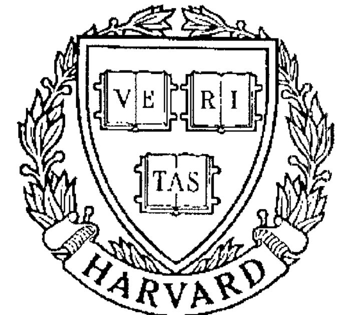
Technical Research Report

S Y S T E M S
R E S E A R C H
C E N T E R



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

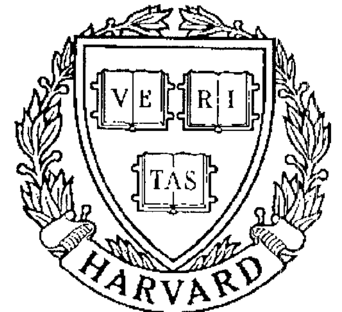
Pre-Processor for Finite Element Analysis of Highway Bridges

by S.L. Creighton, M.A. Austin and P. Albrecht

TR 90-54

Technical Research Report

S Y S T E M S
R E S E A R C H
C E N T E R



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

Research support for this report has been provided by Federal Highway Administration

Pre-Processor for Finite Element Analysis of Highway Bridges

by S.L. Creighton, M.A. Austin and P. Albrecht

SRC TR 90-54

Pre-Processor for Finite Element Analysis of Highway Bridges

Submitted to

Structures Division
Turner-Fairbanks Highway Research Center
6300 Georgetown Pike
McLean, Virginia 22101

by

Steven L. Creighton [1], Mark A. Austin [2], and Pedro Albrecht [3]
University of Maryland
College Park, Maryland 20742

August 1990

[1]Graduate Research Assistant, Department of Civil Engineering & Systems Research Center
[2]Assistant Professor, Department of Civil Engineering & Systems Research Center
[3]Professor, Department of Civil Engineering

ABSTRACT

Historically, the lack of interactive pre-processors to setup tedious finite element problem descriptions has hindered the use of the finite element method for bridge analysis. The work performed in the present study mitigates this problem via the use of highspeed engineering workstations.

This report presents an interactive, graphically based pre-processor for the analysis of highway bridges with the finite element method. With the pre-processor, bridge design engineers can use both keyboard and mouse styles of interaction to describe bridge geometries. The UNIX tool YACC has been used to create a command language for describing and manipulating of structural geometries, material types, loads, and boundary conditions.

The pre-processor was successfully used to create input files in a format acceptable to the finite element analysis program ANSYS. In fact, less than 20 minutes was needed to create a finite element model of the prototype bridge tested at the Turner-Fairbanks Laboratory in Langley, Virginia, in a joint FHWA-AISI project.

This report describes the development of the pre-processor in words that can hopefully be understood by civil engineers and computer scientists alike.

Contents

1 Introduction 1
1.1 Problem Description 1
1.2 Scope of Work 4
1.3 Reading Level 4

2 Approach to Problem Solution: Objectives and Strategy 5
2.1 Introduction 5
2.2 Pre-Processor Provisions 5

3 Problem Solution Environment 7
3.1 Introduction 7
3.2 Hardware 7
3.3 Software 8
3.3.1 User Interface 8
3.3.2 C Language and UNIX 10

4 The XBUILD System for Bridge Analysis 11
4.1 Introduction to XBUILD 11
4.2 Software Architecture 11
4.2.1 Window Layout 12
4.2.2 Design Libraries 13
4.2.3 Tasks and Procedures 13
4.3 Command Language 15
4.3.1 Command Interpreter Design 15
4.3.2 XBUILD's Program Modes and Prompting Mechanism 18
4.3.3 Description of Geometries 19
4.3.4 Design Attributes 24
4.4 Mouse 25
4.4.1 Introduction 25
4.4.2 Three Dimensional Graphical Display 25
4.4.3 Graphical Display Manipulation 26
4.4.4 Grabbing Objects 31
4.4.5 Pull-Down Menu 31
4.5 Query, Modification, and Grouping of Data 34
4.5.1 Data Query 34
4.5.2 Data Modification 34
4.5.3 Creation of Structural Objects (Data Grouping) 34
4.6 Data Structures 35
4.6.1 Introduction 35
4.6.2 Linked Lists 35
4.6.3 Arrays 38
4.6.4 Hash Table 40
4.6.5 Concluding Remarks 42

5 FHWA Test Bridge Example 43

6 Conclusions 54

7 References 57

CHAPTER
ONE

INTRODUCTION

1.1 Problem Description

Few factors are having a greater impact on our view of computer-aided design of highway bridge structures than rapid advances in computer hardware, the introduction of advanced color display workstations to the marketplace, and a significant drop in the price/performance ratio of engineering workstations. When these desk-top workstations were first introduced to the engineering community, a substantial gap existed between the capabilities of the new hardware and the capabilities of bridge design software to exploit this potential.

Unfortunately, solutions to this problem are more complicated than the simple upgrading of traditional design methods to the level of the new hardware. This is because previous generations of software systems were often developed with the intention of providing good design answers with only a moderate amount of computation. In current engineering practice, for example, multi-girder highway bridges are analyzed using greatly simplified analytical models in accordance with specifications established by the American Association of State Highway Transportation Officials (AASHTO). These models assume that (1) each girder is idealized as a two-dimensional beam acting separately from the other beams; (2) the idealized girder resists a fraction of the wheel loads as calculated with simplified formulas for wheel load distribution; (3) in composite bridges, the idealized girder acts compositely with an effective width of the deck as calculated with a semi-empirical formula.

Since slide rules and hand calculators were the dominant forms of computational assistance during the 1950's and 1960's, the aforementioned assumptions were necessary simply to get the job done. During the past decade, the profession has paid a high cost for the convenience of reduced computational effort, often settling for designs that were overly conservative. For example, the moments induced by truck loading on the two-span-continuous, three-girder FHWA-AISI prototype bridge were up to two times smaller when calculated

with the finite element method than with the AASHTO method [29]. In another example from a FHWA sponsored study, the stresses calculated with the finite element method were about three times smaller than those calculated with the AASHTO method [7]. The latter case involved two simple-span bridges on Pennsylvania Route 33, one with six steel girders and the other with six concrete girders.

Previous investigators have worked hard to improve the accuracy of bridge analysis. In a recent $300,000 study sponsored by the National Cooperative Highway Research Program (NCHRP), an extensive parametric analysis resulted in more accurate semi-empirical formulas for the distribution factor in slab-on-girder bridges [23]. Elnahal, Albrecht, and Cayes [8] applied these formulas to the FHWA-AISI bridge, shown in figure 1.1, and found that they lead to maximum moments about 5 to 53 % higher than those determined from experimental work. These distribution factors are only valid for interior girders. Those for the exterior girders, which controlled the design of the prototype bridge, are still based on the AASHTO simple beam model. The findings clearly show the limitations of the distribution factor approach.

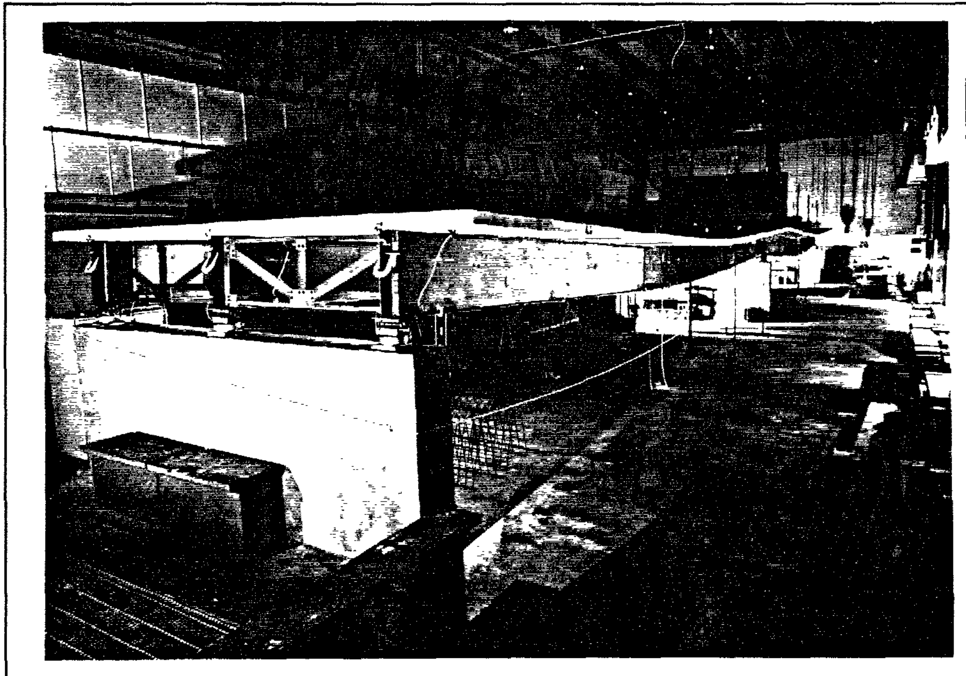


Figure 1.1. FHWA test bridge: Turner-Fairbanks laboratory.

Other attempts to improve the accuracy of bridge analysis involved the modeling of the bridge as a grid system to arrive at a more accurate distribution of wheel loads [2]. Longitudinal girders and transverse diaphragms made up the grid. Although grid models

are a step in the right direction, the designer still has to assume an effective deck width for each girder and an equivalent moment of inertia for the diaphragms.

Today, the picture has changed. Now that high-speed desk-top workstations are readily available, conservative assumptions made for the sole purpose of simplifying analysis methods are no longer justifiable. Needed instead is finite element analysis to more accurately predict bridge strength.

In a step towards improving the accuracy of bridge analysis, Elnahal, Albrecht and Cayes [8] modeled the FHWA-AISI test bridge with the finite element analysis program "ANSYS" [21]. In this way they were able to consistently predict the experimentally determined reactions and moments with an accuracy of ±10%.

Furthermore, at the University of California, Berkeley, a 20-year research program has developed a finite element computer program, called CELL, for analysis of reinforced and pre-stressed bridges. The first version, issued in 1970, was capable of analyzing box girder bridges with arbitrary plan geometry and constant depth [8]. Subsequent modifications to CELL led to a version called "CELL4", released in 1985, which performs linear-elastic analysis of post-tensioned, cellular structures of arbitrary plan geometry and constant depth [27].

Although several general-purpose finite element analysis programs are commercially available, such as ANSYS [21], NASTRAN [22], SAP [34], and STRUDL [14], they all have short-comings that limit their wide-spread application to the analysis of highway bridges. Often, these programs are not well suited for bridges and contain numerous bulky features not needed for bridge analysis.

Additionally, a search of the literature has found neither commercial nor research oriented programs that have pre- and post-processors adequately suited for finite element analysis of bridges. Nearly all currently available bridge software systems lack good facilities to set up, edit, and query input data. In the proceedings of a workshop funded by the National Science Foundation, Research Needs for Short- and Medium-Span Bridges, Scordelis states, "It has become evident to me that if these new analytical tools (referring to finite element analysis programs) are to be used by designers in the U.S., special purpose pre- and post-processing packages, written especially for bridge analysis, need to be developed as part of research projects undertaken in this field" [5].

1.2 Scope of Work

The idea for this project arose from the need to mitigate the deficiencies mentioned in section 1.1. This report presents the development of a prototype pre-processor for generating three-dimensional finite element meshes, applying stationary loads, and specifying boundary conditions for straight, non-skewed, multi-girder highway bridges. Not included are discussions of analysis packages, post- processors, nor the extent to which some finite element models may be better than others.

1.3 Reading Level

Readers of this text are assumed to know the fundamentals of finite element analysis, C programming language, and UNIX programming environment.

CHAPTER
TWO

APPROACH TO PROBLEM SOLUTION

2.1 Introduction

As mentioned in Chapter 1, the absence of bridge oriented finite element analysis pre-processors within the engineering profession is a major reason FEA methods have not gained wider acceptance in analysis of bridges. Many engineers continue to analyze bridges with simplified methods, such as the AASHTO method, simply because they are easier to use. This situation is unlikely to improve until software systems are developed that assist engineers with FEA. With this background in mind, this chapter outlines a wish-list of features a pre-processor should have to make FEA methods competitive with traditional analysis methods.

2.2 Pre-processor Provisions

An effective pre-processor must provide mechanisms for rapid and concise generation of nodes and elements, external loadings, and support conditions. For multi-girder highway bridge analysis, methods are required for describing the geometric layout of girders, deck, and cross braces (diaphragms). Additional features that should be incorporated into the pre-processor are described below.

Graphical Echo of Finite Element Model

The pre-processor should allow the designer to view a finite element model during all stages of its creation. Visual feedback helps to check commands and detect typographical errors.

Storage, Query, and Modification of Data

A pre-processor must provide computationally efficient algorithms and data structures for storing, querying, and modifying input data. Ways of storing and querying data are needed

for two reasons. First, the user may not remember some of the data input earlier and need to retrieve it. Second, the user may need to verify that information is correctly stored in the database.

Procedures for modifying input data are also important. Bridge analysis and design is typically an iterative process in which various parameters may have to be updated and tested on a trial and error basis before a suitable model results.

A programmer's ability to quickly modify input data shortens the turnaround time of successive iterations in the analysis/design process. It should be noted that management of the different versions of a design, in the iterative search for an optimal version, is a concern shared by all engineering disciplines. A detailed explanation of research currently underway on design version management can be found in the recent paper by Katz et al. [18].

Automatic Generation of Data Files

A good pre-processor must be able to interface easily with analysis routines. Two approaches are possible. Either the pre-processor can have analysis routines attached to it, or it can transfer information from its database to files compatible with existing FEA software.

Tools for Measurements and Calibration

Bridge designers need ancillary computational tools (such as a calculator) and mechanisms for quickly converting between SI and US units of measure. Additionally, on-line tools are needed to provide the user with a quick reference to information about (1) the cross-sectional geometries of girders and AISC [2] sections, (2) materials and their properties, and (3) a help menu of available commands and their use.

Save and Reload Data

A well designed pre-processor must be able to save and reload data.

Ease of Expansion

The architecture of the pre-processor should lend itself to easy expansion without having to substantially modify it or requiring an in-depth knowledge of its source code. Also, the pre-processor should be written in a computer language that is highly portable and capable of linking with routines written in different languages.

CHAPTER
THREE

PROBLEM SOLUTION ENVIRONMENT

3.1 Introduction

Modern engineering workstations provide engineering designers with computational speed once restricted to mainframes, highly graphical user interfaces, and network communication. Before these workstations were introduced to the marketplace, FEA had to be performed in batch mode on personal computers, mini main-frames, or main-frames. Compared to the benefits offered by today's engineering workstations, each hardware option had significant disadvantages. A common shortcoming of personal computers, for example, is insufficient memory to perform FEA of structures with many degrees of freedom. This limitation may be overcome by resorting to special techniques for solving only parts of a problem in memory at one time [35]. In comparison to user interfaces available on modern workstations, the terminals through which most main frame and mini computers are accessed provide interfaces incapable of simultaneously offering both powerful graphical capabilities and direct methods for data input. Also, the costs of main frames and mini computers is more than what most design firms can afford.

3.2 Hardware

After evaluating various hardware alternatives, the family of SUN Workstations was found best suited for development of a pre-processor for FEA of bridges. SUN's family of machines includes the SUN2 series, SUN3 series, SUN4 series, the SUN386i, and the newly introduced "SPARC Station".

Apart from their fast computational speed - the "SPARC Station" was benchmarked at 12.5 MIPS [30] - SUN Workstations have many advantages. For example, they: (1) are widely used in both industry and universities; (2) operate under the UNIX operating system, which provides good general support for software development; (3) support windowing and

multi-tasking; and (4) come equipped with a high resolution bit-mapped graphic display and a separate graphics processor that facilitate the development of interactive user interfaces.

3.3 Software

3.3.1 User Interface

The importance of an effective user interface cannot be over emphasized. With interactive computing environments becoming indispensable in solving difficult problems, growing evidence suggests that ease of use is at least as important as functionality in determining the likely success of an application program [9]. Foley and VanDam [9] point out that ease of use varies in accordance with a user's familiarity with a software package. Since beginners and more experienced users often differ on what they think is the easiest - or most convenient - way to interact with a program, software should be developed to support multiple styles of interaction.

Styles of Interaction

An important criterion in developing the pre-processor software was to create a user interface that allows engineers to freely switch between menu-based and keyboard styles of interaction. Menus and other mouse activated devices are preferred by beginners because they can guide the user through the steps needed to input data. Experienced users, on the other hand, prefer command languages because they type a command without being presented an explicit set of alternatives [9].

Hutchins et al. [16] describe the ways humans interact with computers with the metaphors "conversational world" and "model world". These metaphors correspond to sequential and asynchronous dialogues, respectively [15]. In the "conversational world", the user typically describes the task to be performed with a command language. This kind of dialogue is called "sequential" because it moves in a predictable manner from one part of the dialogue to the next. Sequential dialogue allows both users and developers to visualize a specific logical sequence of events. It includes request-response interactions, typed command strings, navigation through networks of menus, and other data entry [15]. In the "model world", the user "grabs" visual representations of objects and manipulates them with the mouse. This style of interaction is illustrated in figure 3.1. Notice that the box in figure 3.1(a) is not large enough to enclose the baseball player. The user can "grab" one of the "handles", in this case the lower right-hand corner as shown in figure 3.1(a), and stretch the rectangle by moving

the mouse until the rectangle is the desired size and at the desired place, figure 3.1(b) [15].

