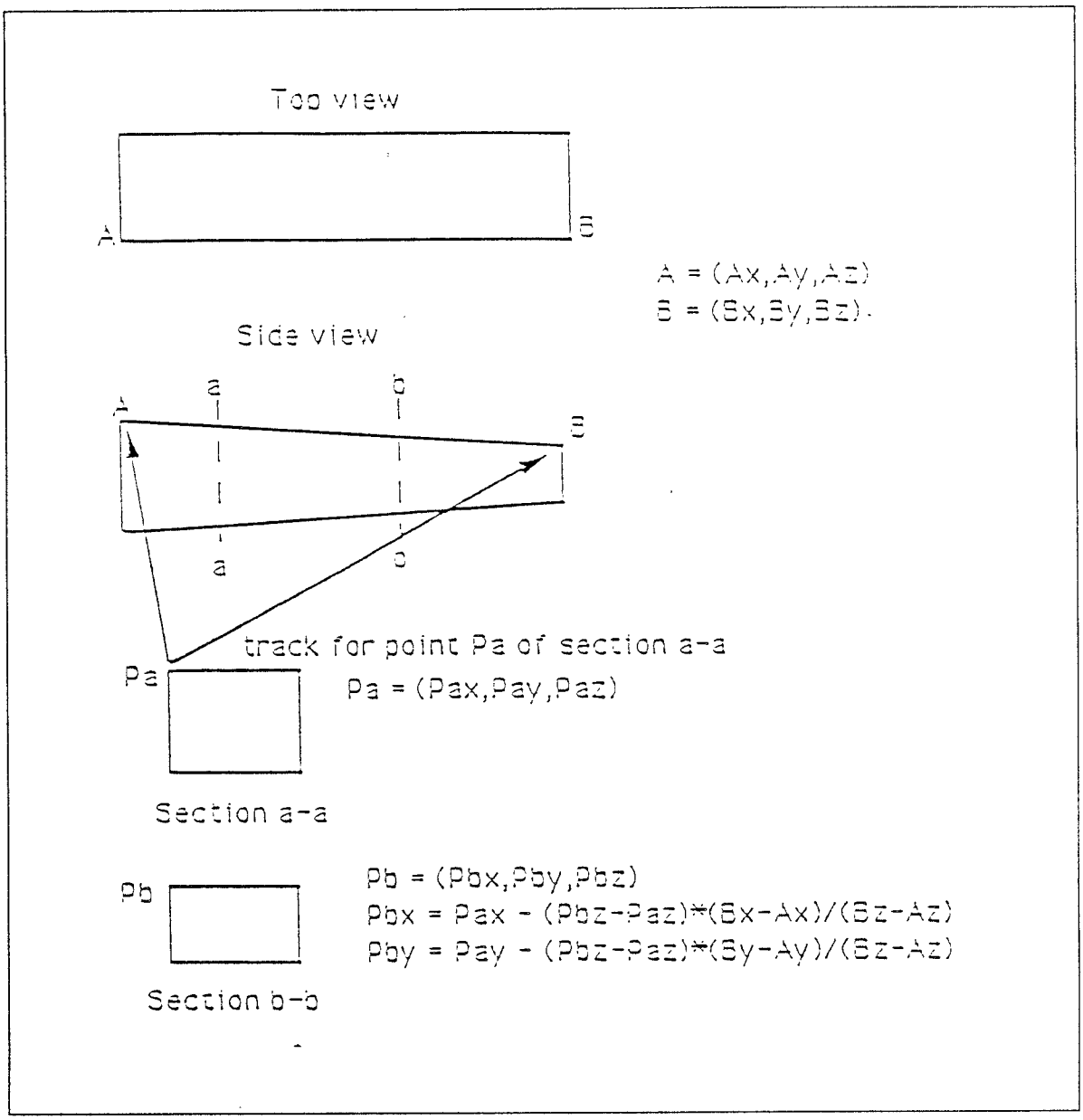


FIG. 6. Modifying an Existing Cross Section Using Tracks

[8] Compute the intersection of the three-dimensional reinforcing bars and the cutting plane. Store each bar/plane intersection as a two-dimensional reinforcing bar in the cross-section data structure.

The result of this algorithm is a `loneface` data structure containing the geometry of the

cross-section and a list of two-dimensional reinforcing bars. When further cross section shapes are needed from the same beam segment, the sectioning algorithm is modified to take advantage of the fact that the required and initial cross sections will have the same topology. Indeed, the (x,y,z) vertex coordinates of new cross sections may be very quickly computed via linear interpolation along the `track` of the longitudinal edges computed during Step 3. Once the geometry of the new section is known, Step 8 of the sectioning algorithm is re-executed to compute the location of two-dimensional reinforcing bars.

Geometric Properties of Cross Sections. - Geometric properties of the cross sections, such as centroid location, moments of inertia, and section area, are computed via Green's Theorem; for details, see Wylie and Barrett (1982). For example, if R is a closed region with external perimeter C made up of straight line segments, then the moment of inertia I_{xx} about the x axis is given by

$$I_{xx} = \int_C xy^2 dy \;=\; \left[\frac{1}{12}\right] \cdot \sum_{i=1}^{N} \left[x_i - x_{(i-1)}\right] \cdot \left[y_{(i-1)} + y_i\right] \cdot \left[{y_{(i-1)}}^2 + {y_i}^2\right] \qquad (1)$$

where (x_i, y_i) is the coordinate of the i^{th} vertex, and N is the total number of straight line segments on perimeter C. When cross sections contain holes, the appropriate geometric property is the sum of component properties from the outer and inner loops, with the direction of each loop serving to maintain the correct sign for the contribution of each loop. For a complete list of functions to compute geometric properties, the interested reader is referred to Pinto De Magalheas (1979).

ULTIMATE FLEXURAL STRENGTH

The prototype system has an algorithm for computing ultimate flexural strength of beam cross sections where the layout of reinforcement, and the section shape, are both symmetric about the vertical axis. External forces are restricted to uniaxial bending alone. Together these assumptions allow the effects of torsion and axial forces to be neglected, and imply that in the analysis, the neutral axis will be horizontally aligned.

Our desire to work with general section shapes meant that it was not possible to derive closed form analytical formulae for the ultimate section strength. Instead, the analysis uses an iterative strategy based on assumed neutral axis position, and the equations of equilibrium to obtain a sequence of estimates for ultimate section strength. During the formulation stages of the algorithm, it became evident that an efficient implementation would require a fast way of computing: (a) the area of the stress block given a neutral axis location, and (b) the neutral axis location given a required stress-block area. The adopted solution is to start each section analysis by constructing a cumulative distribution function (CDF) of cross section area above a given horizontal line position.

The algorithm for computing the CDF makes extensive use of the polygon intersection algorithm developed by Margalit (1988). Figs. 7(a-d) show, for example, the initial stages of computing the CDF for a tapered section with one hole in it. First, the section is divided into horizontal segments separated by vertex discontinuities, as shown in Fig. 7a. A new polygon is generated to intersect the cross-section at the desired location (see Fig. 7b). Two lists of edges are generated, one for the set of segments making up the cross section, and a second for the temporary intersection polygon. All of the cross section edges are then classified as being `inside`, `outside`, or `on` the boundary of the intersecting polygon

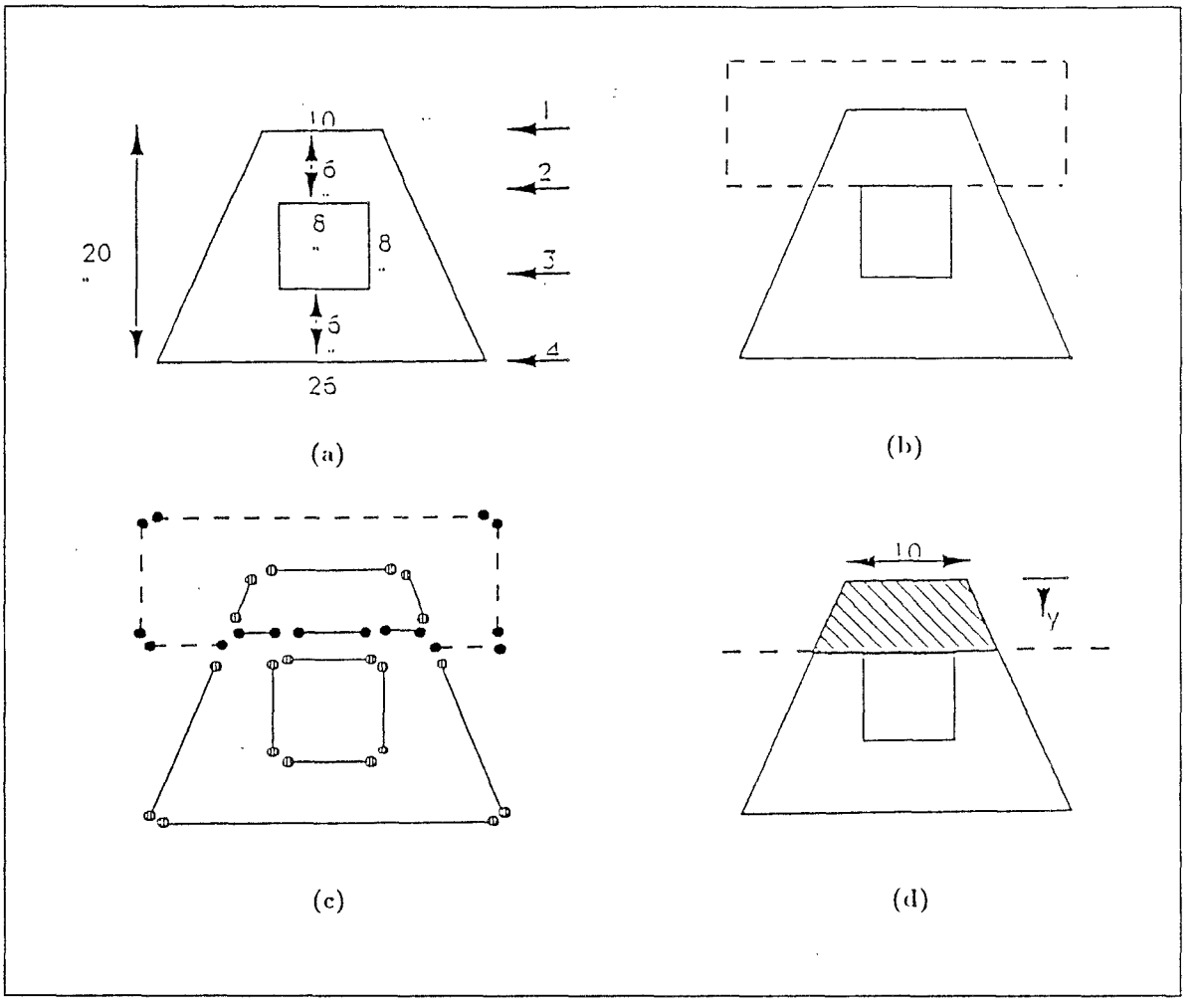


FIG. 7. Generating CDF of Section Area

(see Fig. 7c). After duplicate edges on the boundary are eliminated, a single loneface is assembled using the Euler operators, as shown in Fig. 7d.

Given that the top edge of the example polygon function is 10 units long, the bottom intersecting edge length is 14.8 units long, and the area of the intersecting polygon is 74.4, it follows that the CDF for the first polygon segment is

$$\text{CDF Area}(y) \;=\; \frac{y}{2} \cdot [20 + 0.8y] \tag{2}$$

where y is the vertical distance of the horizontal axis from the top edge. Continuing the

process for the remaining two sections gives the piecewise continuous quadratic function

$$\text{CDF Area}(y) = \begin{cases} \frac{1}{2} \cdot y \cdot [20 + 0.8y], & \text{for } 0.0 \leq \text{y} \leq 6.0; \\ 74.4 + \frac{1}{2} \cdot [y - 6] \cdot [13.6 + 0.8(y - 6)], & \text{for } 6.0 \leq \text{y} \leq 14.0; \\ 154.4 + \frac{1}{2} \cdot [y - 14] \cdot [42.4 + 0.8(y - 14)], & \text{for } 14.0 \leq \text{y} \leq 20.0. \end{cases} \quad (3)$$

Once the CDF function is computed, the flexural strength calculation proceeds in the usual way - see Preston (1991) for step-by-step details - and makes use of the standard assumptions; plane sections remain plane, negligible tensile strength of the concrete, and stress-strain curve for steel is perfectly elasto-plastic. The stress distribution within the compressive zone of the concrete at ultimate loading is modeled with equivalent rectangular stress distribution, as described by ACI 318-83 (1983).

CONCLUSIONS AND FUTURE WORK

This paper has focussed on the formulation of data structures and algorithms for the solid modeling of RC beam structures composed of three dimensional concrete solids, and steel reinforcing bar trajectories. Future versions of the prototype system will be capable of analysing cross sections subject to axial loads and biaxial bending. The long term research goal is to work out the details of combining generally shaped RC components into frame and frame-wall assemblies.

ACKNOWLEDGEMENTS

This work was supported in part by the Minta Martin Foundation, NSF's Engineering Research Centers Program : NSFD CDR 8803012, and NSF Research Initiation Grant NSF BCS 8907722. This support is gratefully acknowledged. The views expressed in this paper are those of the authors, and not necessarily those of the sponsors.

APPENDIX I. - REFERENCES

[1] ACI 318-83 : Building Code Requirements for Reinforced Concrete (1983), American Concrete Institute, Detroit, MI.

[2] Baumgart B.G., (1972), "Winged-edge Polyhedron Representation," STAN-CS-320. Department of Computer Science, Stanford University, Palo Alto, CA.

[3] Mantyla M. (1988), **An Introduction to Solid Modeling**, Computer Science Press, Rockville, Maryland.

[4] Margalit A. (1988), " An Algorithm for Computing the Union, Intersection, and Difference of Two Polygons," *Center for Automation Research*, Technical Report 1995, University of Maryland, College Park, MD.

[5] Martini K., and Powell G.H. (1990), "Geometric Modeling Requirements for Structural Design," *Engineering with Computers*, Vol. 6, No. 2, pp. 95-102.

[6] Pinto de Magalhaes M. (1979), "Biaxially Loaded Concrete Sections," *Journal of the Structural Division*, ASCE, Vol. 105, No. ST12.

[7] Preston J. (1991), An Application of Solid Modeling Concepts to Design Software for Reinforced Concrete Beams, *Masters Thesis*, Department of Civil Engineering, University of Maryland, College Park, MD 20742.

[8] Preston J., and Austin M.A. (1992), "Solid Modeling Based Design of RC Beams," *Journal of Computing in Civil Engineering*, This issue.

[9] Requicha A.A.G. (1990), "Representations for Rigid Solids: Theory, Methods, ans Systems," *Computing Surveys*, pp. 437-464, Vol. 12, No. 4.

[10] Subrahmanian E., Podnar G., Elm B., and Westerberg A. (1989), "Towards a Shared Computational Support Environment for Engineering Design," *Technical Report EDRC 05-41-89*, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

[11] Weiler K. (1986), "Topological Structures for Geometrical Modeling," Rensselaer Polytechnical Institute, Troy, NY.

[12] Westerberg A., Grossman I., Talukdar S., Prinz F., Fenves S., and Maher M.L. (1989), "Applications of AI in Design Research at Carnegie Mellon University's EDRC," *Technical Report EDRC 05-30-89*, Carnegie Mellon University, Pittsburg, PA.

[13] Wylie C.R. and Barrett L.C. (1982), **Advanced Engineering Mathematics**, McGraw-Hill Book Company, pp. 1103.

[14] Zamanian M.K., Fenves S.J., Thewalt C.R., and Finger S. (1991), "A Feature-Based Approach to Structural Design," *Engineering with Computers*, Vol. 7, No. 1, pp. 1-9.

To appear in ASCE
Journal of Computing in Civil Engineering

SOLID MODELING OF RC BEAMS
PART 2 : COMPUTATIONAL ENVIRONMENT

By J.L. Preston[1], and M.A. Austin, A.M. ASCE[2]

ABSTRACT

This paper is the second of a two-part series describing the design and implementation of an interactive reinforced concrete beam design environment called BeamTool. Part two focuses on issues of implementing and testing BeamTool. This includes design of the user interface to work on two- and three-dimensional descriptions of the design problem, techniques for specifying the geometry of the three dimensional concrete solid and the reinforcing bar trajectories. Features of the BeamTool system are illustrated via the design of a three-span reinforced single-cell box girder beam.

Keywords : Solid Modeling, Computer-Aided Design, Reinforced Concrete.

[1] Graduate Research Assistant, Department of Civil Engineering and Systems Research Center, University of Maryland, College Park, MD 20742, USA.