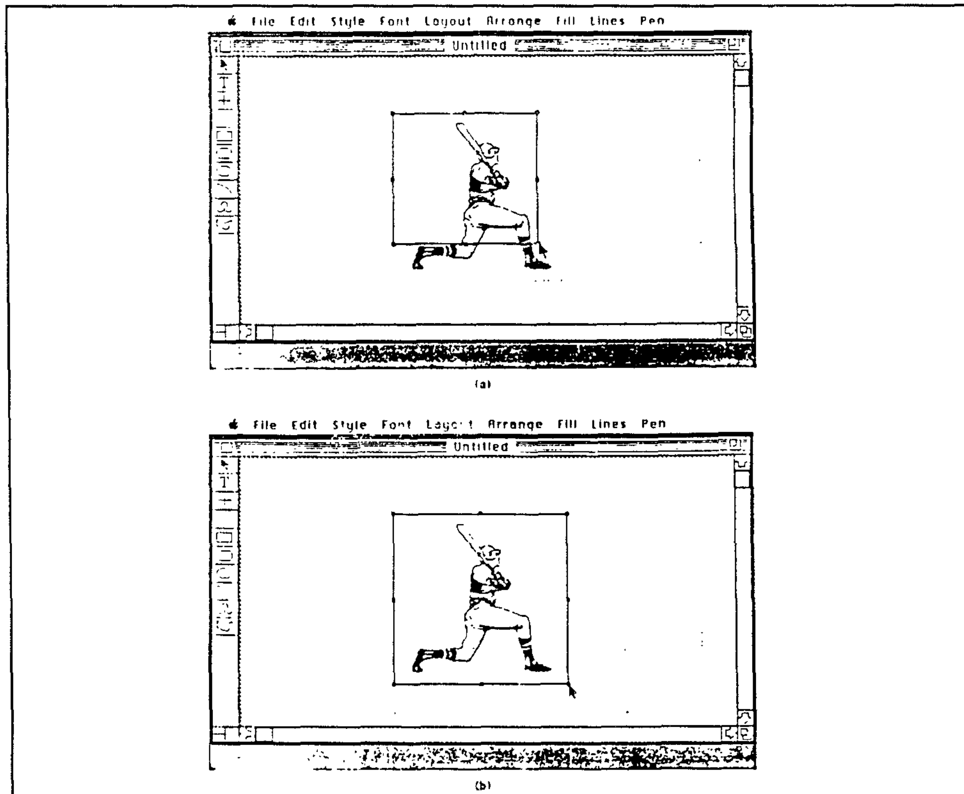


Figure 3.1 Asyncronous Dialogue: Grabbing With the Mouse

SunView: A Brief Description

SUN Workstations provide software developers with a user interface toolkit called "SunView" (SUN Visual/Integrated Environment for Workstations), which supports both the "conversational world" and "model world" styles of interaction.

SunView offers visual building blocks such as windows, pull-down menus, scrollbars, buttons, terminal emulators, and icons that can be arranged on the screen and customized to best meet the needs of the application program [30]. The SunView building blocks may be setup to respond to a variety of user initiated "events." Typical SunView events include, toggling a pull-down menu item, pushing (or releasing) a mouse button, and depressing keys on the keyboard. Each event has a unique identification and calls on specific subroutines. For example, toggling the pull-down menu item labeled "clear screen" calls the subroutine "clear_screen()".

3.3.2 C Language and UNIX

The pre-processor was written in the C language under the UNIX operating system. Among the benefits offered by C are (1) flexible data structures for storing data, (2) dynamic allocation and utilization of memory, (3) easy manipulation of pointers to memory, (4) compatibility with graphical user interface tool kits such as SunView [30] and X Windows [10], and (5) portability to many other computers.

An important reason for selecting the UNIX operating system was the availability of convenient tools for writing command languages. As is discussed in Chapter 4, the command language for the pre-processor was written with the UNIX tool "Yet Another Compiler Compiler" (YACC) [17] .

CHAPTER

FOUR

THE XBUILD SYSTEM FOR BRIDGE ANALYSIS

4.1 Introduction

The XBUILD pre-processor may be used to prepare input data for finite element analysis of highway bridges. Version 1 of XBUILD employs the SunView window toolkit and runs on the SUN family of workstations. Still, it was named "XBUILD" because the authors will eventually modify it to run in the "X-Windows" environment [10].

This chapter presents an overview of XBUILD, and describes the issues that were considered in its design. The main components of the XBUILD software architecture are described in section 4.2. Sections 4.3 through 4.5 cover XBUILD's user interface, and sections 4.6 through 4.7 its database. The User's Guide, a companion document to this report, contains instructions for inputting data.

4.2 Software Architecture

Figure 4.1 shows a schematic of XBUILD's main components. These include: (1) SunView windows as part of an event based user interface; (2) design libraries containing finite element attribute data; (3) tasks (and procedures) that may be called by both command language and three-button mouse; and (4) data structures for storing finite element input data and intermediate results.

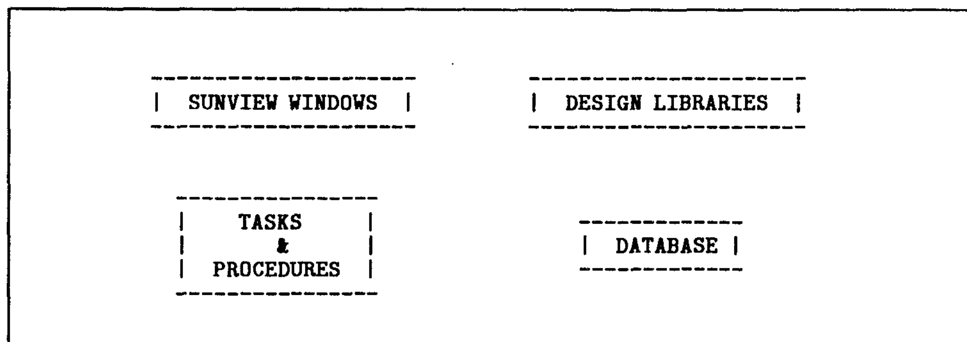


Figure 4.1. Schematic of XBUILD software components.

4.2.1 Window Layout

XBUILD's visual interface consists of a "parent" window and several "child" subwindows. The subwindows shown in figure 4.2 include: (1) a terminal emulator (TTY) for entering commands with the keyboard; (2) a canvas for displaying the xy elevation view of a structure; (3) a canvas for displaying the xz plan view of a structure; (4) a canvas for displaying the zy section view of a structure, in icon form; (5) a movement pad to change viewing planes; and (6) pull-down menu items for ease of inputting data.

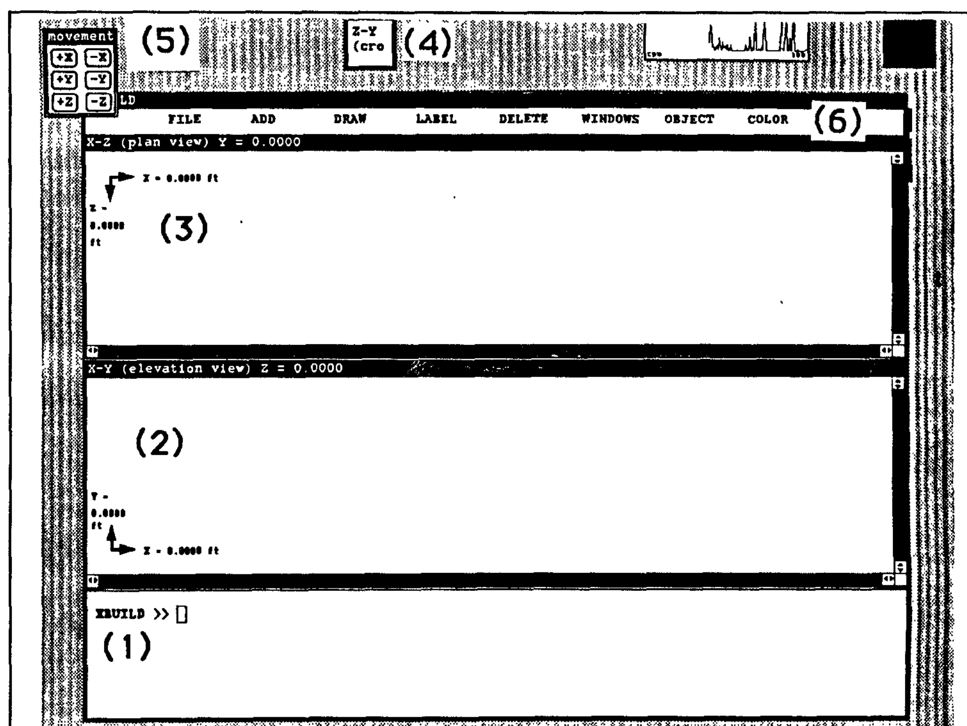


Figure 4.2. Window layout.

4.2.2 Design Libraries

XBUILD's design libraries contain: (1) beam/column element cross section property data for over 200 AISC sections and several girders; and (2) property data for several engineering materials. The information is read from the external files "material.dat" and "sections.dat" when XBUILD is first activated.

4.2.3 Tasks and Procedures

A significant factor influencing the development of XBUILD was the observation that most commands required for engineering design may be expressed as an "action" applied to an "object." A concise way of formulating this idea is to enter a <verb> followed by a <noun>. For example, a designer can convey to XBUILD that nodes need to be added by simply typing add node, or, toggling an "add node" pull-down menu item. Table 4.1 summarizes the verbs and nouns currently supported by XBUILD.

Verbs & Nouns	
<verb>	add, delete, label, fix, move, link, create, change, copy, get_data, change, set, print, save
<noun>	node, offset_node, bc_elmt, plate4, point_load, linear_load, area_load, bcond, object, ANSYS

Table 4.1. Verbs and nouns.

Unfortunately, simply entering a <verb> <noun> combination does not convey to the computer "which" design items or "where" the verb-noun combination is to be applied. Collectively, the terms "which" and "where" define the "scope" of the required command. New nodes cannot be added, for example, until their locations are specified. Similarly, existing nodes cannot be labeled until their numbers are given. As will be explained in the following sections, XBUILD has several command language and mouse features that allow designers to specify exactly "which" items are to be acted on, or, "where" a command is to be applied.

Table 4.2 lists the tasks XBUILD can accomplish together with the interface mechanisms available to execute them. Notice that all of these tasks are expressed as <verb> <noun> combinations.

Task : <verb> <noun>	Language	Mouse	Both Required
add node	*		
add offset_node	*		
add bc_elmt	*	*	
add plate4	*	*	
add point_load		*	
add linear_load		*	
add area_load		*	
delete node		*	
delete bc_elmt		*	
delete plate4		*	
label node		*	
label bc_elmt		*	
label plate4		*	
fix bcond		*	
move node	*		
link offset_node	*		
create object			*
print ANSYS	*		
copy object	*		
get_data bc_elmt	*	*	
get_data plate4	*	*	
get_data point_load	*	*	
get_data linear_load	*	*	
change bc_elmt	*	*	
change plate4	*	*	
change point_load	*	*	

Table 4.2. Styles of interaction for XBUILD tasks.

4.3 Command Language

The command language written in this study enables users to enter commands at descriptive prompts. The commands:

[1] Create finite element models consisting of nodes, elements, boundary conditions, and loads.

[2] Specify material types and cross-sectional parameters for various elements.

[3] Select engineering units.

[4] Store, query, modify, and group input data.

[5] Save and load created models.

[6] Print ANSYS compatible data files.

[7] Create temporary sub-structure objects.

4.3.1 Command Interpreter Design

The interpreter for XBUILD's command language was written with the UNIX tool YACC [14], which produces the C source code for a function capable of parsing streams of commands. YACC generates this "parser" with a hierarchical set of rules defined by the programmer.

An important consideration in the design of the language's command interpreter was to find ways of providing users with a wide variety of permissible design commands, without requiring them to type lengthy, complicated instructions. Indeed, previous work has shown that for two systems that are functionally equivalent, the one that has the simpler syntax produced fewer errors and was more easily learned [25].

XBUILD's YACC grammar has over 150 rules. Rather than give an exhaustive description of each rule, only the rules for evaluating arithmetic expressions, specifying engineering quantities with units, and generating numerical lists are discussed. This sub-set of rules was selected because they are used extensively in commands for generating finite element meshes. More specifically, numerical lists are used to answer the questions of "which" and "where" raised in section 4.2.3. However, since a numerical list is actually a list of values, and XBUILD's YACC grammar defines a value with combinations of arithmetic expressions and engineering units, the latter are presented first.

Evaluation of Arithmetic Expressions

Figure 4.3 shows the YACC grammar for evaluating arithmetic expressions. The symbol ::= is read as "is" and separates left and right hand side components of the grammar. The symbol | is read as "or" and separates the choices of rules. The symbols < and > enclose variables representing constructs. The rules are numbered for explanation purposes.

```
<expr>::=  <number>           (rule 1)
        |  <expr> + <expr>    (rule 2)
        |  <expr> - <expr>    (rule 3)
        |  <expr> * <expr>    (rule 4)
        |  <expr> / <expr>    (rule 5)
        |  '(' <expr> ')'     (rule 6)
        ;
```

Figure 4.3. Arithmetic expressions in YACC grammar.

According to these rules, an arithmetic expression can be a number (rule 1), an expression added to an expression (rule 2), an expression subtracted from an expression (rule 3), an expression multiplied by an expression (rule 4), an expression divided by an expression (rule 5), or an expression in parenthesis. Although the details are not shown in figure 4.3, rule 6 has the highest precedence for evaluation, rules 4 & 5 have the 2nd level of precedence, and rules 2 & 3, the third level of precedence. rules of equal precedence are evaluated from left to right.

```
XBUILD >> {  5*6 }
XBUILD >> Value is 30
XBUILD >> {  6+(2*4)/2  }
XBUILD >> Value is 10
```

Figure 4.4. XBUILD's calculator utility.

The script of code shown in figure 4.4 is taken from XBUILD's calculator utility. It demonstrates the combined effect of rules 1 through 6 and the imposed hierarchy of evaluation.

Units

Quantities in bridge design are often specified in a variety of units. For example, the moment of inertia is usually specified in^4, and the length of a girder in feet. The command language recognizes the quantities length, force, area, force per unit length (linear load), force times

length (moment), and pressure. It accepts most SI and US units for these quantities. The rules below define each quantity.

```
<quantity> ::= <length>
             | <force>
             | <area>
             | <moment>
             | <linear_load>
             | <pressure>
             ;

  <length> ::= <expr> LENGTH
             ;

   <force> ::= <expr> FORCE
             ;

    <area> ::= <expr> AREA
             | <expr> LENGTH * LENGTH
             | <expr> LENGTH^2
             ;

  <moment> ::= <expr> FORCE * LENGTH
             ;

<linear_load> ::= <expr> FORCE / LENGTH
             ;

<pressure> ::= <expr> PRESSURE
             | <expr> FORCE / AREA
             | <expr> FORCE / LENGTH^2
             | <expr> FORCE / ( LENGTH * LENGTH )
             ;
```

Figure 4.5. YACC rules for units.

For example, "5 sqin", "5 in^2", and "5 in*in" are all acceptable ways of expressing five square inches. Similarly, "10 psi", "10 lb/sqin", "10 lb/in^2" and "10 lb/(in*in)" each express ten pounds per square inch. Thus, quantities are defined by either single word descriptors or arithmetic combinations of a quantity's fundamental sub-units.

Numerical Lists

A list is a sequence of zero or more elements of a given type. Since the nodes and elements in a FE model may be referenced uniquely by a number, the XBUILD grammar allows designers to construct numerical lists for specifying "which" or "where" an action is to be applied to an object.

The XBUILD grammar should mimic that of the English language as close as possible (see section 4.5). However, in practical engineering design complicated numerical lists of regularly and irregularly spaced items often need to be constructed. Consequently, XBUILD

allows lists to be created in several ways, some with token names that closely mimic the English grammar, others with compact forms using braces, parenthesis and semi-colons. The latter methods require less key-strokes, and are preferred by more advanced users.

```
  <value> ::= <expr>                                    (rule 7)
            | <quantity>                                (rule 8)
            ;

<numlist> ::= <value>                                   (rule 9)
            | [ <numlist> , <value> ]                   (rule 10)
            | <value> to <value>                        (rule 11)
            | [ <value> : <value> ]                     (rule 12)
            | <value> to <value> by <value>             (rule 13)
            | [ ( <value> ~ <value> ) : <value> ]       (rule 14)
            | [ <numlist> ';' <numlist> ]               (rule 15)
            ;
```

Figure 4.6. Numerical list rules.

Figure 4.6 summarises the XBUILD grammar for constructing numerical lists. The simplest numerical lists are single values (shown in rule 7) and engineering quantities (rule 8); i.e. values accompanied by a unit of measure. Rules 9 through 15 define the grammar for constructing numerical lists containing multiple elements. For example, typing **[4,5,6,8]** matches rule 10 and generates the numerical list 4,5,6,8. An important draw back of rule 10 is the burden placed on the user when a long list of regularly spaced numbers needs to be specified. To rectify this shortcoming, rules 11 and 12 allow a list to be generated by specifying its starting and ending values (i.e., range), with 1 being the implied separation increment. Typing either **4 to 8** or **[4:8]**, for example, specifies the numerical list 4,5,6,7,8. Rules 13 and 14 are similar to rules 12 and 13, except that separation increments may be explicitly specified. For example, typing either **0 to 6 by 2** or **[(0 2):6]**, generates the list 0,2,4,6. Also, notice that numerical lists may be concatenated to each other. For example, typing **[0 to 6 by 2; 6 to 12 by 3]** generates the single list 0,2,4,6,9,12. The use of concatenated lists is illustrated in chapter 5.