[2] Assistant Professor, Department of Civil Engineering and Systems Research Center, University of Maryland, College Park, MD 20742, USA.

INTRODUCTION

This paper is the second in a two-part series describing the formulation and development of a prototype system for the interactive specification, analysis, and design of reinforced concrete (RC) beam structures. In the companion paper (Austin 1992), it was proposed that the prototype system should satisfy the following set of criteria:

[1] To provide for the interactive generation of solid models of the beams, with arbitrarily shaped cross-sections, possibly containing holes. This was to be implemented by means of interactive drawing and sweeping algorithms.

[2] Create an editing environment that allows for the placement of reinforcing bars both within the beam at a three-dimensional level, and within cross-sections at a two-dimensional level. Practical design considerations dictate that reinforcing bars be contained completely inside the concrete solid. The checking of these geometric constraints must be enforceable at both the two- and three-dimensional levels.

[3] Develop interactive software tools that allow the user to freely move between a two-dimensional design and analysis environment where cross-sections are the highest level of objects, and a three-dimensional environment where work is done on complete beams.

[4] Provide analytical tools to determine the ultimate flexural strength of the beam along its length.

This paper describes the design and functionality of the concrete beam design software system `BeamTool`. BeamTool is an exploratory attempt at implementing the data structures and algorithms described in Austin (1992). The main goal of this phase of the project was a small prototype system, the development of which would improve our understanding of the

likely problems faced in implementing the solid modeling approach to large-scale software systems.

DESIGN OF REINFORCED CONCRETE BEAMS

The design and analysis of RC beams is so complex that, in general, it is impossible for engineers to consider all aspects of a problem at once. A common strategy for making the design process tractable is to decompose the overall design problem into sub-problems, and focus on the solution of sub-problems, each of which considers the remaining aspects of the design to be at a more abstract level of detail.

Kalay (1989) points out that the levels of abstraction associated with a particular phase of a design process can often be classified by dimensionality. In the design of RC beams, for example, one typically moves back and forth between tentative three-dimensional beam geometries, and the design (layout of rebar and analyses of engineering performance) of two-dimensional cross sections. This strategy of freely switching between two- and three-dimensional levels of abstraction is motivated in part by the `principle of least dimensionality`; that is, humans find it is easier to check containment issues on two-dimensional representations of an object than in a three-dimensional model. The same principle applies to the complexity of operands needed to evaluate containment issues in an automated system.

BEAMTOOL SOFTWARE SYSTEM

BeamTool is a software system for the interactive design of RC beam structures using abovementioned style of design. Its main components are a graphical user interface, and software tools for RC design and geometric modeling. The software was developed for use

on Sun Microsystems SUN/3 and SUN/4 workstations under the UNIX operating system. The C programming language was used for this implementation because of the ease with which powerful and complex data structures may be defined, and because it allowed for the use of the SunView libraries (Sun 1988) in the development of the point-and-click graphical user interface.

The prototype implementation reflects a balance in competing design criteria. First, there is the matter of having enough screen space for the edit mode buttons, and to graphically display the design. During the early stages of interface development it became evident that three separate screens would be needed, one for each stage of the design process. Their purposes are: (a) Definition and assembly of the solid boundary model of the concrete beam, (b) Placement and sizing of reinforcing bars within the three-dimensional beam model, and (c) Determination of the flexural capacity envelope. Designers are expected to progress from screens (a)-(c) in the natural course of setting up and solving a design problem.

It is conceivable, however, that a designer might want to edit the three-dimensional geometry of the beam structure after the reinforcing bar trajectories have been specified. This scenario pits the need for sophisticated, yet easy-to-use, editing operations directly against the complexity of maintaining geometric coherency in a design. It also raises difficult issues as to how the BeamTool system should behave. For example, when three dimensional reinforcing bar trajectories are being specified, practical design considerations dictate that they be constrained to lie inside an already defined concrete solid. It does not automatically follow that the editing of concrete solids should be constrained to remain outside all previously defined reinforcing bars? Indeed, major modifications to the three-dimensional geometry (and reinforcing layout) are likely to be most efficiently executed if

violations in design rules and geometric integrity are temporarily permitted.

The first cut of the BeamTool software does not provide for the relaxation of geometric containment rules during the editing process. In order to limit the need for such a relaxation of constraints, the solid model cannot be edited after is has been assembled.

DESIGN EXAMPLE

In the following sections, features of the BeamTool system are illustrated via the design of a three-span reinforced single-cell box girder beam. The design objective is to determine a beam geometry, and layout of steel reinforcing, which has a diagram of ultimate flexural capacity that just covers the (factored) design moment envelope.

Design Specifications. - Fig. 1 shows the plan and elevation views of the example footbridge. The bridge is twelve feet wide, and is made up of three equal spans of 40 feet each. The bridge will be continuous across the interior supports, hinged at the two ends, and symmetric in the longitudinal direction. The design load will be a factored uniform dead load of 3.36 kips/ft. and a factored uniform live load of 5.1 kips/ft. Fig. 2. shows the design moment envelope resulting from these loads. The maximum positive moment is 1252 kip-ft. and occurs 17 feet from the end of the bridge. The maximum negative moment is -1490 kip-ft. at the interior support. Within the center span, the maximum positive and negative moments are 560 kip-ft. and -268 kip-ft., respectively. Since the magnitude of the maximum positive moment near the center of the end spans is less than the maximum negative moment at the interior supports, it was decided to design a haunched beam with a greater depth at the supports than in the spans.

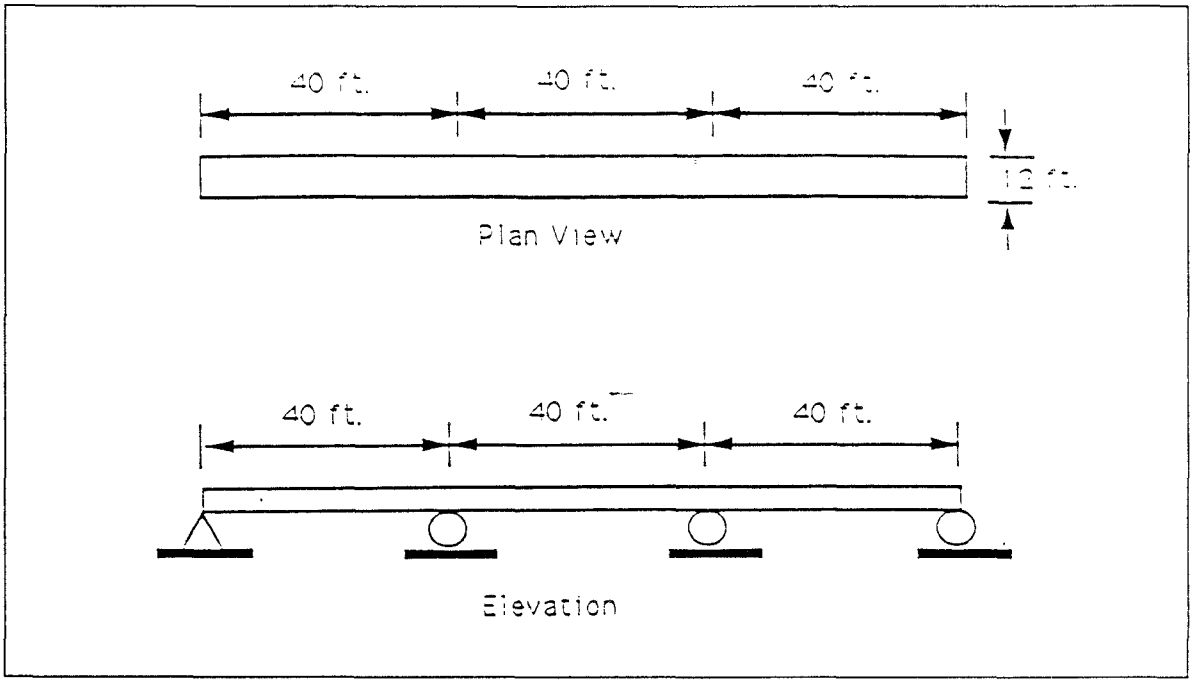


FIG. 1. Geometry of FootBridge

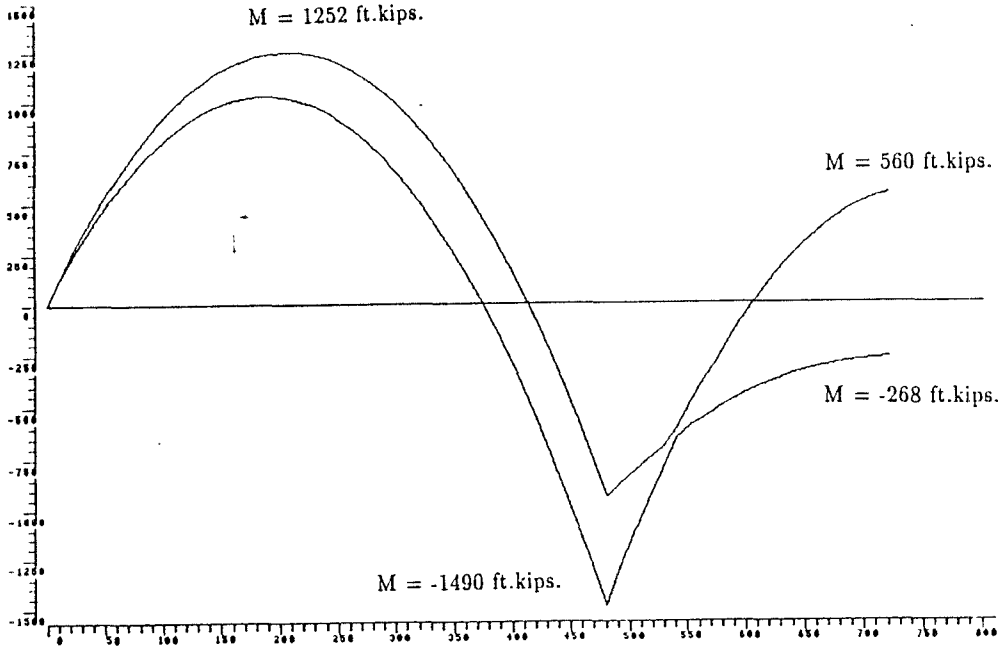


FIG. 2. The Design Moment Envelope

SOLID BUILDING SCREEN

The Solid Building Screen supports the operations a designer must work through to proportion design cross-sections, and create a new three-dimensional beam solid. It appears when the BeamTool software is first executed. The main components, as labeled in Fig. 3, are: (a) Main Control Panel, (b) Window with Side View of Beam for Placing Cross Sections, (c) Upper Control Panel for Creating Solid Model of Beam, (d) Window for Drawing and Editing Cross-Sections, (e) Lower Control Panel for Cross-Section Operations, and (f) Information Window. The Main Control Panel is common to all three screens; it contains buttons for global operations, such as setting the scale of the beam and choosing which of the three design screens to work on.

Setting Scales for Design Problem Description. - The first step in the design process is to specify three scales that will cover the approximate width, height and length of the beam to be designed. For the current design, the width and height scales are set to 200 inches. The scale in the longitudinal direction is set to 800 inches (66.7 ft); this reflects the need to model only one half of a symmetric bridge design. After the design scales are set, axes are automatically drawn on the screen in the side view and cross-sectional view windows, as shown in Fig. 3.

BeamTool provides mouse-driven commands and grid snap facilities for the precise specification of cross section shapes and reinforcing bar trajectories. The design grid is a three-dimensional block of points whose granularity in each direction is governed by the finest increment of distance a designer expects to work with. In the case of our footbridge design example, grid spacings of 3 inches are specified in the horizontal and vertical directions. A grid spacing of 20 inches is used along the length of the beam. Once the design

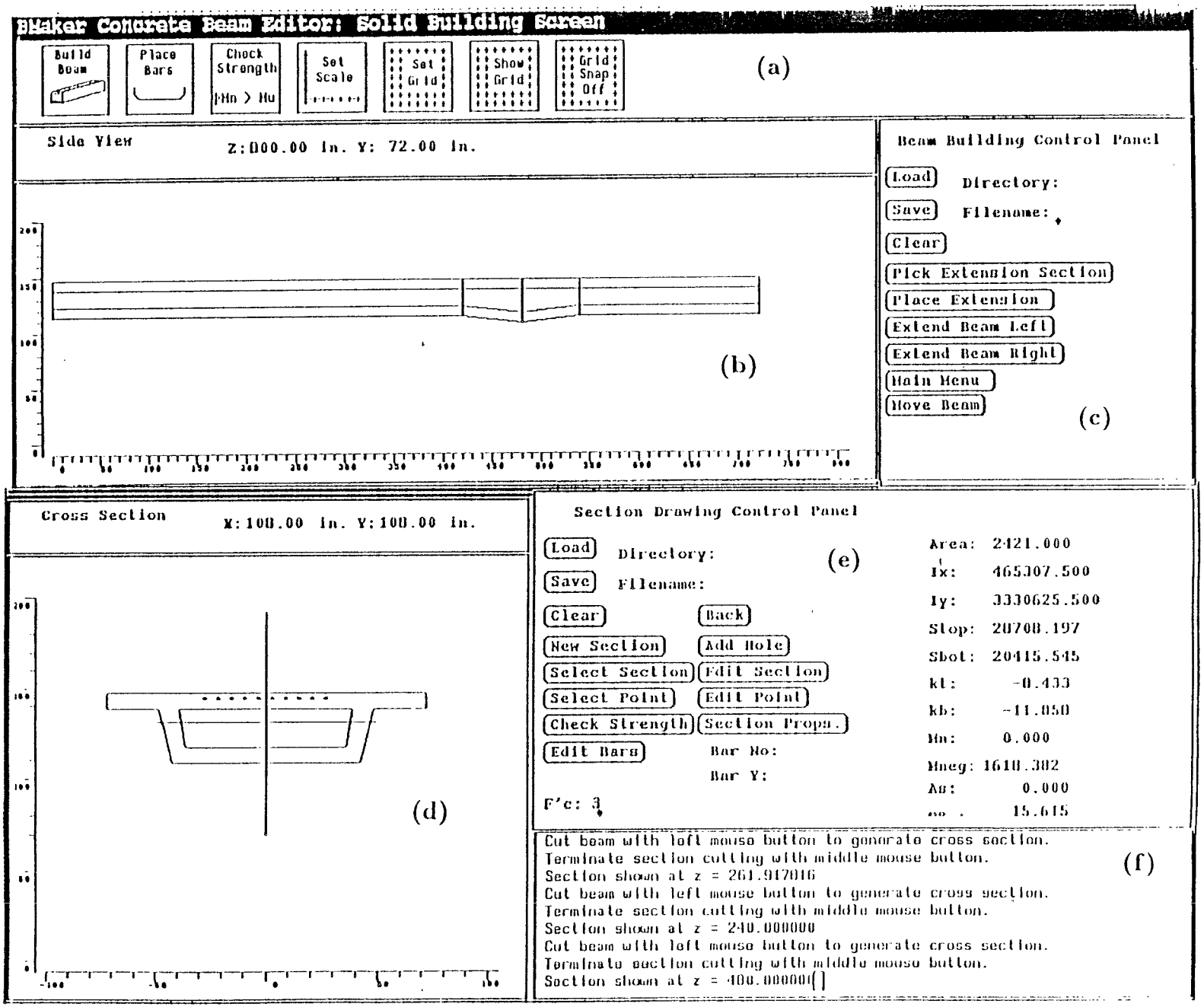


FIG. 3. - The Solid Building Screen

grid is assembled, grid snap procedures permit lines to be drawn only from one grid point to another. Design grids are displayed and hidden by activating the `Show Grid` and `Hide Grid` buttons, respectively (see the Main Control Panel in Fig. 3).

INTERACTIVE DESIGN OF CROSS SECTIONS

BeamTool provides designers with the computational tools to interactively layout cross section shapes, position two-dimensional reinforcing bars, and compute the ultimate flexural strength of tentative cross section designs.

Specification of Cross Section Shape. - Cross section shapes are created by drawing with the mouse in the cross-section window. At each step of the procedure, the position of the mouse is transformed into the user-defined coordinate scale, and mapped to functions for generating the section model. Button events are used to enter and leave cross section drawing modes, and snap perimeter lines to the closest design grid point. They also trigger callbacks to functions for the diagnostic checking of cross section geometry, and incremental assembly of the geometric model with Euler operators. For example, the first time the left mouse button is clicked in the section-drawing mode, a new primitive `loneface` structure is generated with an adapted version of the Euler operator MVFS (make vertex-face-solid). This operator creates a new `loneface` with one loop, and one vertex. Further mouse clicks are given to indicate the location of remaining vertices in the outer loop of the section. As successive edges are added to the outer loop, the `loneface` is extended by means of MEV (make edge-vertex) operators to form the bounding edges of the cross-section. The last vertex in the cross section perimeter is specified with a single click of the middle mouse button. The `loneface` is closed with a modified Euler operator similar to the MEF (make

edge-face) operator.