4.3.2 XBUILD's Program Modes and Prompting Mechanism

When a designer is working through the details of setting up a bridge design problem description, sequential commands tend to have similar "actions" and "objects", with only the details of the "which" or "where" information changing from command to command. Rather than requiring the user to type the complete details of his intent with each command, XBUILD supports the idea of program "modes". Program modes store the current <verb> <noun>

combination of interest. This allows users to convey the precise intent of a command by only typing the scope information, with the <verb> <noun> settings being inferred from the current program mode.

Program modes may be entered by either typing the required program state at the keyboard, or, selecting a menu item from the bar located along the top of the XBUILD window frame. To enter the "add node" program state with the keyboard, for example, the user simply types "add node". XBUILD provides visual feedback of the current program state by appending components of the <verb> <noun> combination to the XBUILD prompt located in the TTY window; i.e., the "add node" program mode is indicated with the prompt XBUILD_add_node >>.

XBUILD stores the details of the current program mode as a linked list of <verb> <noun> components. Program mode items are appended and deleted from the end of the linked list — similar to pushing and popping items from a stack data structure — as program modes are entered and exited, respectively.

After the scope of a command is specified, XBUILD calls a "post_event_manager()" subroutine to determine the correct function call for the current program state. The early versions of XBUILD naively handled this decision process by explicitly setting program flags and testing every program mode combination. Now the present version simplifies and significantly shortens this process by noting that the function call and prompt appendage can have identical names. When XBUILD is started up, character strings for each <verb>_<noun> combination are stored in the hash table along with a pointer to a function having an identical name. Consequently, the new "post_event_manager()" (1) builds a character string of the prompt appendage, for example, the appendage extracted from the XBUILD_add_node_>> program state is "add_node"; (2) looks up the hash table using the character string as an argument; (3) verifies that contents of the returned hash table item is of type *pointer to function*; and finally (4) calls the function pointed to in the hash table item. This sequence of tasks is worked through every time the post_event_manager() is called, whether it is initiated from a keyboard command or from a combined graphics/mouse event.

4.3.3 Description of Geometries

The geometry of a finite element model is created by first specifying the geometric location of nodes and offset nodes, followed by the attachment of elements to these nodes.

Nodes

Nodes are created by keyboard input because it allows their coordinates to be specified to the machine precision of the computer. The procedure for adding nodes is to first enter the "add node" mode and then specify "where" the nodes are to be placed using commands of the form:

"x <numlist>   y <numlist>   z <numlist>"

where <numlist> is a numerical list, and x, y, and z are identifiers for numerical lists containing the nodal coordinates along the x, y and z axes, respectively. In its most general form, this grammar generates a three-dimensional rectangular block of nodal coordinates. However, when one or more dimensions take a single numerical value, nodes may be added one at a time, in a line, or in a rectangular mesh. For example, the command:

```
XBUILD_add_node >>   x 0 to 180 by 60   y 0 to 100 by 50   z 30
```

generates a rectangular grid of 4 by 3 = 12 nodes spaced 60 ft apart in the x-direction, spaced 50 ft apart in the y-direction, and located in a plane normal to the z axis 30 ft away from the x-y plane (see figure 4.7). Units of length were not specified in this example. Instead, the entered values were assumed to be in units of feet. XBUILD's assumptions of this nature are discussed in section 4.3.4.

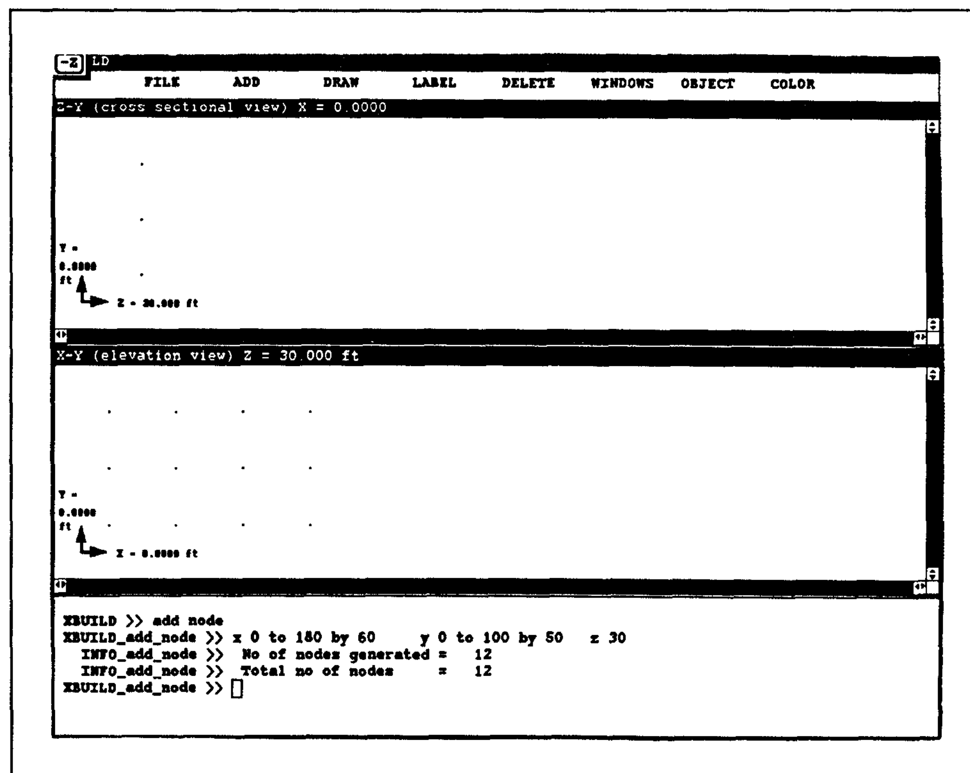


Figure 4.7. Node mesh.

The following mesh generation command illustrates XBUILD's ability to interpret arithmetic expressions and units within a command stream. Suppose that the command:

```
XBUILD_add_node >>    x 0 to 150*2 by 75    y (150*(2+.54)*12) cm    z 6 in
```

is entered while the design attribute for the quantity "length" is "feet", XBUILD generates a line of five nodes spaced 75 ft apart in the x direction and having the coordinates y = 150 ft and z = 6 ft (see figure 4.8).

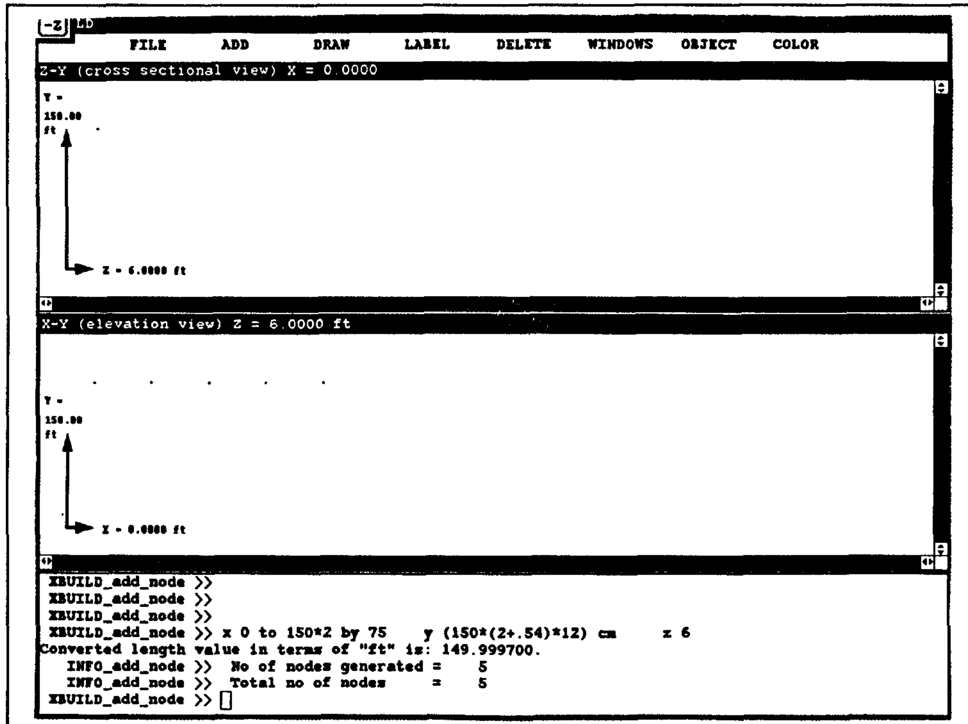
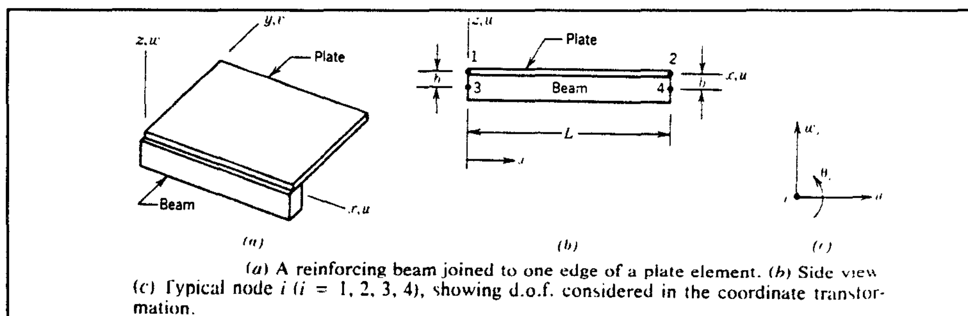


Figure 4.8. Node mesh with units and expressions.

Offset Nodes

XBUILD provides special nodes with constrained degrees of freedom, called "offset nodes". In the example shown in figure 4.9, the beam nodes 3 and 4 are connected to the plate



(*a*) A reinforcing beam joined to one edge of a plate element. (*b*) Side view (*c*) Typical node *i* (i = 1, 2, 3, 4), showing d.o.f. considered in the coordinate transformation.

Figure 4.9. Illustration of rigid links.

nodes 1 and 2 with rigid links. The degrees of freedom (DOFs) of the beam nodes then become linear functions of the DOFs of the corresponding plate nodes. Only the latter appear in the assembled stiffness matrix [6].

Offset nodes are needed for modeling composite highway bridge girders. They are created with the same commands as those for generating a regular mesh of nodes, while XBUILD is in the "add offset node" mode.

The DOFs of the offset nodes are "slaves" of the DOFs of the regular nodes. Offset nodes are linked to regular nodes with commands of the form:

```
XBUILD >>    link offset_node [3,4,5] to node [45,46,47]
```

Offset nodes are displayed on the screen with blue colored symbols larger than those used for regular nodes.

Elements

The current version of XBUILD creates two-node beam/column elements and four-node quadrilateral plate elements. Beam elements are modeled with three translational and three rotational degrees of freedom per node. Each node of a four-node plate element is modeled with three translational and two rotational degrees of freedom.

Beam/column elements may be added either one at a time using "from" and "to" grammar, or in series by specifying node connectivity with numerical lists. For example, as figure 4.10 shows, a single beam/column element may be added to a structure with the following command:

```
XBUILD_add_bc_elmt >>    from node 6 to node 11
```

Plate elements are also generated using numerical lists. In fact, planes of plates are specified with the same grammar and syntax as planes of nodes (figure 4.7). For example, the command:

```
XBUILD_add_plate4 >>    x 0 to 112 by 2    z 0 to 20 by 2    y 3
```

generates a grid of 56 by 10 = 560 plate elements, 2 units long in the x direction, 2 units wide in the z directions, all lying in the y = 3 plane (figure 4.11). As described in chapter 5, the concrete deck of a highway bridge can be easily modeled with plate elements in this manner.

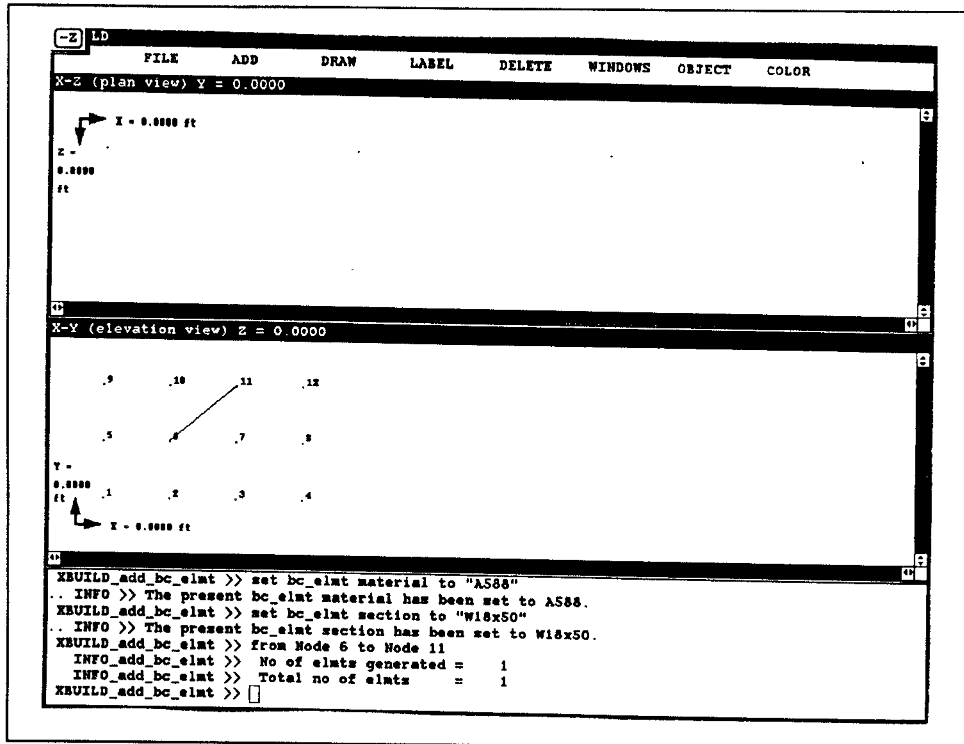


Figure 4.10. Addition of bc_elmts via language.

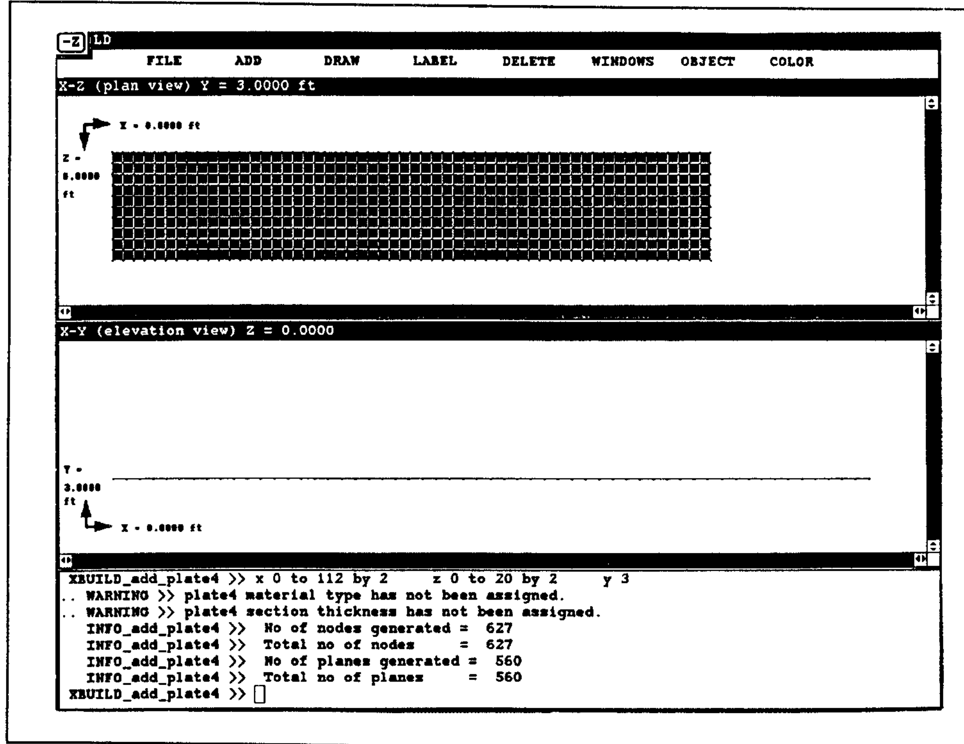


Figure 4.11. Addition of plates via language.

4.3.4 Design Attributes

Design attributes, set by the user, reduce the number of key strokes required to enter commands. For example, a pre-selected unit is assigned to a number entered without a unit. Additionally, pre-set material and section types are assigned to elements such as beams or plates. To illustrate how the user sets design attributes, consider the following command:

```
XBUILD >>    set bc_elmt material to "A36_STEEL"
```

This command assigns the material "A36_STEEL" to all beam/column elements. Similarly, the command:

```
XBUILD >>    set length units to feet
```

sets the design attribute for the quantity "length" to "feet". The User's Guide lists all of XBUILD's design attributes and explains how each is set.

4.4 Mouse

4.4.1 Introduction

In addition to a command language, XBUILD supports interaction by a mouse for changing program modes, manipulating graphical displays, and acting as a pointing devise to identify nodes and elements.