Geometric Validity Of Cross-Sections. - A proposed cross section edge is not added to the geometric model unless it passes a series of diagnostic checks. The checks ensure that: (a) each polygon is non-self-intersecting, and (b) all holes lie completely within the outer boundary, and are completely outside any other inner loops. If any intersections occur, then the proposed edge is rejected. This strategy ensures that the area bounded by the cross-section will be one well-defined contiguous region, and thus, may be swept out along the z-axis to form a geometrically valid solid model.

Sizing and Placement of Reinforcing Bars in Cross Section. - Computational tools are provided for the definition, placement and editing of individual reinforcing bars, and groups of reinforcing bars. The section drawing control panel in Fig. 4 shows, for instance, that editing modes are available to `Set` and `Move` horizontal bar lines, which fix the y-coordinate in the cross-section that all the bars must lie. The bar line is set by selecting the `Set Bar Line` button in the lower control panel, moving the mouse into the cross-sectional view window, and pressing the left mouse button with the cursor at the desired y-coordinate. Bars are placed in the section by selecting the `New Bar` button (see the Bar Placement Control Panel of Fig. 6), and then placing individual bars by pressing the left mouse button with the cursor at the desired x-coordinate. Further editing modes are available to `Add` and `Select` individual bars, and to `Add`, `Move`, `Copy` and `Delete` groups of reinforcing bars. In all cases, geometric checks are performed to prohibit placement of bars outside the section, or on top of other bars.

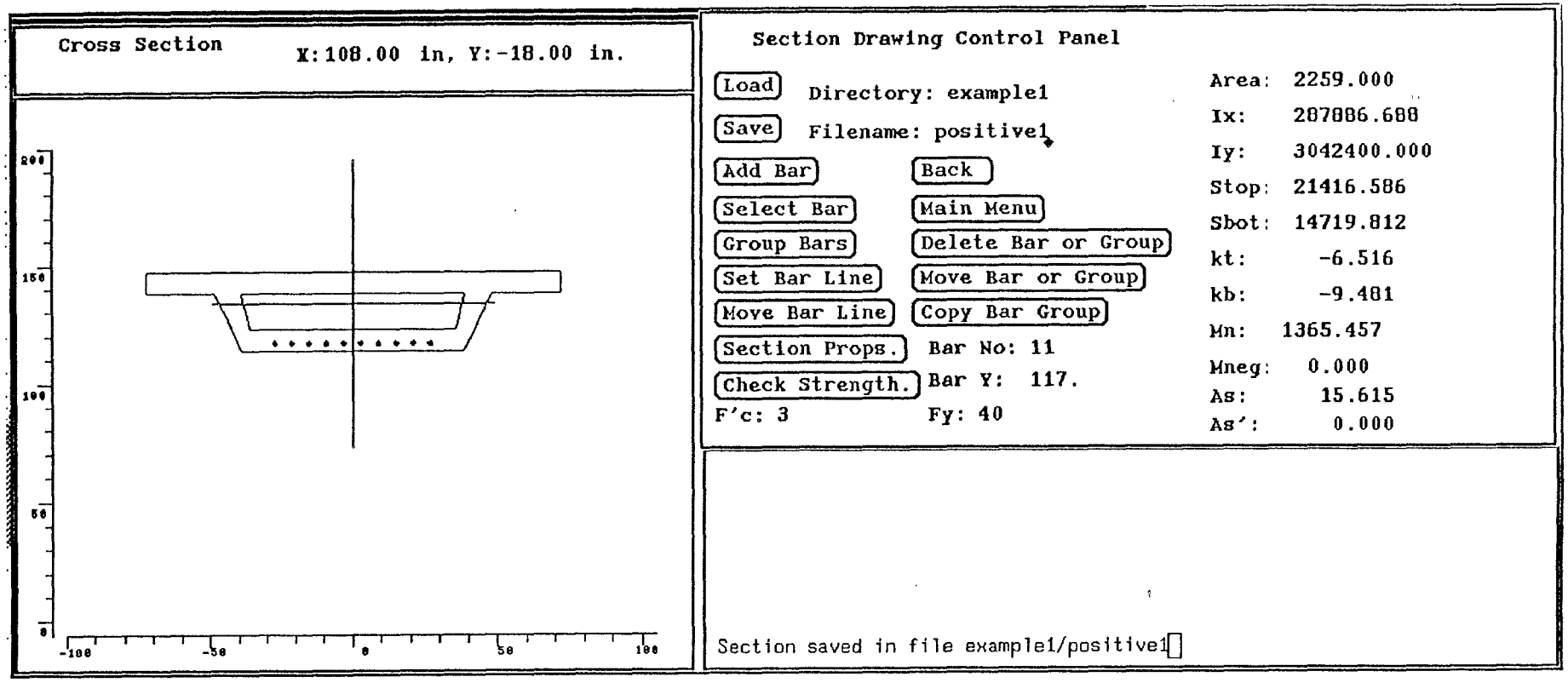


FIG. 4. Cross Section Design

Computation of Ultimate Flexural Strength. - Selecting the `Check Strength` button on the lower control panel triggers the execution of a series of analysis functions to compute the flexural strength of the cross section. First, all of the gross section properties are calculated and displayed in the appropriate data fields on the lower control panel. The position and orientation of the centroidal axes for the cross-section are computed and and drawn in the cross-sectional view window. Next, the areas of the bottom and top reinforcing bars are determined and these values are displayed in the data fields labeled A_s and $A_s^{'}$ respectively. The ultimate flexural strength is calculated according to ACI 318-83, and displayed on the screen. Together these features allow designers to determine the shape and rebar quantities of cross sections at critical locations in the beam structure, before moving on to consider the three dimensional design as a single entity.

DESIGN OF FOOTBRIDGE CROSS SECTIONS

The footbridge design begins with the proportioning of a box-girder cross section for the central portion of the end span; this is where the maximum positive design moment is 1252 kip-ft. The first step is to use the aforementioned sequence of button and mouse-driven commands to layout the outside perimeter of the box section. The `Add Hole` button is pressed to initiate the drawing (and creation) of a hole in the box-girder section. Fig. 4 shows the completed geometry of the cross-section displayed on the screen. Next, it was decided that the reinforcing bars would be size no 11, and have a yield strength of 40 ksi. A concrete compressive strength $f_c^{'}$ of 3 ksi was assumed. To assist in the positioning of flexural reinforcing, the horizontal bar line was set at y = 117.0 inches, three inches from the bottom edge of the section. Ten no 11 bars were then placed within the section, as shown in Fig. 4. Finally, the the flexural capacity of the cross section was computed. It is

1365 kip-ft. in positive bending; this is adequate.

A second box-girder cross section was proportioned for the interior support, where the section is subject to negative bending under all loading conditions. As a starting point, the end span cross section shape was assumed, with the ten no 11 reinforcing bars moved to the top of the section. This gave a flexural capacity for positive bending of 0.0 kip-ft, and a negative bending capacity of -1332 kip-ft. Since the latter quantity is greater than the required -1490 kip-ft, the section was deepened by interactively lowering the endpoints of the bottom edge and hole by six inches. The lower half of Fig. 3 shows the new shape, engineering properties, and flexural strength - for negative bending, it is -1618 kip-ft - of the deepened cross section.

THREE-DIMENSIONAL SOLID MODEL FOR FOOTBRIDGE

The proposed footbridge design consists of prismatic beams over the end spans and central portion of the center span, plus non-prismatic regions over the interior supports. Each non-prismatic region tapers from the shallow section to the deeper section at the support, and then tapers back up to the shallow section within the center span.

BeamTool provides designers with the computational tools to interactively position the end span and interior support cross section shapes along the longitudinal axis of the beam structure, and a sweeping algorithm to connect adjacent cross sections. A three-dimensional model of the footbridge was created from left to right by working through the following step-by-step procedure:

[1] The end span cross section (designed in the previous section) was selected and interactively positioned at z = 0.0 in the side view window (see Fig. 5a). Placement of the

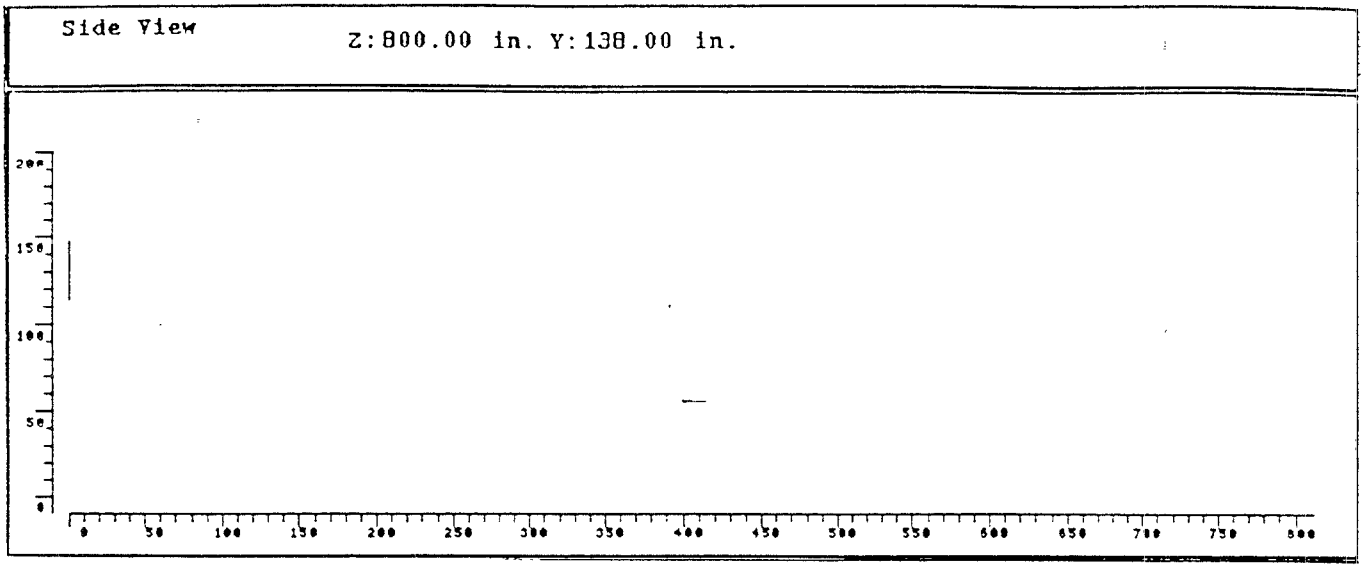


FIG. 5a. Placing the Left End of the Solid

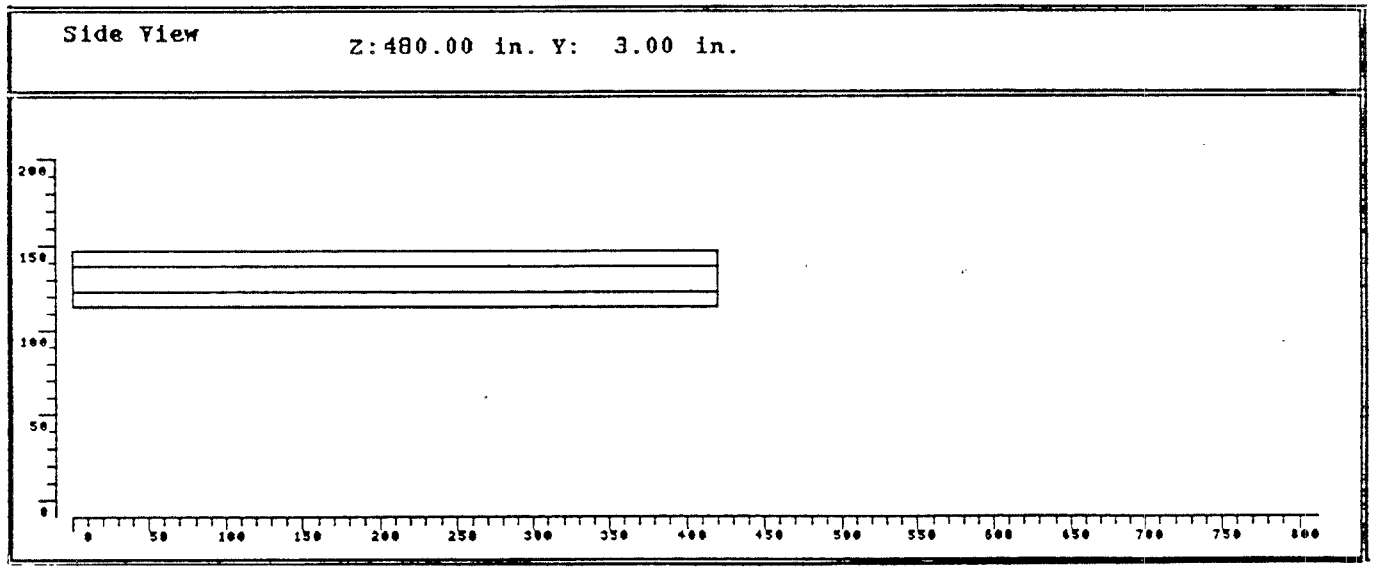


FIG. 5b. The First Portion of the Solid

cross section automatically triggers an operation to generate a complete laminar solid from the cross-section geometry; the generated solid has two faces, but no length.

[2] Two steps are need to generate the solid model for the bridge end span. First, the face of the laminar solid which is oriented outward toward the right is moved to the location of the right end (z = 420 inches). This sweeping operation generates the longitudinal edges of the solid shown in Fig. 5b. Second, the vertices of the right end of the solid are moved to the (x,y) locations of the section selected for the right end. Now the solid takes the shape that the user intended.

[3] A tapered prismatic section is appended to the right hand side of the end span beam segment. First, the interior support cross section design is redisplayed by selecting the `Front/Back` button in the lower control panel. Picking the `Extend Beam Right` button in the upper control panel causes a new set of choices to be displayed, namely `Pick Extension Section`, and `Place Extension Section`. These features are employed to select and position the deeper section over the interior support at z = 480 inches. The underlying geometric model is appended to include the tapered end span/interior support segment.

[4] The extension process is repeated two more times with the shallow section to complete the left half of the bridge, which is symmetrical about the midpoint of the center span. The side view subwindow of Fig. 3 shows the completed solid model of the beam without any reinforcing bars.

Mapping Algorithm for Sweeping Operation. - The algorithm for generating a three-dimensional solid from adjacent cross sections assumes that the loops and vertices of one cross section may be matched to the second cross section with a one-to-one mapping. Such a mapping is well defined when the left and right end sections have the same number of loops (and vertices along the perimeter of each loop), and when the cross-sections are

uniformly oriented, either clockwise or counter-clockwise. If these conditions are satisfied, then a heuristic strategy is used to establish the mapping. First, the halfedge and vertex of each loop with the greatest y-coordinate is determined. If the vertex is not a unique, then the vertex with the lowest x-coordinate of this group is selected. Once the vertices for each loop are identified, the edges of each loop are ordered, and edges inserted between the cross sections. This simple ordering scheme, although admittedly not robust, has been adequate for the generation of beam models.

BAR EDITING SCREEN

The Bar Editing Screen supports the placement of reinforcing bars along the length of the three-dimensional concrete beam, and is shown in Fig. 6. The labeled components of the Bar Editing Screen are: (a) Main Control Panel, (b) Window with Side View of Beam, (c) Bar Bending Dials, (d) Window with Cross-Section of Beam, (e) Information Window, and (f) Lower Control Panel for Bar Editing Operations. Notice that both top and side views of the three-dimensional model are shown, thereby allowing for complete control of placing the bars within the solid using the mouse. A cross-sectional view is also shown. Its purpose is to allow the user to cut the beam at any given point, and evaluate the flexural strength of the cross-section.