4.4.2 Three-Dimensional Graphical Display

A major obstacle in developing XBUILD was to devise a way of viewing both exterior and interior components of FE models. Although three-dimensional (3-D) views are useful for checking the overall FE model, they often cannot clearly show internal details. For example, the

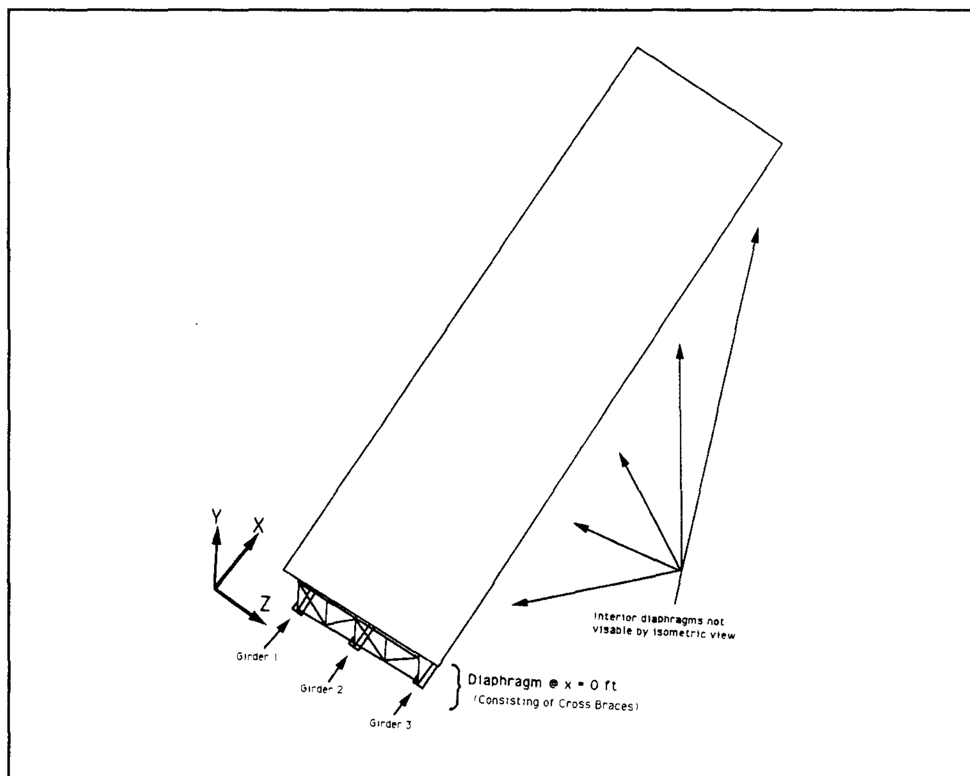


Figure 4.12. Isometric projection of FHWA test bridge.

isometric projection of the FHWA Test Bridge shown in figure 4.12 cannot provide an isolated viewing of the bridge's interior diaphragms. At best, this projection could be inverted, and the interior diaphragms viewed at an angle.

XBUILD mitigates this potential problem by providing three two-dimensional graphic displays of the plan, elevation, and cross-section views. Each view is obtained by cutting the model with a clipping plane, thereby allowing the user to view the contents of the model located in that plane [9]. The current version of XBUILD restricts clipping to planes perpendicular to one of the three principal axis (x,y,or z). Consequently, clipping views are always either x-y elevation views of clips along the z axis, x-z plan views of clips along the y axis, or z-y section views of clips along the x axis (see figures 4.14 through 4.17).

4.4.3 Graphical Display Manipulation

XBUILD's 2-D views can be changed with the mouse by either moving coordinate arrows or pressing buttons on the movement pad.

Coordinate Arrows

Each of XBUILD's three canvas subwindows (x-y, x-z, and z-y) contain two "coordinate arrows" that are located at the subwindow's origin and are parallel to its two principal axis. Each arrow can be extended or contracted in a direction parallel to its initial orientation. For example, in the X-Y subwindow the X and Y arrows can be moved in the X and Y directions respectively. Coordinate arrows are moved by a press, drag, and release sequence using either the right or middle buttons of the mouse. The middle button controls all vertical arrows while the right button controls the horizontal arrows.

For example, in figure 4.18 the X arrow in the X-Y window (elevation view) is positioned at X=10 ft, and the Z-Y window (cross section view) displays the diaphragm located on the X=10 ft clipping plane. Initial depression of the right button (i.e., the SunView event identified as "RT_MOUSE_BUTTON = DOWN") snaps the cursor to the tip of the right arrow. A drag of the mouse along the mouse pad (i.e., RT_MOUSE_BUTTON = LOC_DRAG) moves the right arrow with the cursor. Finally, when the right button is released at X=50 ft (i.e., RT_MOUSE_BUTTON = UP), the Z-Y window is cleared and the diaphragm located on the X=50 ft clipping plane is displayed. The graphical update resulting from this sequence of mouse events is shown in figure 4.19.

Similarly, in figure 4.20, the Z arrow in the Z-Y window is positioned at Z=9.5987 ft, and the X-Y window displays the interior girder located on the Z=9.5987 ft clipping plane. If the Z arrow is moved to Z=16.401 ft, the X-Y window is updated and displays the exterior girder located on the Z=16.401 ft clipping plane (figure 4.21).

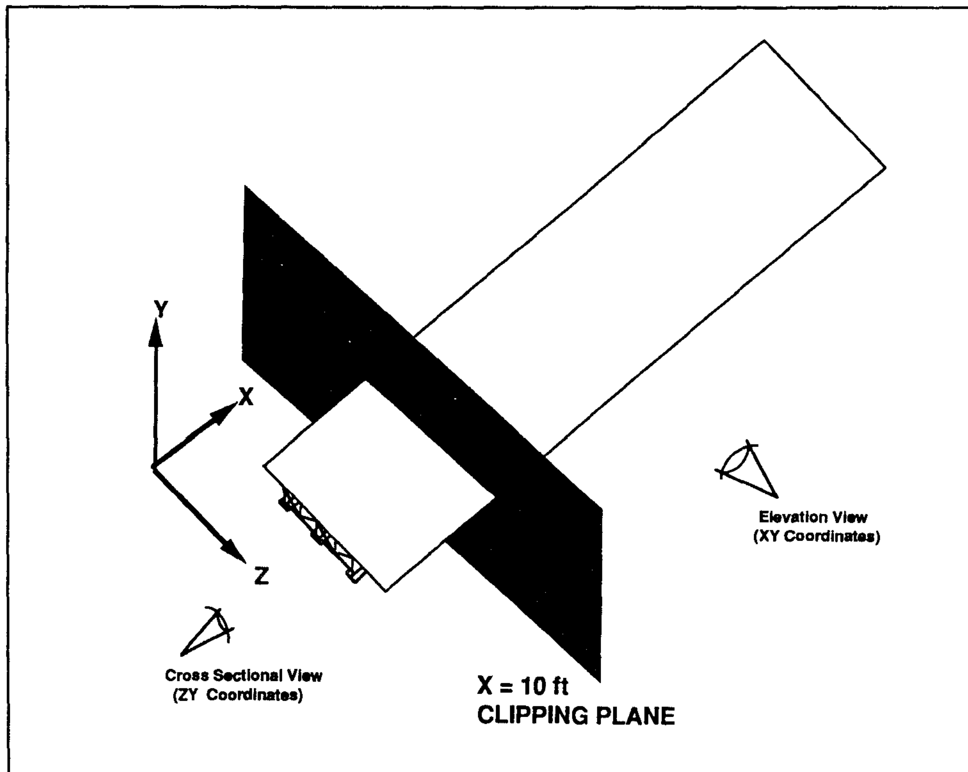


Figure 4.14. x = 10.0 ft clipping plane.

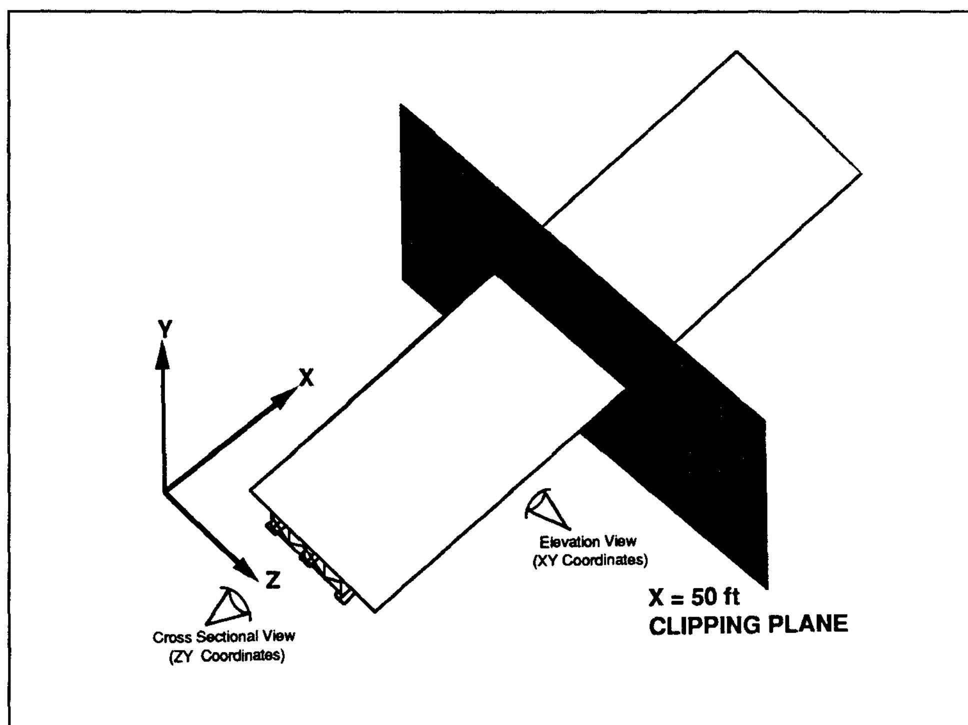


Figure 4.15. x = 50.0 ft clipping plane.

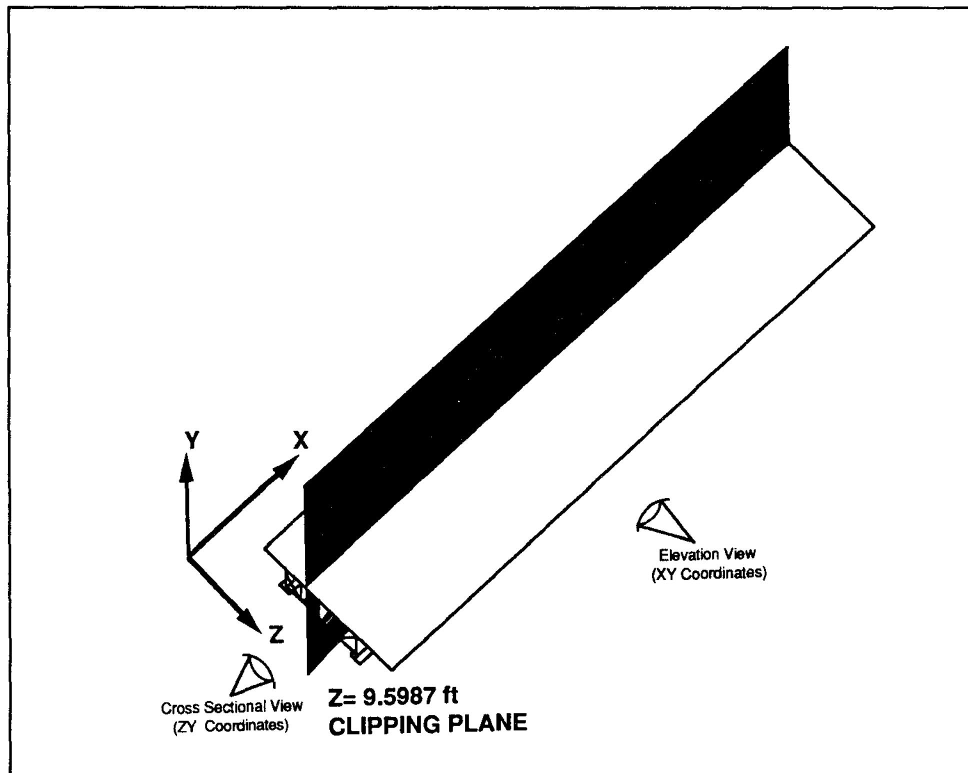


Figure 4.16. z = 9.5987 ft clipping plane.

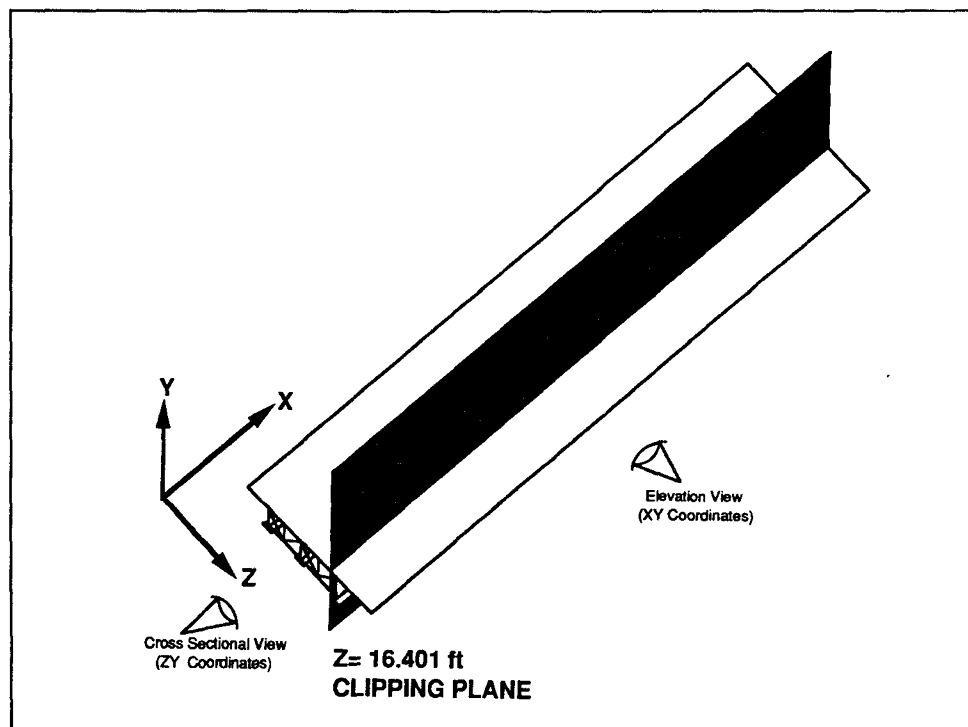


Figure 4.17. z = 16.401 ft clipping plane.

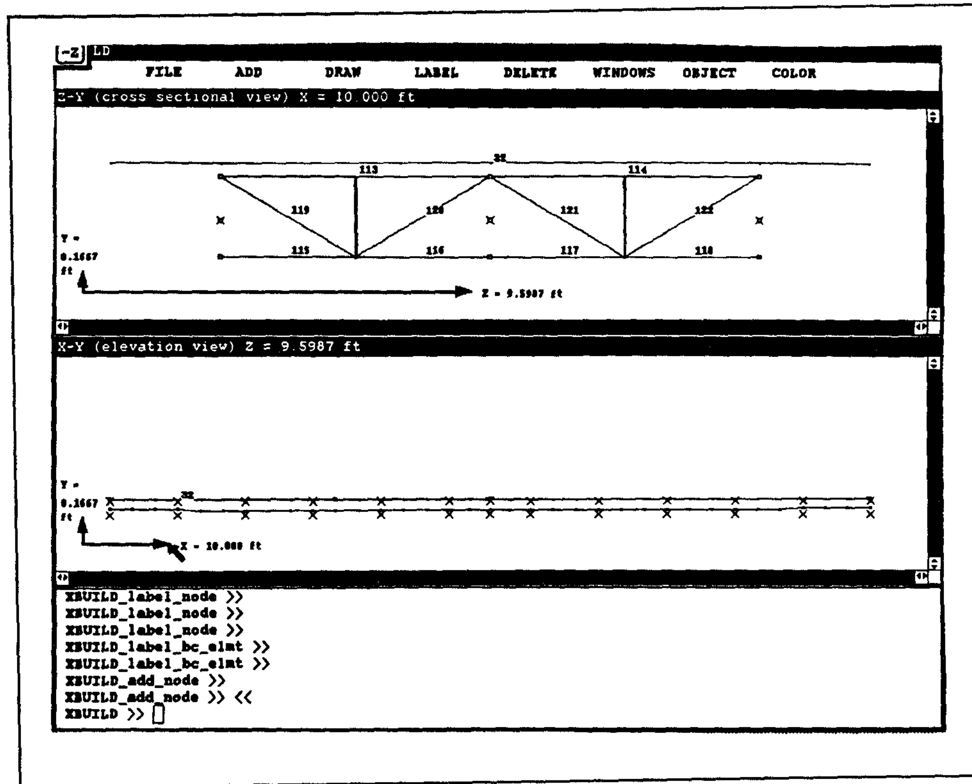


Figure 4.18. XBUILD's x = 10 ft clipping plane.

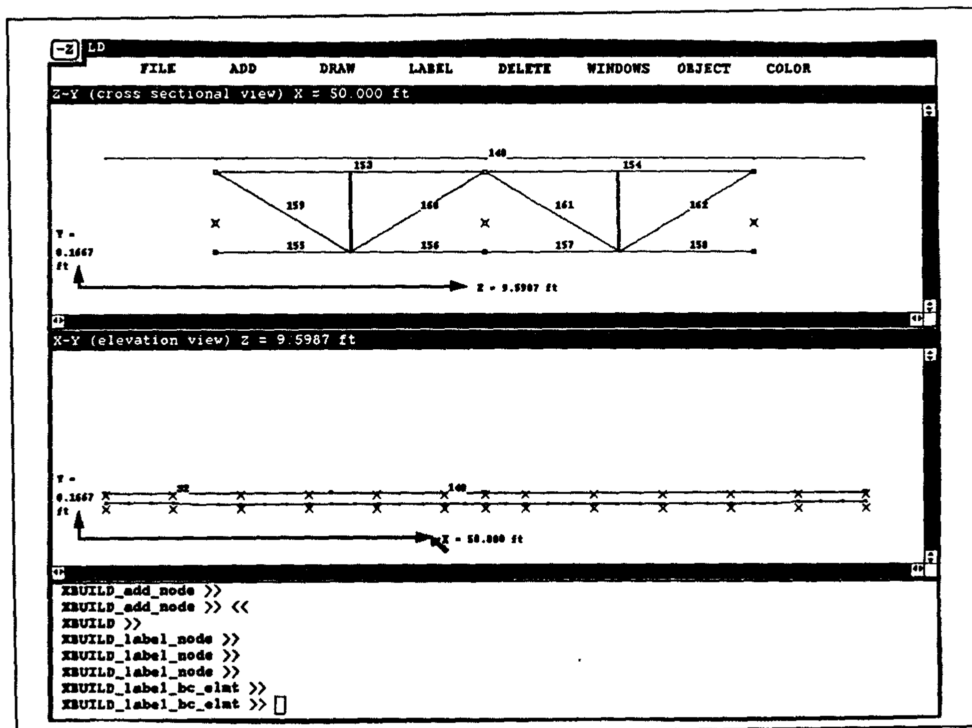


Figure 4.19. XBUILD's x = 50 ft clipping plane.

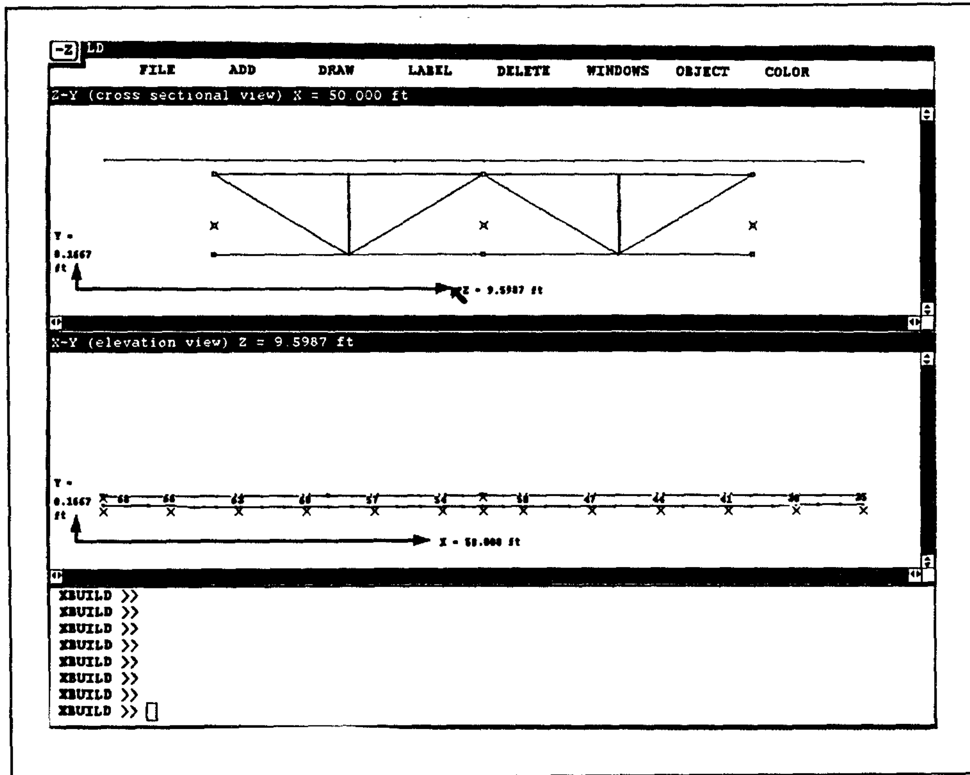


Figure 4.20. XBUILD's z = 9.5987 ft clipping plane.

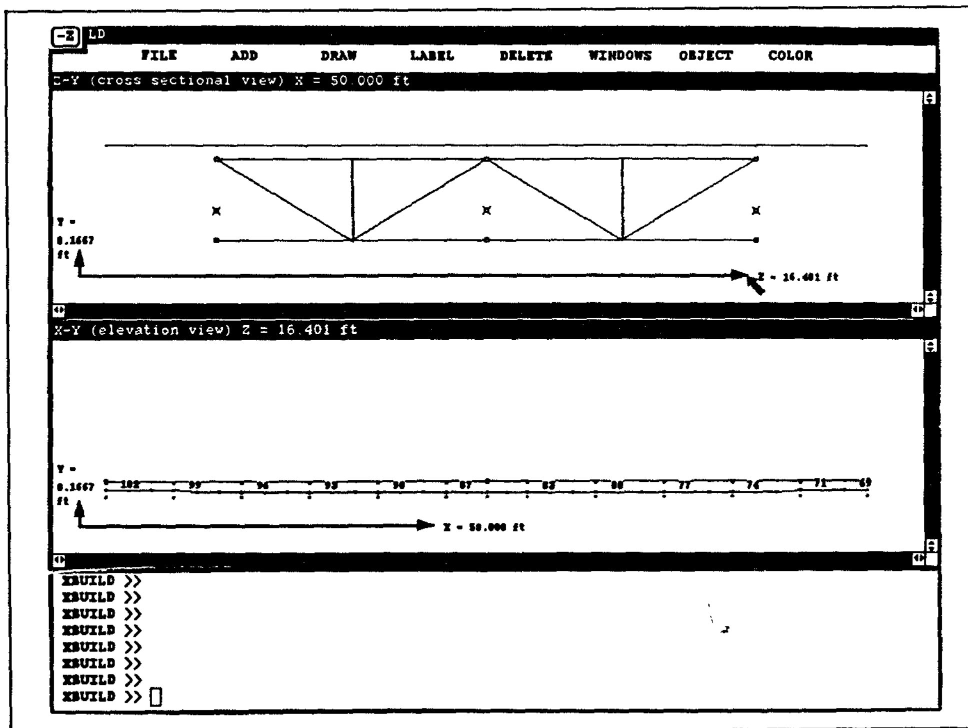


Figure 4.21. XBUILD's z = 16.401 ft clipping plane.

Movement Pad

XBUILD's movement pad, located in the upper left corner of the display screen, provides a second mechanism for changing clipping planes. The buttons on the movement pad, labeled [+X], [-X], [+Y], [-Y], [+Z], and [-Z], represent incremental movements of clipping planes in positive and negative directions along an axis. When a movement button is pressed, views are updated corresponding to the clipping plane displayed on the button. For example, if the +x button is pressed, the z-y subwindow displays the next clipping plane in the positive x direction. Similarly, if the -x button is pressed, the z-y subwindow displays the next clipping plane in the negative x direction.

Consequently, the movement pad is especially useful for viewing clipping planes in large problems where nodes and planes are densely packed on the screen, since this relieves a designer from needing a high level of eye/hand coordination.

4.4.4 Grabbing Objects

When the left button is pressed and the mouse is moved, a rectangle appears on the screen that can be used to enclose nodes and elements. Releasing the button "grabs" the enclosed items and temporarily stores their numbers in lists (i.e., numeric lists are created). Thus, grabbing items with the mouse is a method, alternate to the language, for answering the question of "which" items are to be the scope of mode specific action. For example, grabbing four nodes while the "add plate4" mode is active adds a quadrilateral plate element. This process is shown in figures 4.22 through 4.25. The User's Guide lists the available modes.

4.4.5 Pull-Down Menu

Pull-down menus speed the execution of frequently called procedures such as changing program modes (i.e., specifying a <verb> <noun> combination), saving a model, clearing the screen, changing element colors, and exiting the program.

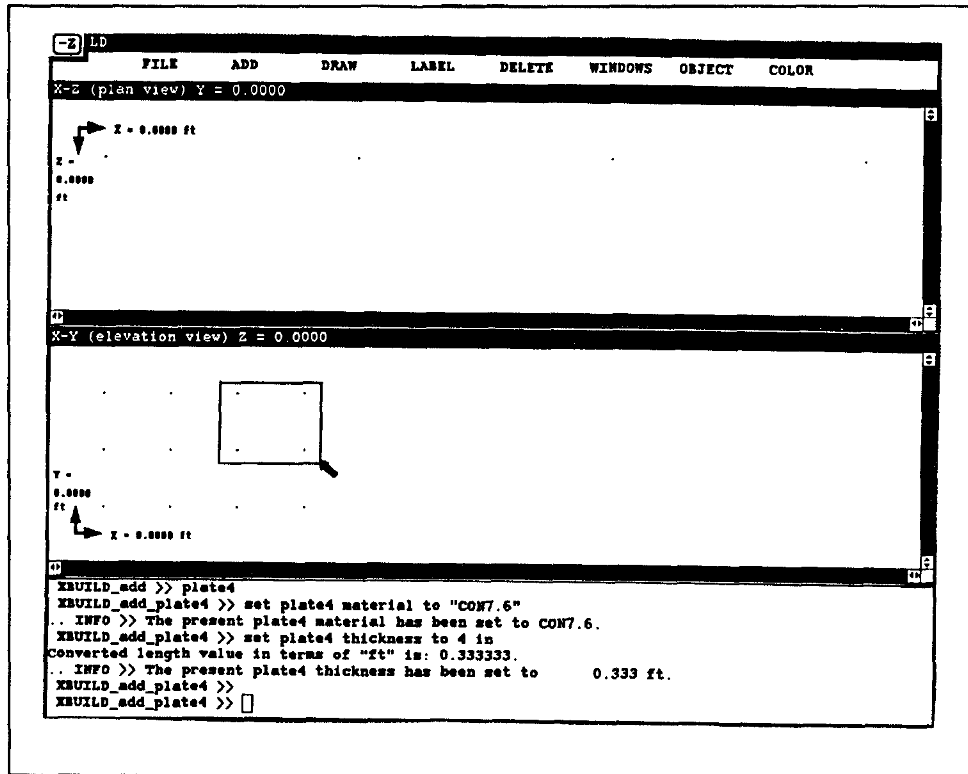


Figure 4.22. Nodes are grabbed.

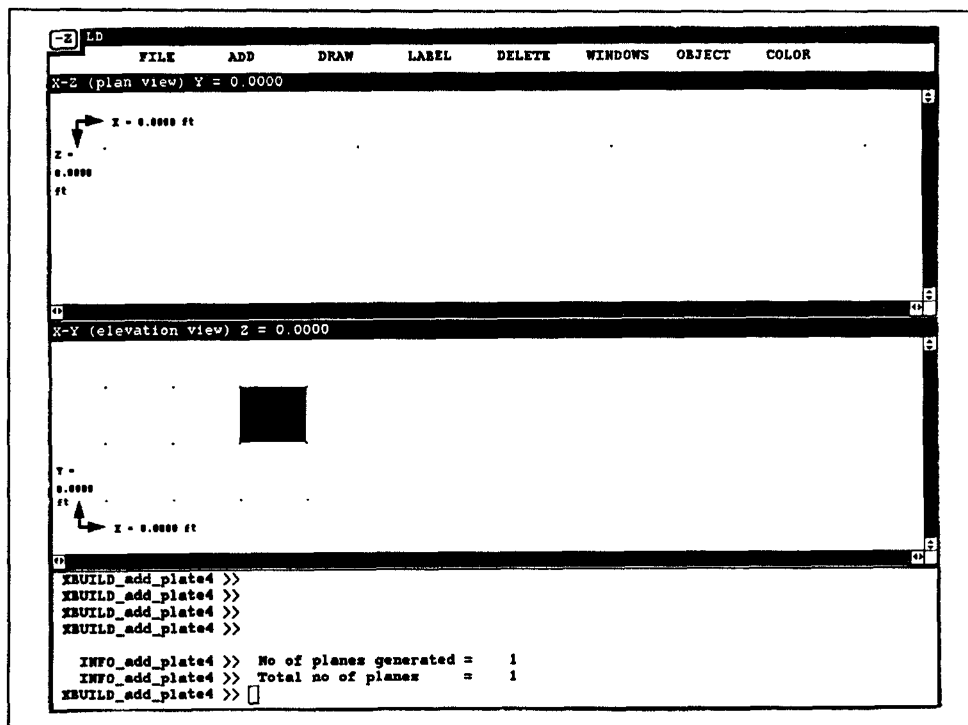


Figure 4.23. Four-node plate element is added.

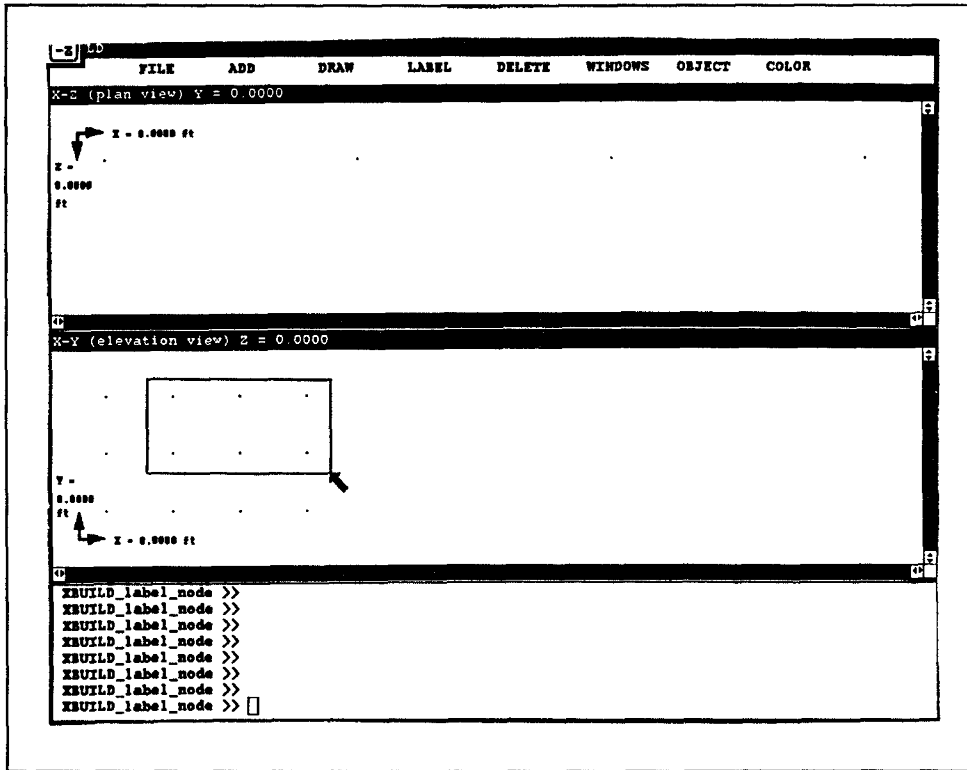


Figure 4.24. Nodes are grabbed.

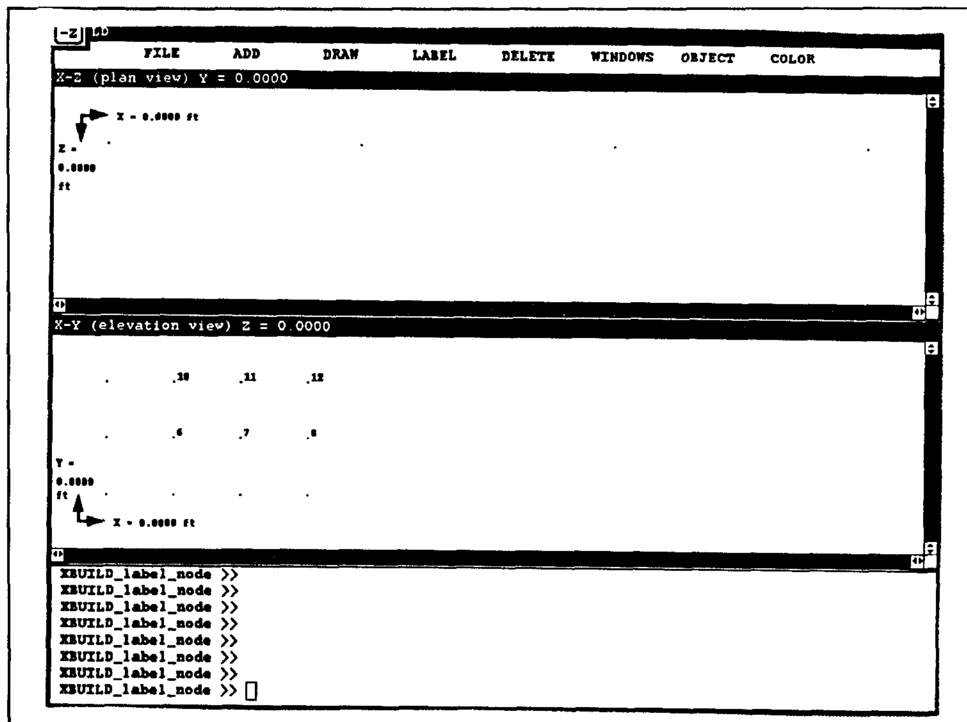


Figure 4.25. Nodes are labeled.

4.5 Query, Modification, and Grouping of Data

Query, modification, and grouping of design data such as material properties, cross section information, element parameters, and load information is supported by both the command language and mouse.

4.5.1 Data Query

Data queries are accomplished by specifying numerical lists containing the numbers of the nodes, elements, loads, or boundary conditions in question while in an appropriate "get data" mode. For example, the design data for beam/column elements numbered 35 through 40 may be obtained by either the command:

```
XBUILD_get_data_bc_elmt >>    [35:40]
```

or by grabbing those elements with the mouse in the "get_data bc_elmt" mode.