Specification of 3-D Reinforcement Trajectories. - Reinforcing bar trajectories are constrained to lie in either a vertical or horizontal plane parallel to the longitudinal axis of the beam. This constraint allowed the writers to develop point-and-click editing operations for the placement of bars. For instance, a horizontal bar plane is set by buttoning on `Set Bar Plane`, moving the cursor to the desired y-coordinate in the side view window,

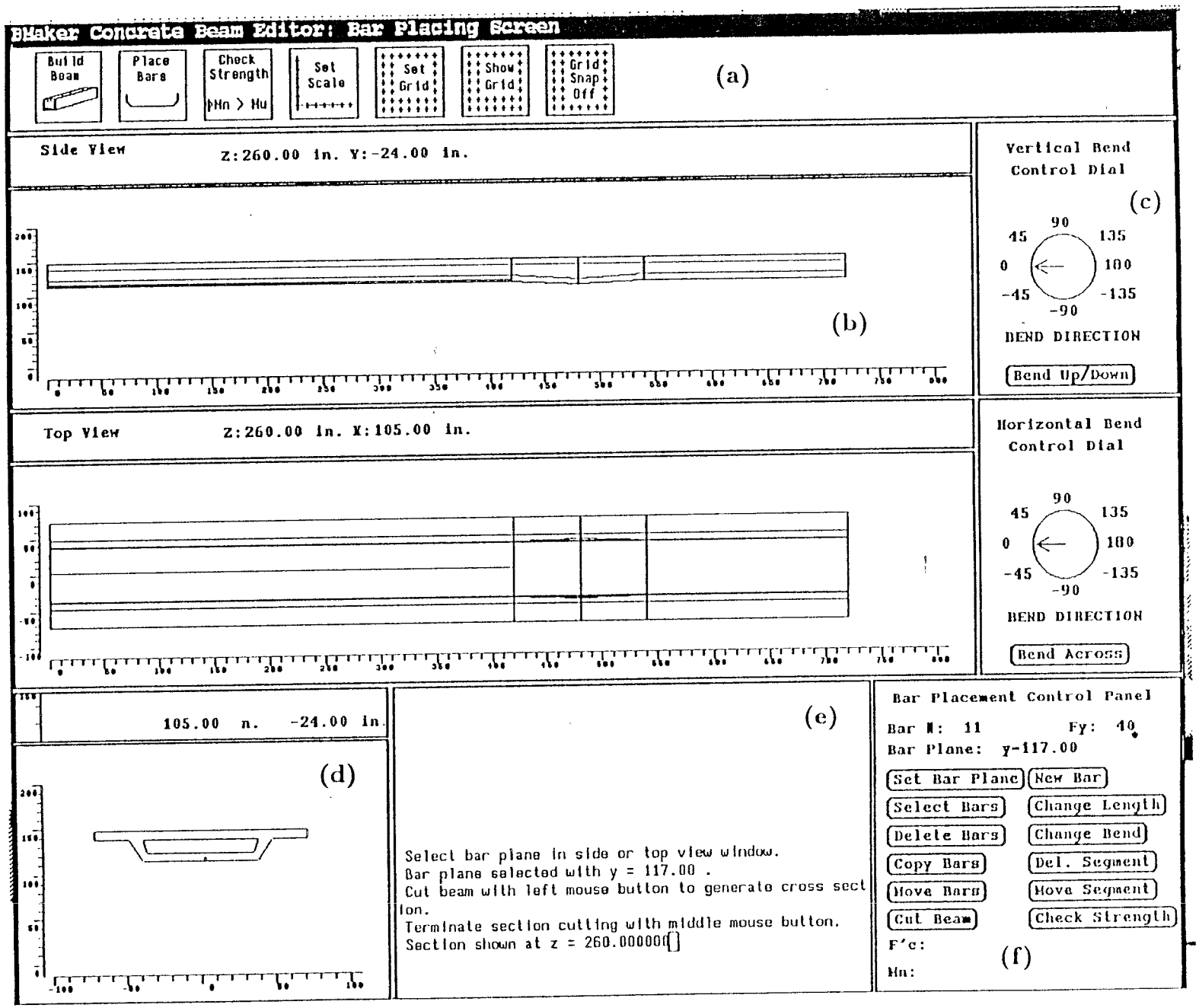


FIG. 6. - The Bar Editing Screen

and buttoning the mouse again. Once the horizontal plane is fixed, the longitudinal and horizontal coordinates of reinforcing bars may be specified in the top view window. Bars of more than one straight segment are positioned by sequentially locating the first end of each segment with the left mouse button, and the matching endpoint with the middle mouse button. The use of the curved segments has yet to be implemented in the software.

PLACEMENT OF REINFORCING IN FOOTBRIDGE

The three dimensional layout of reinforcing in the box girder bridge is based on the end span and interior support cross section designs. For example, flexural reinforcement in the end span beam segment is positioned by recalling that the cross section design contains ten number 11 bars in the horizontal plane y = 117 in. After the bar size and material yield strength have been input, and the horizontal plane set, the first of ten bar trajectories is specified by selecting the `New Bar` button, moving the cursor to location x = 3 in., z = 0 in. the top view window, pressing the left mouse button, moving the cursor to x = 3 in, z = 420 in., and finally, pressing the middle mouse button. The first bar is drawn in the side and top view windows, and extracted view of the bridge cross section (see Fig. 6). Because the trajectory shape of all the lower reinforcing bars are identical, the remaining bars in this region are created by selecting and positioning (see `Select Bars` and `Copy Bars` in the lower control panel) copies of the first bar at the required x coordinate.

The same procedure - with horizontal bar plane y = 144 in. - is repeated to position flexural reinforcement over the interior supports. Fig. 7 shows details of the fully reinforced bridge.

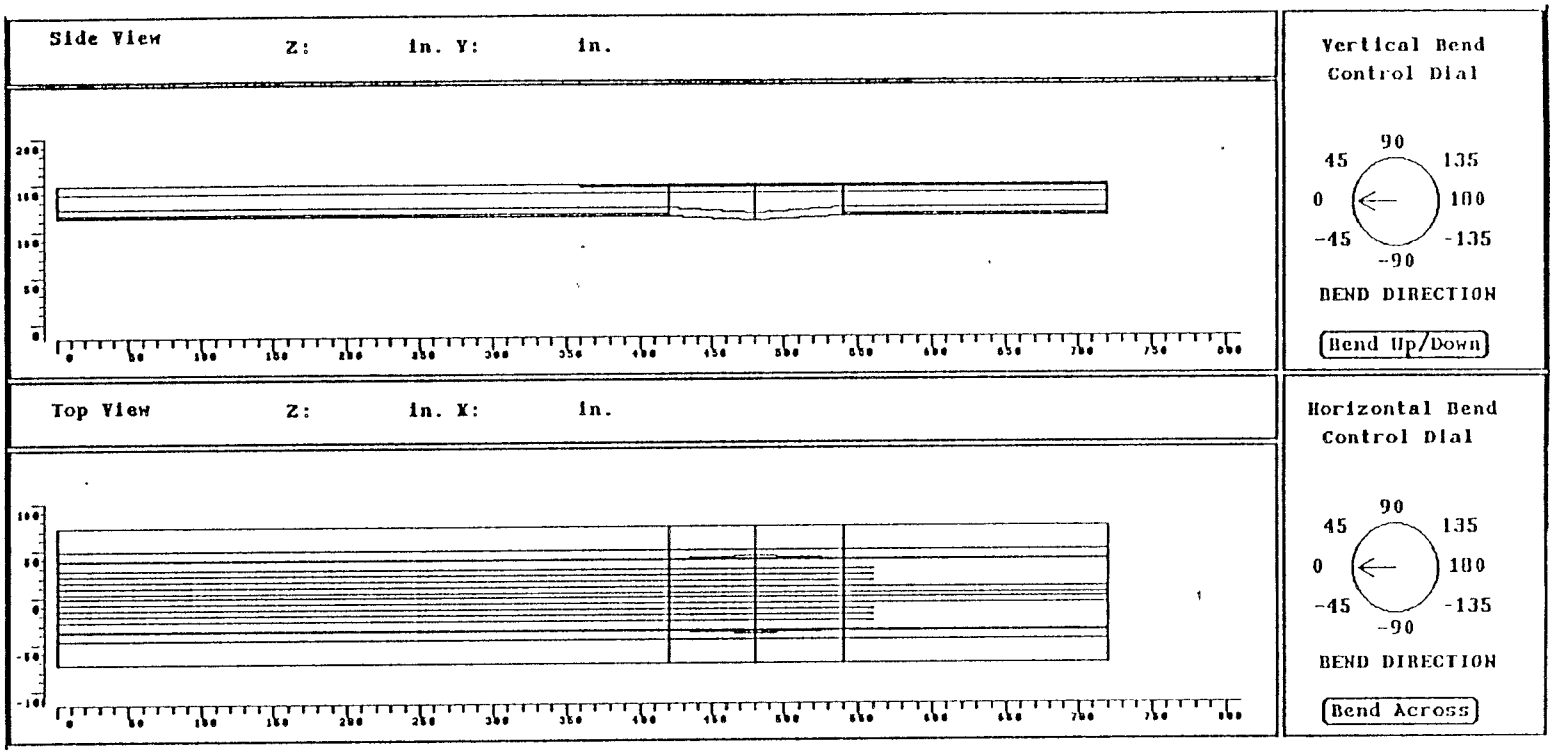


FIG. 7 The Reinforced Bridge

STRENGTH EVALUATION SCREEN

The Strength Evaluation Screen, shown in Fig. 8, provides a visual comparison of the actual flexural strength of the beam structure, versus the design envelope of required strength. Its main components are: (a) Main Control Panel, (b) Window with Side View of Beam, (c) Window with Graph of Flexural Strength Capacity, (d) Lower Control Panel for Strength Evaluation Operations, and (e) Information Window.