4.5.2 Data Modification

The procedure for modifying design data is identical to that for making design queries except a "change <noun>" mode must be active. For example, to change the material types of the beam/column elements in section 4.5.1 to the pre-set "attribute" material, the elements must either be specified by number with the keyboard or grabbed with the mouse while the "change bc_elmt" mode is active.

4.5.3 Creation of Structural Objects (Data Grouping)

A structural object is a collection of items such as nodes, elements, loads, and boundary conditions that can be given a name, duplicated, queried, and modified. With the duplication feature, copies of an object can be positioned throughout a model, thus reducing the amount of work. For example, all of the diaphragms in the FE model of the FHWA Test Bridge discussed in chapter 5 were generated with the structural object facility. Rather than add the nodes, offset_nodes, and elements for each of the bridge's eleven diaphragms, one diaphragm was added, and an object named "dia" was created. Duplicate copies were then placed at the remaining ten positions along the bridge length.

4.6 Data Structures

4.6.1 Introduction

The management of data for finite element analysis and design is a problem complicated by both the wide variety of design information and the changes in data as a design evolves from the preliminary to the final version. Since the volume of data can be large, even for problems of moderate size, procedures for storing, retrieving, and updating data must be carefully planned. XBUILD employs three forms of data structures: linked lists, arrays, and hash tables. The following sections describe each data structure and discuss their use in XBUILD.

4.6.2 Linked Lists

A linked list is a sequence of items of any size. Each item contains a pointer to the next item in the sequence. Unlike the items in an array, the items in a linked list do not have to reside in any particular order, such as ascending numerically, nor do they have to be physically next to each other in memory. Take, for example, a linked list consisting the four items A, B, C, and D (figure 4.26). Item A might point to item C which, in turn, might point to item B, which might point to item D. Thus, starting with item A (the head of the list), each item can be processed in the order determined by following the pointers [32]. A threaded linked list, by definition, is a linked list where each item contains, as one of its members, the head of another list [32].

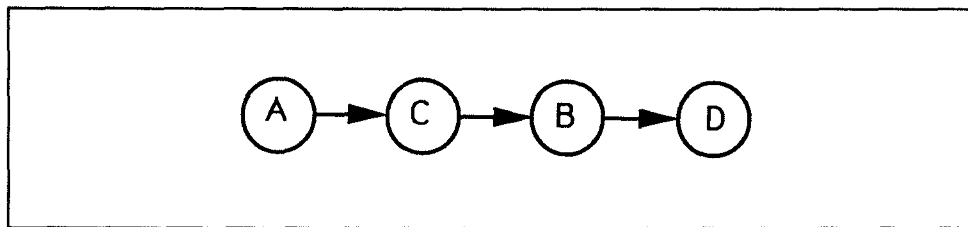


Figure 4.26. Schematic of one-way linked list.

Applications of Linked Lists

XBUILD's linked lists are dynamic, meaning they grow and shrink as need requires. Thus, only the amount of memory required by the situation at hand, is allocated during program execution. The ability to rapidly insert and delete nodes and elements is important in FE

pre-processing. XBUILD accomplishes this through manipulations of simple one-way linked lists containing double precision numbers. The type definition for one such list is shown in figure 4.27.

```
typedef struct arglist {
        double number;
        struct arglist *next;
    } ARG_LIST,    *ARG_LIST_PTR;
```

Figure 4.27. Linked list type definition.

Data Storage via Threaded Linked Lists

One potential problem with XBUILD is that graphics windows are slowly updated in large problems, especially if all nodes and elements must be searched and tested to see if they should be plotted. To help overcome this problem, XBUILD uses threaded two-way linked lists to store nodes and elements by their coordinate values.

To see how this idea works, consider the space frame shown in figure 4.28. As nodes and elements are created, links are added to a system of lists according to their x, y, and z values. Nodes and elements having common coordinates are stored together and remain in close proximity.

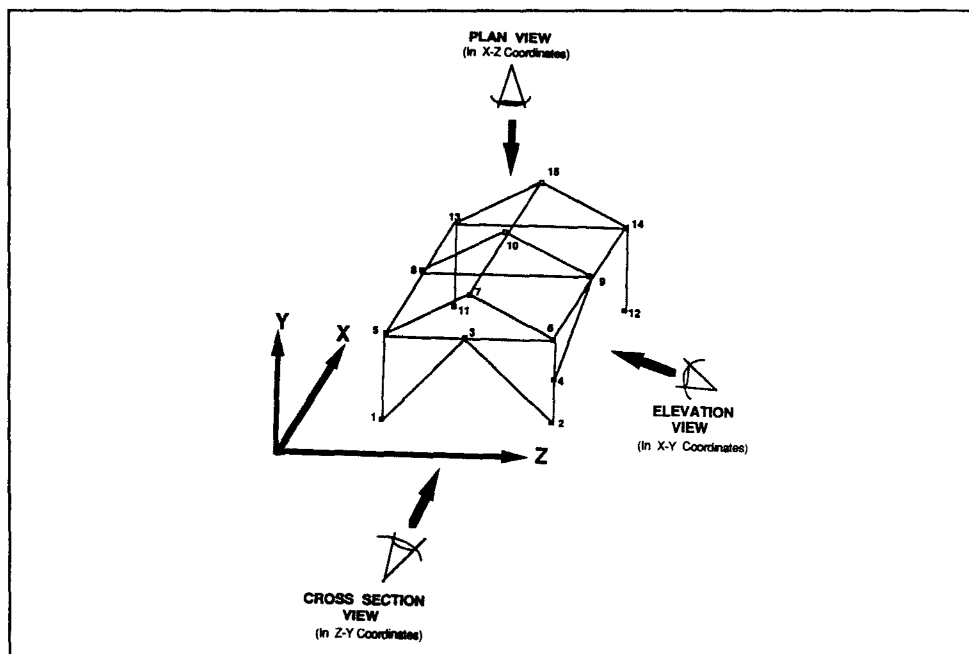


Figure 4.28. Space frame.

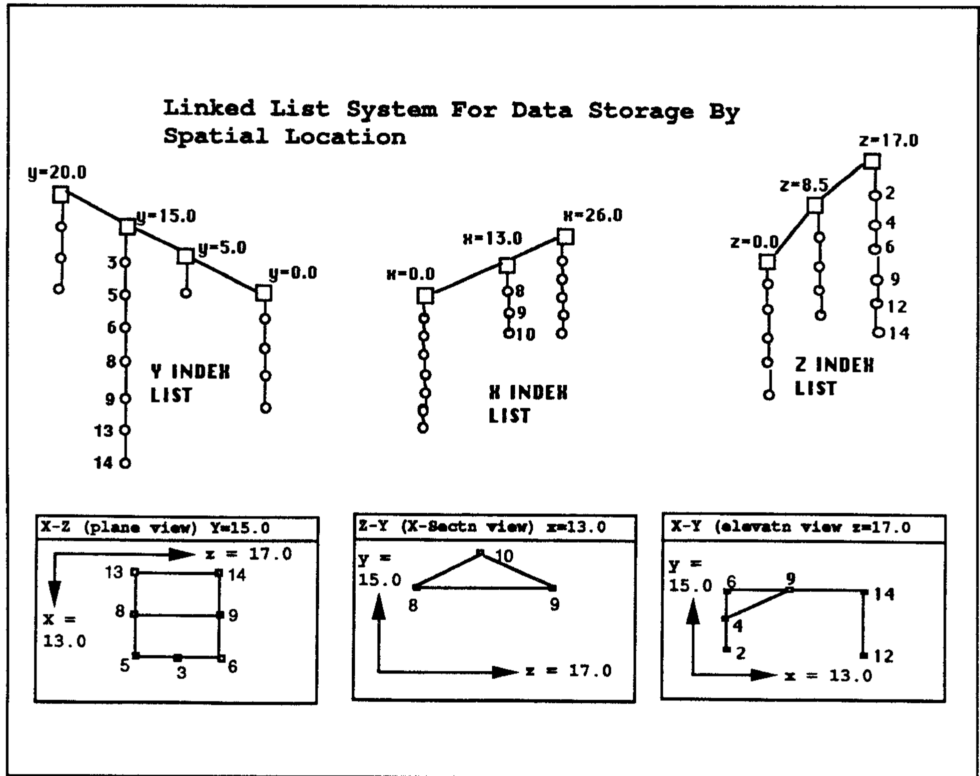


Figure 4.29. Index lists: x, y, and z.

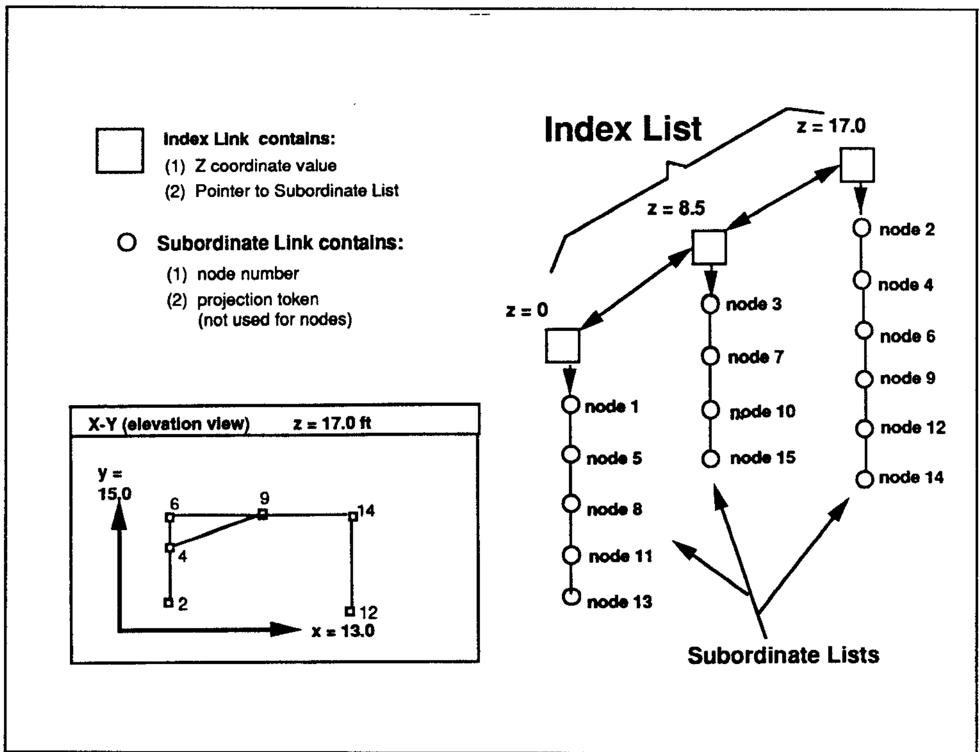


Figure 4.30. z node index list.

XBUILD's linked list system depends on three primary sets of index lists for organizing its nodes and elements. These primary lists are the x, y, and z index lists (figure 4.29 and table 4.3). Each item in an index list represents a particular x, y, or z clipping plane. Moreover, each item in an index list contains a pointer to a subordinate list which in turn stores the numbers of the nodes or elements that lie on a given clipping plane.

Index Lists		
x_node_index_list	y_node_index_list	z_node_index_list
x_bc_elmt_index_list	y_bc_elmt_index_list	z_bc_elmt_index_list
x_plate4_index_list	y_plate4_index_list	z_plate4_index_list

Table 4.3. XBUILD's index lists.

Each item in the z node index list shown in figure 4.30 (e.g., items z =0.0, 8.5, and 17.0) contains the head of a subordinate Z list. For example, the Z=17.0 index head points to a list of all of the nodes located in the clipping plane z = 17.0.

Figure 4.30 illustrates how the index-list system enhances data access in support of graphics. When XBUILD's drawing routines are called to display this elevation view, the following occurs: (1) the nodes located on the Z=17.0 plane are accessed from the z_node_index_list and displayed on the screen; and (2) the beam/column elements located on the Z=17.0 plane are accessed from the z_bc_elmt_index_list and displayed.

4.6.3 Arrays

All attributes of XBUILD's nodes and elements, such as material properties, cross-sectional information, and coordinate values, are stored in arrays. The retrieval of these attributes is immediate because — given a node, beam, or plate number as the index — the array member can be read directly.

Additionally, each array position contains pointers to corresponding locations in the index-list system. Through these pointers, information about the nodes or elements in the vicinity of the one designated by number may be sought. For example, all nodes or elements close to node number 9 may be accessed through pointers in array position 9 (figure 4.31).

Figure 4.32 shows the source code that defines the data structure used in an array containing node information (i.e., the array nodes[-]). Array positions reserved for nodes store the following: color; x, y, and z coordinates; and a pointer to the list of all elements attached to the node. To delete a node, all elements attached to it must be deleted first. Consequentially, routines for deleting elements update the array positions of the nodes attached to the elements.

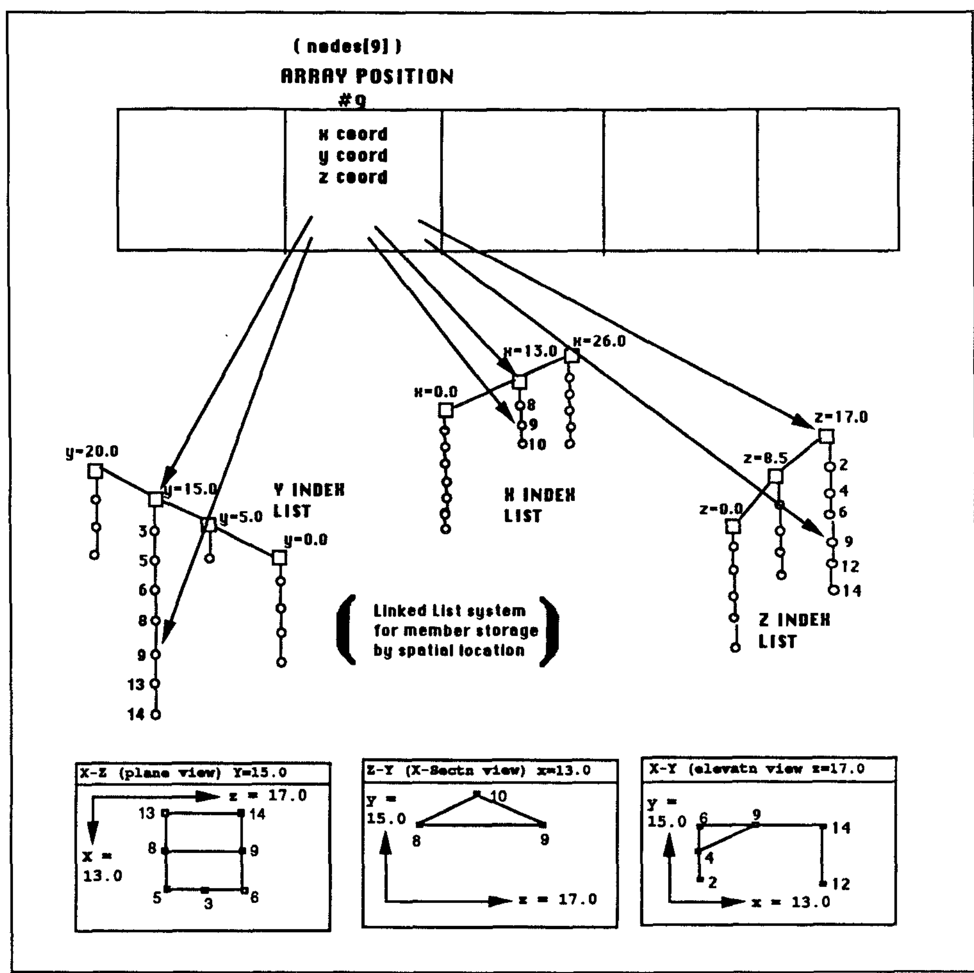


Figure 4.31. Array-list couple.

```
typedef struct node {
      int off_set_node_num;
      int color;
      double  x_coord;       /* <= x nodal */
      double  y_coord;
      double  z_coord;
      double  mass;          /* <= For Dynamic use */
        int  new_nodeno;
        int  deleted;
        CONN_LIST_PTR   attached_elmts_list;
    NODE_BC_LIST_PTR   bcond;   /* <= Point to bcond */
  NODE_LOAD_LIST_PTR   load;    /* <= and load lists */
      COORD_LIST_PTR   x_geom_link;
 COORD_LIST_INDEX_PTR   x_geom_head;
 COORD_LIST_PTR   y_geom_link; /* <= List Locations */
  COORD_LIST_INDEX_PTR   y_geom_head; /* <= List Head */
  COORD_LIST_PTR   z_geom_link; /* <= system. */
  COORD_LIST_INDEX_PTR   z_geom_head;
        } NODE_LIST, *NODE_LIST_PTR;
```

Figure 4.32. Type definition for node array.

4.6.4 Hash Table

A hash table is a data structure characterized by its directness and speed for accessing and storing information. Each items in a hash table is assumed to have a unique name, and therefore may be directly referenced by performing arithmetic or other operations on data provided as an access key [28].

Although a number of hashing schemes and data structures are available — see Sedgewick [28] for a survey — XBUILD uses an array of bucket table headers pointing to linked lists of table items. In fact, only one hash table is needed to store several types of information, including:

[1] Key words used in the YACC grammar

[2] AISC sections

[3] Material properties

[4] Lists of nodes, beam elements, and plate elements

[5] Pointers to functions that implement user commands

[6] Integers used to flag stages of processing

To access a table item, XBUILD hashes, or "looks up", a grammatical expression either entered by the user or sent by an XBUILD routine. XBUILD's hash access mechanism

consists of an array of pointers to linked lists. Figure 4.33 shows a schematic of this data arrangement. Items in the linked list are called "hash nodes". The type definition for hash nodes is shown in figure 4.34.

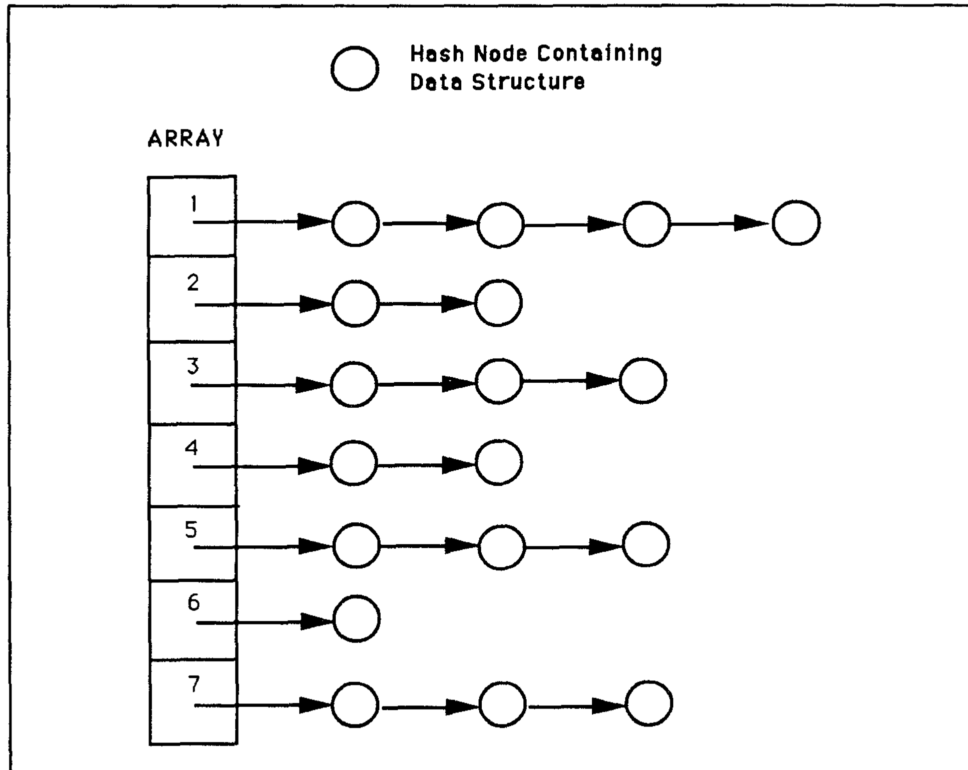


Figure 4.33. Schematic of hash table.

```
typedef struct hash_node {
           char   *name;
           short   name_token;
                  short   type;
        union {
                     double          val;
                     double    (*func)();
                     BC_SECTION_PTR   sp;
                     MATERIAL_PTR     mp;
                     ARG_LIST_PTR     ap;
              } u;
             struct  hash_node        *next;
} HASH_NODE, *HASH_NODE_PTR;
```

Figure 4.34. Type definition for hash node.

Each hash node contains a name describing the information stored in it, a representation of the name in token form, the type of information being stored, and a union. Unions allow one of several possible data types to occupy a single area of memory [20]. For example, in

figure 4.35, the hash node "node", stores a pointer to a linked list of type ARG_LIST_PTR (defined in figure 4.27). However, elsewhere, the hash node "W18x50" stores AISC section data.

Temporary Lists in Hash Table

When numerical lists are generated, as described in section 4.4, they are temporarily stored in the hash table. For example, when a list of node numbers is generated, it is stored in the hash table under the name "node" and may be retrieved at a later time by simply looking up (i.e., hashing on) the word "node". Figure 4.35, an excerpt from a routine that labels nodes, shows how XBUILD does this.

```
double label_node()

HASH_NODE_PTR hn;
ARG_LIST_PTR p;

hn = lookup("node"); /* <= Hashing Routine Called */


 for (p=hn->u.ap; p!=NULL; p=p->next) {

... Code for labeling nodes in a list ...
}

hn->u.ap = NULL;
return;
}
```

Figure 4.35. Routine that calls the hashing function.

4.6.5 Concluding Remarks

The benefits of having several types of data structures to store information becomes apparent when XBUILD is running on the SUN SPARC Station; graphical updates appear instantaneously.

CHAPTER FIVE

FHWA TEST BRIDGE EXAMPLE

This chapter explains the sequence of keyboard commands and mouse operations needed to create a FE model of the FHWA test bridge. The model consists of 341 nodes and 504 elements, and was devised by Robert Canham, a Graduate Research Fellow at the FHWA's Turner-Fairbanks Highway Research Center.

The User's Guide gives examples of the steps engineers must work through to setup models of simpler structures.

```
------------------------------------------------------------
Phase 1)  Generation of a quadrilateral plate element
          mesh that models the bridge deck.
------------------------------------------------------------

DECK INTRODUCTION:

The procedures in this phase generate a mesh of four-node
quadrilateral plate elements that model the bridge's deck.

In the X direction, the plates are 40 inches long from X=0
to X=600 inches, 36 inches long from X=600 inches to X=744
inches, and 40 inches long from X=744 to X=1344 inches.

In the Z direction, the plates are 33.56 inches wide from
Z=0 to Z=33.56 inches,  27.208 inches wide from Z=33.56 to
Z=196.808 inches, and 33.56 inches wide from Z=196.808 to
Z=230.368 inches.

The mesh of plates lies on the Y=29.707 inch plane.  All
plates are 4 inches thick and assumed to be made out of 7.6
ksi concrete.

COMMANDS:

1) With the mouse, the color "concrete" is selected from the
pull-down menu labeled "colors".

2) XBUILD >> add plate4

XBUILD is put into its "add plate4" mode.

3) XBUILD_add_plate4 >> set plate4 thickness to 4 in

The plate thickness design attribute is set to four inches.

4) XBUILD_add_plate4 >> set plate4 material to "CON7.6"

The plate element material attribute is set to type "CON7.6"
The properties associated this material may be found in the
file "materials.dat".

5) XBUILD_add_plate4 >> set length units to in

The unit for length is set to inches, thus all values
entered when a length is required will be assumed of the
unit inch.

6) XBUILD_add_plate4 >>  x [0 to 600 by 40; 600 to 744 by
36; 744     to 1344 by 40]   z [0 to 33.56 by 33.56; 33.56
to 196.808 by 27.208;  196.808 to 230.368 by 33.56]  y
29.707

A grid of quadrilateral plate elements is generated, having
the X and Z spacings defined above (in the deck
introduction) and the design attributes specified in
commands 3 and 4 (see figure 5.1).

7) XBUILD_add_plate4 >> save "deck"

The current structure (ie. the deck) is saved in a binary
file called "deck.object".
```

Example: FHWA Test Bridge

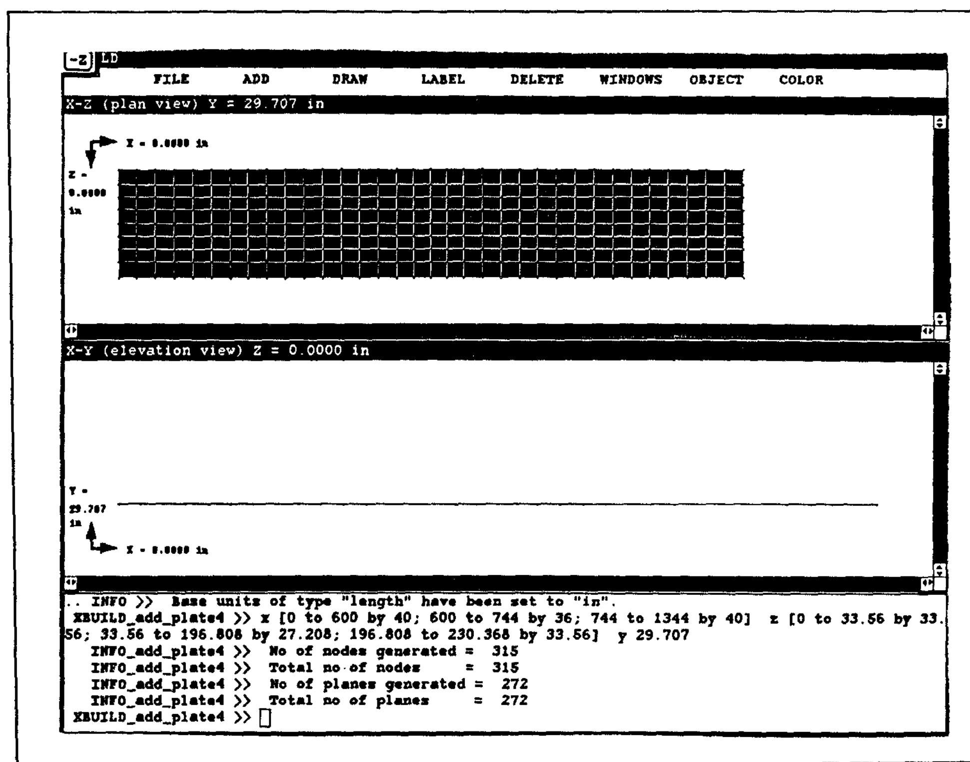


Figure 5.1. Plate element mesh for modeling bridge deck.

Phase 2: Generation of Bridge Girders using Offset Nodes

GIRDER INTRODUCTION:

The bridge's three girders are modeled as two-node beam elements. To connect the beam elements of the girders to the plate elements of the deck, slave-master constraints are imposed with XBUILD's offset node feature. Refer to Section 4.3.1 for more on offset nodes.

3) XBUILD_add_offset_node >> x [0 to 600 by 40; 600 to 744 by 36; 744 to 1344 by 40] z 33.56 to 196.808 by (27.208*3) y 12.51

Offset nodes for modeling the girders are generated.

NOTE: This command assigns a Y value of 12.51 inches to all nodes, the center nodes will be lowered to Y=10.79 inches at a later time (see figure 5.2).

5) XBUILD_add_nodal_off >> <

6) XBUILD_add >> bc_elmt

7) With the mouse, the color "weath-steel" is selected.

8) XBUILD_add_bc_elmt >> set bc_elmt section to "FHWA_GC"

The beam/column element cross-section type attribute has been set to "FHWA_GC". The variables associated this section may be found in the file "sections.dat".
5) XBUILD_add_bc_elmt >> set bc_elmt material to "FHWA_MAT".

The beam/column element material attribute is set to type "FHWA_MAT".

6) Nodes are grabbed with the mouse, thus inserting beam/column elements between them (see figure 5.3).

Phase 3: Lowering of Center Nodes.

----

1) XBUILD_add_bc_elmt >> <<

2) XBUILD >> move node

3) XBUILD_move_node >> [(-10~-3):-94] to y 10.79

Offset nodes numbered: 10,13,16,..,94 are lowered to a Y coordinate of 10.79 inches. The node numbers specified in this command are negative because the nodes moved are actually offset nodes.

4) XBUILD_move_node >> save "moved"

The structure is saved as "moved.object".

Example: FHWA Test Bridge

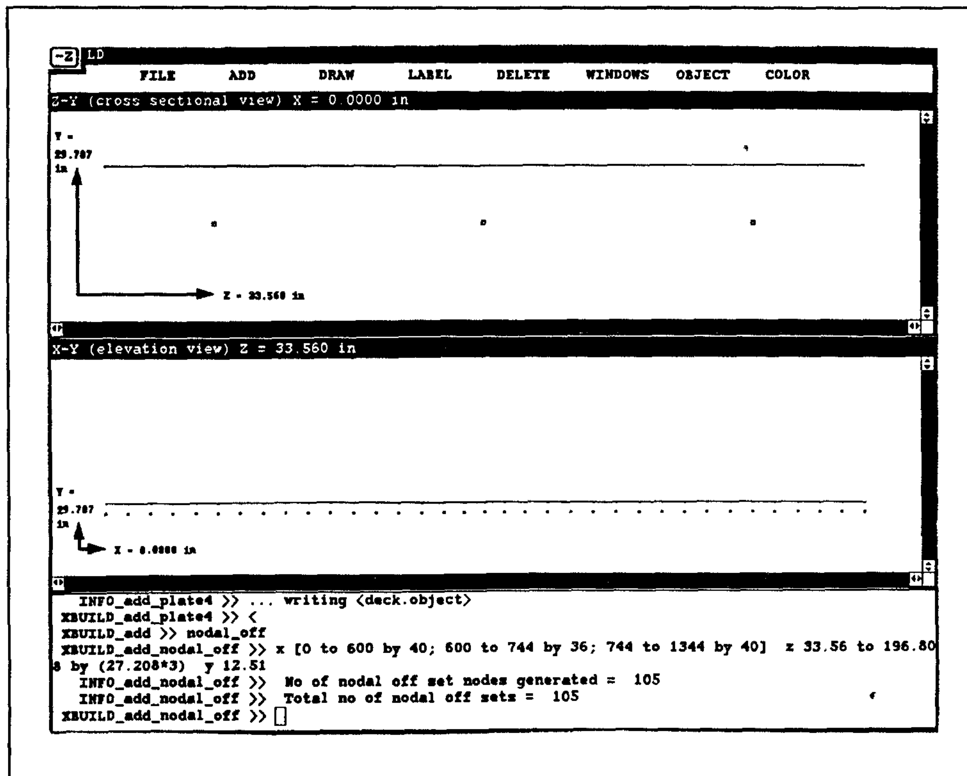


Figure 5.2. Generation of offset nodes for girders.

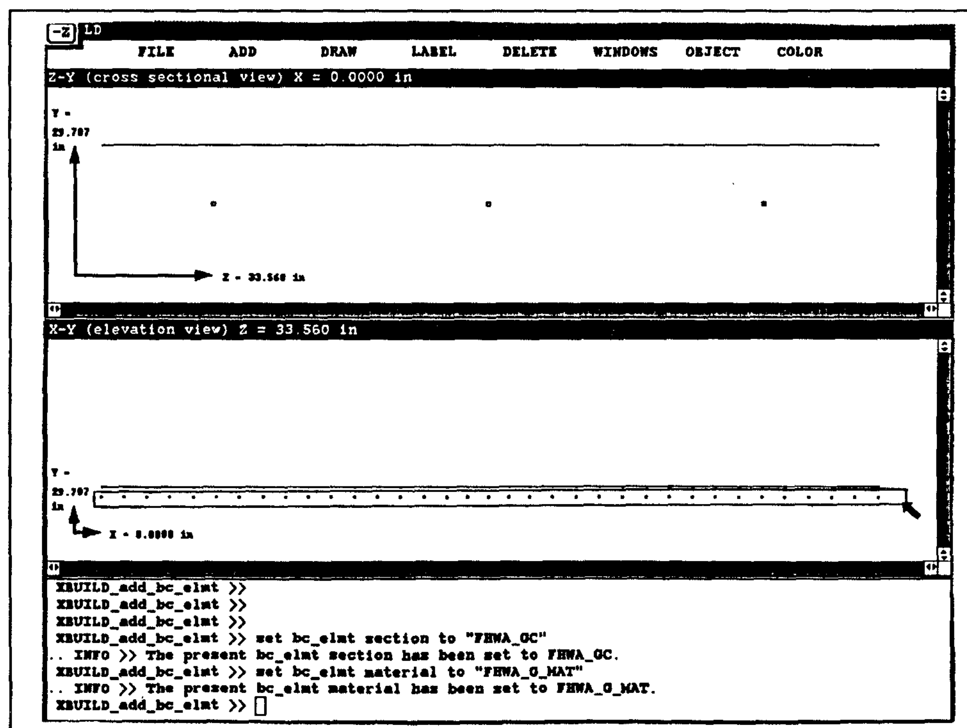


Figure 5.3. Addition of girder elements with the mouse.

```
-------------------------------------------------------------
Phase 4:  Generation of First Diaphragm.
-------------------------------------------------------------

NOTE: The vertical cross-braces, located midway between the
girders on each diaphragm, are assumed to be zero force
members and are omitted from the model.

1) XBUILD_move_node >>  <<

2) XBUILD >> add nodal_off

3) XBUILD_add_nodal_off >> x 0    y 1.707 to 25.707 by 24
z 33.56 to 196.808 by (27.208*3)

See figure 5.4.

4) XBUILD_add_nodal_off >> <

5) XBUILD_add >> node

6) XBUILD_add_node >> x 0   y 1.707   z 74.621 to
156.243 by (156.243-74.621)
7) XBUILD_add_node >> <

8) XBUILD_add >> bc_elmt

9) XBUILD_add_bc_elmt >> set bc_elmt section to "FHWA_DIA"

10) The color "black" is selected with the mouse.

11) Beam/column elements are added with the mouse as shown
in figure 5.5.

The following commands add the beam/column elements whose
nodes are difficult to grab with the mouse.

12) XBUILD_add_bc_elmt >> from Nodal_off 109 to Node 316

13) XBUILD_add_bc_elmt >> from Nodal_off 110 to Node 316

14) XBUILD_add_bc_elmt >> from Nodal_off 110 to Node 317

15) XBUILD_add_bc_elmt >> from Nodal_off 111 to Node 317

See figure 5.6.

16) XBUILD_add_bc_elmt >> save "one_dia"

The structure is saved as "one_dia.object".
```

Example: FHWA Test Bridge

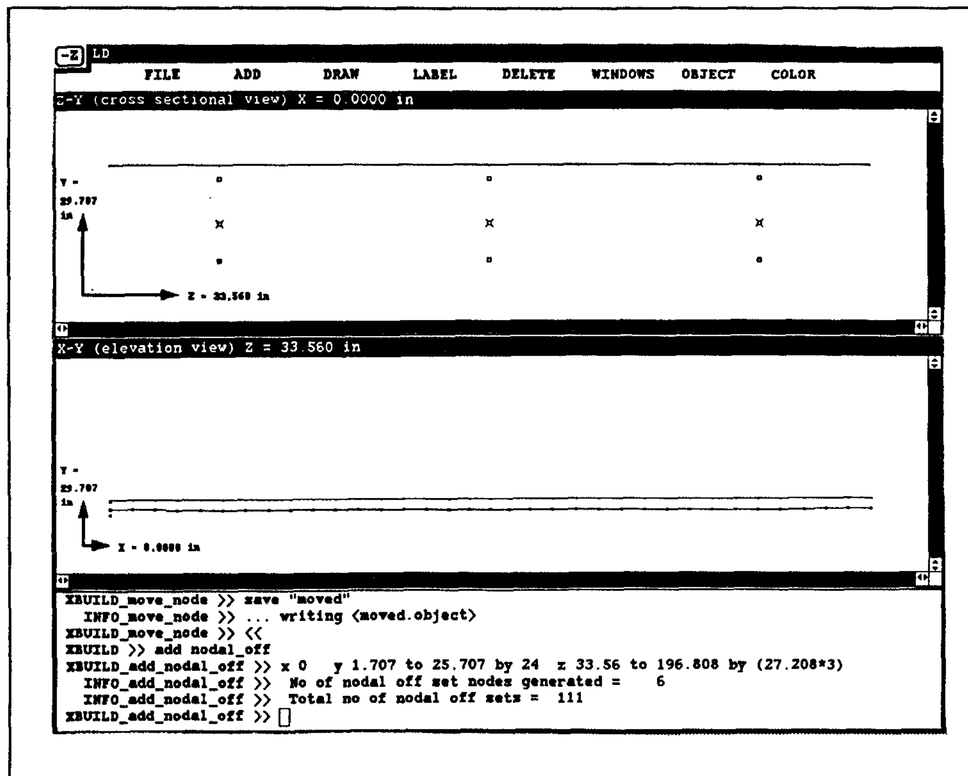


Figure 5.4. Addition of offset nodes for first diaphragm.