The control panel to the right of the strength diagram is used to specify the concrete compressive strength, and to input the name of files containing (z, M_n) coordinates in the moment envelope. The moment envelope coordinates are read and plotted for comparison with the actual strength of the beam. For the reasons stated above, neither the solid model, nor the reinforcing bars can be edited at this level. If the user is dissatisfied with the flexural strength of the beam, he or she must return to the Bar Editing Screen to edit the reinforcing bar trajectories.

Locations for Strength Evaluation. - An envelope of positive and negative flexural strength along the length of the beam is constructed from strength evaluations of cross sections extracted from the beam structure. Cross sections are extracted from the beam in the following locations:

[1] At all discontinuities, i.e. where a sections are placed to build up the beam.

[2] At each end of every bar. If more than one bar ends at a given cross section location, then the cross section is extracted and evaluated only once.

[3] At the point along each bar where it achieves its full development length. Only basic development lengths have been used in the prototype implementation. Moreover, the

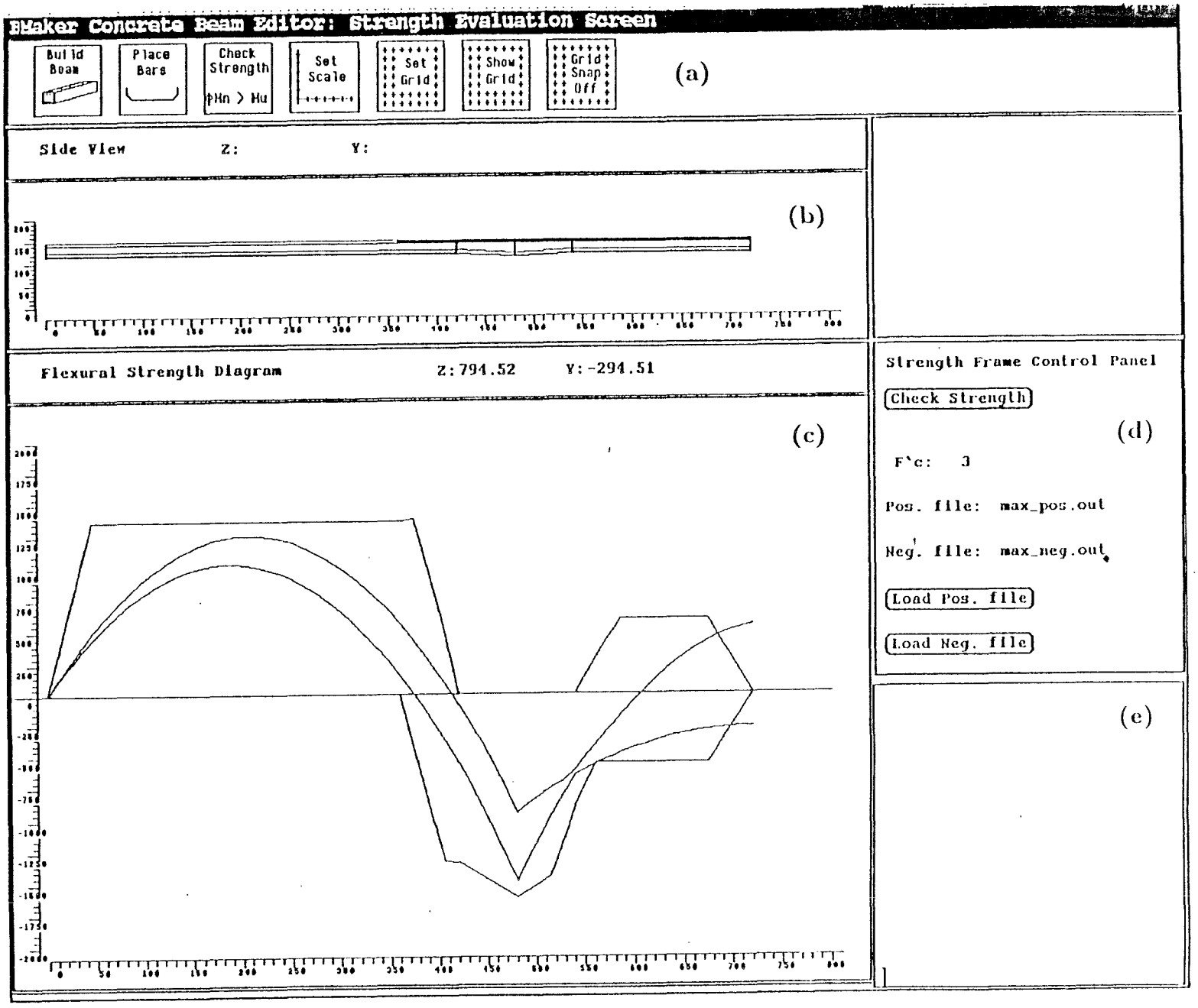


FIG. 8. - The Strength Evaluation Screen

development length factor is employed only when the cross-section is generated by cutting a three-dimensional beam model. Strength analyses performed on preliminary cross section designs at the two dimensional level assume all of the bars are completely developed.

When a user selects the `Check Strength` button on the Strength Evaluation Screen, these criteria are used to construct an ordered list of locations for cross section extraction and flexural strength evaluation. The algorithms for extracting reinforced cross sections from the beam structure, and evaluating their flexural strength are as described in Austin (1992).

FLEXURAL CAPACITY OF FOOTBRIDGE

Fig. 8 shows the envelope of flexural capacity circumscribing the applied moment envelope. Because the cross-sections of the beam were designed carefully for the critical sections, we have arrived at an adequate bridge design in the first iteration. The sample design is now complete for flexural reinforcement.

CONCLUSIONS AND FUTURE WORK

The BeamTool software system is a first cut at implementing the data structures and algorithms described in the companion paper (Austin 1992). BeamTool works well; the writers are satisfied that geometric modeling techniques are a sound basis for developing interactive systems to support RC design.

Further work is needed to extend the scope of design problems BeamTool can handle. Future versions should allow for the definition of external loads, and the placement of shear reinforcement. Finite element analysis packages should be used to compute distributions of bending moment and shear forces. A valid criticism of the prototype implementation is

that all new elements must be assembled from the section level. BeamTool should be setup so that designer may work with combinations of customized and standard elements. In the latter case, the topology of regularly used section shapes (e.g., rectangular beams, I-beams, T-beam) could be defined *apriori*, with the geometry (or shape) of each instance of a beam object being completely determined by the values assigned to a set of parameters that accompany the model. For a system of this type to be implemented, however, considerable work is needed on data structures, algorithms, and user interfaces tailored to the assembly, editing and checking of structural assemblies. This is not a trivial step.

Future work should also focus on providing designers with the ability to control and customize the response of the BeamTool system to editing operations, and to impose a hierarchy of (geometric and design) rule checking. As a minimum requirement, the system should support hard and soft rules, and allow the checking of rules to be turned on/off. A hard rule is one that must be satisfied at all times; any editing operation that results in hard rule violation would be immediately rejected by the system. Violations in soft rules are less catastrophic. An appropriate system response would be to signal the error to the user, with the expectation that it would be manually repaired at a later time.

ACKNOWLEDGEMENTS

This work was supported in part by the Minta Martin Foundation, NSF's Engineering Research Centers Program : NSFD CDR 8803012, and NSF Research Initiation Grant NSF BCS 8907722. This support is gratefully acknowledged. The views expressed in this paper are those of the authors, and not necessarily those of the sponsors.

APPENDIX I. - REFERENCES

[1] Austin M.A., and Preston J. (1992), "Solid Modeling Based Design of RC Beams Part 1 : Data Structures and Algorithms," *Journal of Computing in Civil Engineering*, This issue.

[2] Kalay Y.E. (1989), "The Hybrid Edge : A Topological Data Structure for Vertically Integrated Geometric Modeling," *Computer Aided Design*, Vol. 21, No. 3, pp. 130-140.

[3] Sun Microsystems (1988), "SunView 1 Programmers Guide," Copyright 1982, 1988, Sun Microsystems Inc.