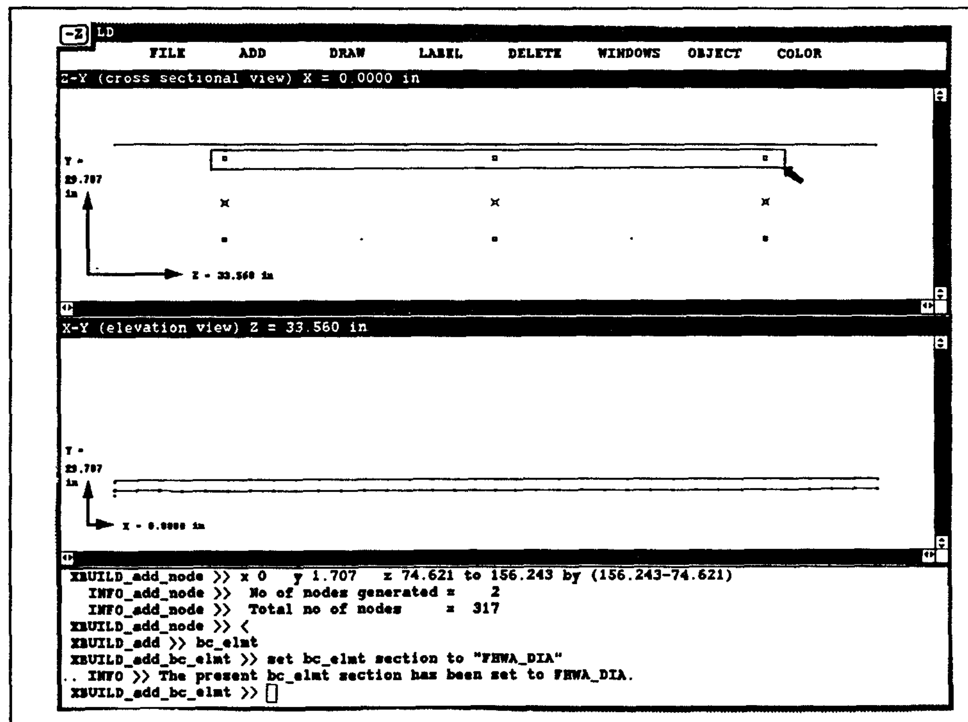


Figure 5.5. Cross braces are added with the mouse.

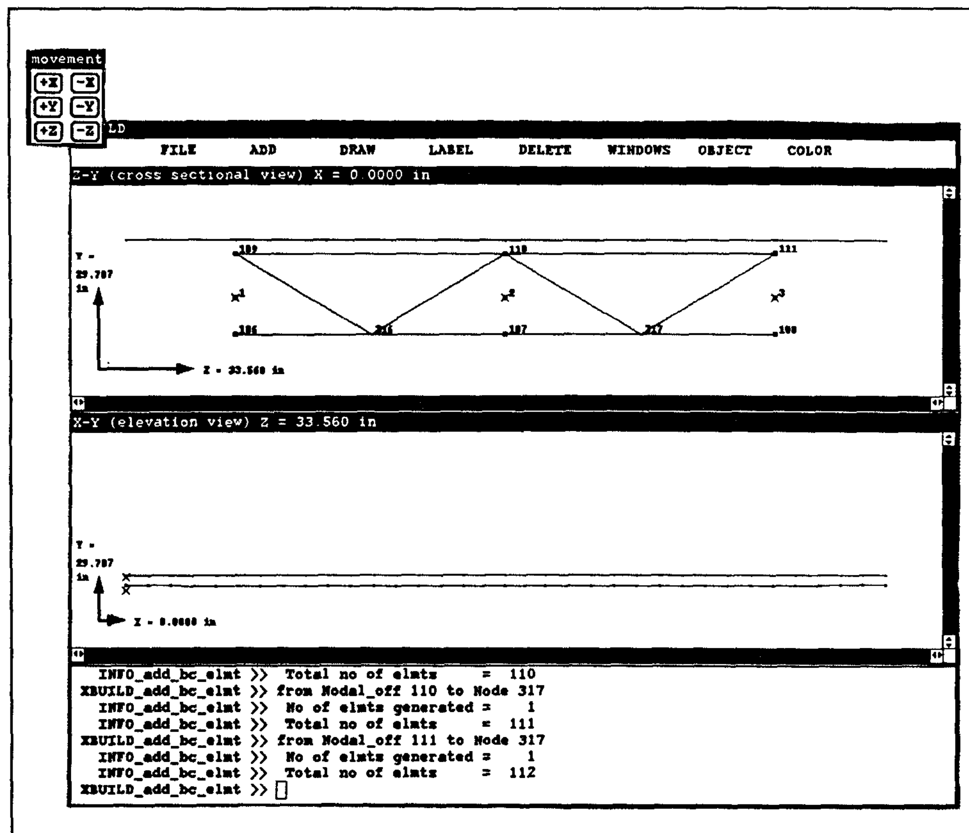


Figure 5.6. Diagonal cross braces are added with the language.

```
----------------------------------------------------------
Phase 5:  Creation of object "dia" (diaphragm).
----------------------------------------------------------

1) XBUILD_add_bc_elmt >> <<

2) XBUILD >> create object

3) XBUILD_create_object >> start object "dia"

4) XBUILD_create_object >> object
5) The beam/column elements that will belong to "dia" are
grabbed with the mouse.

The following commands add elements to "dia" that are
difficult to grab with the mouse.

6) XBUILD_create_object >> print object "dia"

7) XBUILD_create_object >> include bc_elmt 109

8) XBUILD_create_object >> include bc_elmt 110
9) XBUILD_create_object >> include bc_elmt 111

10) XBUILD_create_object >> include bc_elmt 112

11) XBUILD_create_object >> print object "dia"

See figure 5.7.

12) XBUILD_create_object >> stop object "dia"

13) XBUILD_create_object >> object

----------------------------------------------------------
Phase 6:  The object "dia" (diaphragm) is copied to
          various locations longitudinally along the bridge.
----------------------------------------------------------

1) XBUILD_create_object >> set length units to ft

2) XBUILD_create_object >> copy "dia" x 10 to 50 by 10

The object "dia" is duplicated and placed at 10 foot
intervals along the first span of the bridge (ie. at
x=10,x=20,x=30,x=40,x=50).

3) XBUILD_create_object >> copy "dia" x 56

The object ""dia" is duplicated and placed at x=56 ft, the
center of the bridge.

4) XBUILD_create_object >> copy "dia" x 112-50 to 112 by 10

The object "dia" is duplicated and placed at 10 foot
intervals along the second span of the bridge.

See figure 5.8.

5) XBUILD_create_object >> save "good"

The structure is saved as "good.object".
```

Example: FHWA Test Bridge

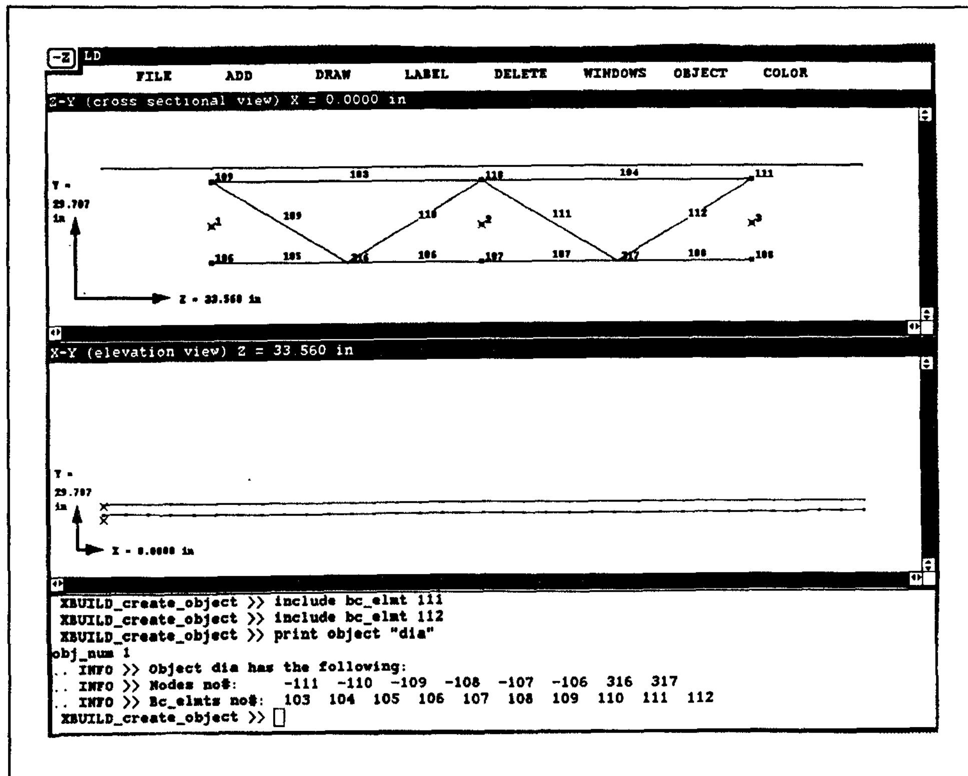


Figure 5.7. Verification of "dia" components.

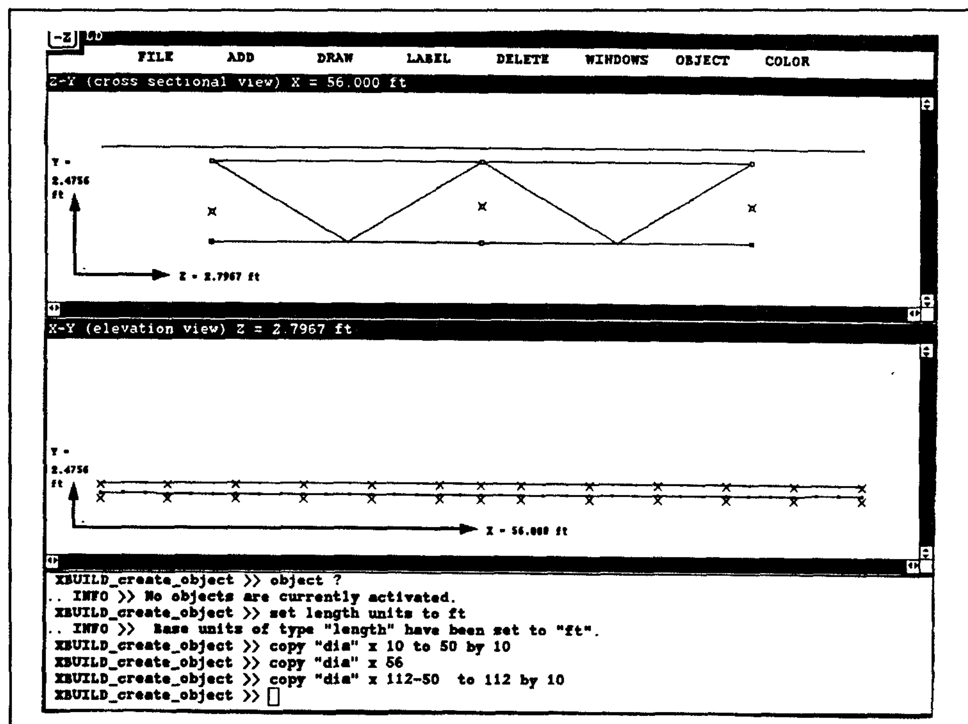


Figure 5.8. Duplication of first diaphragm along bridge.

```
----------------------------------------------------------
Phase 7:  Adding a Point Load.
----------------------------------------------------------

1) XBUILD_create_object >> <<

2) XBUILD >> add point_load

3) XBUILD_add_point_load >> set point_load magnitude to 16
kips
4) XBUILD_add_point_load >> set point_load direction to -Y

5) A node is selected with the mouse, thus applying a 16 kip
load in the -Y direction to it.

----------------------------------------------------------
Phase 8:  Fixing boundary conditions.
----------------------------------------------------------

1) XBUILD_add_point_load >> <<

2) XBUILD >> fix bcond

3) XBUILD_fix_bcond >> set bcond direction [X,Y]

Translations in the X and Y directions are declared the
boundary condition restraint attributes.

4) Nodes that are to have their displacements in the X and Y
directions restrained are grabbed with the mouse.

5) XBUILD_fix_bcond >> set bcond direction [Y]

The translation in the Y direction is declared the boundary
condition restraint attributes.

6) Nodes that are to have their displacement in the Y
direction restrained are grabbed with the mouse.

7) XBUILD_fix_bcond >> save "bridge"

The model is saved as "bridge.object".

----------------------------------------------------------
Phase 9:  ANSYS Translation.
----------------------------------------------------------

1) XBUILD_fix_bcond >> print Ansys "bridge"

An ASCII file called "bridge.dat" has been generated that is
acceptable for input to the finite elements analysis program
ANSYS [4].

----------------------------------------------------------
Phase 10:  Starting a New Problem.
----------------------------------------------------------

1) XBUILD >> clear

2) XBUILD >>
```

Example: FHWA Test Bridge

CHAPTER

SIX

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

Recent research results indicate that finite element analysis (FEA) methods are not only capable of predicting the behavior of bridge structures more accurately than approximate analysis methods, but may lead to significant economic savings in bridge design [7,8]. Still, many engineers prefer approximate bridge analysis methods simply because they are easier to use.

However, the entry of workstations into the marketplace during the past five years will raise the level of analysis in engineering firms. These workstations have computational speeds once restricted to main frame computers and come equipped with software libraries for developing customized user interfaces such as graphical windows, pull-down menus, and TTY windows for keyboard input. Together these features make it possible for engineers to take advantage of the accuracy and economic benefits of FEM's, without undue extra work.

The long term goal of this research is to formulate new design methodologies and develop computer software specifically tailored to the FEA and design highway bridges. This work will include the development of software and interactive techniques to: (a) setup finite element models of bridges, (a) analyse bridge structures with FEM, (b) graphically interpret structural behavior, (c) automatically check design specifications (or standards), (d) assist in comparing design alternatives, and (e) optimize the design.

This report documents the first 18 months of work. The main result is XBUILD, a highly interactive pre-processor for generating finite element models of bridges. Chapters 2 through 4 summarized the software development strategies, and the treatment of various problems considered. Chapter 5 described the commands needed to setup a finite element model of the bridge tested at the FHWA's Turner-Fairbanks Highway Research Center.

The benefits of using XBUILD are evident. Steve Creighton can setup a finite element model of the test bridge and automatically generate a 1700-line ANSYS compatible data file in approximately 20 minutes.

XBUILD's features are still quite limited. Work is currently underway on Version 2 of XBUILD, which will include the enhancements described below. This additional work is being performed by Chris Hudson, a graduate student at the Systems Research Center.

[1] Although bridges of arbitrary geometry may be input with Version 1, the graphical interface is currently restricted to rectangular coordinate systems. As a result, components such as a bridge diaphragm may be viewed only if its orientation is parallel to the x, y or z axes. The customized linked-list data structures described in section 4.6.2 are affected by this constraint. Relaxation of this constraint will require work in two steps. First, the notion of a general design object that is not subject to geometric restrictions needs to be formulated. For example, bridge diaphragms and decks may be neither parallel to a coordinate axis, nor, even planar. Less tangible design objects include the lanes on a bridge. Once the object is defined, step 2 of this work will be to develop interactive techniques and software for naming, viewing and performing operations on these objects. For example, a designer may want to position a substructure, join two substructures together, or specify that a truck object should incrementally move along a trajectory defined by the geometric location of a lane object. Steps 1 and 2 pose tough problems. As a starting point, research is currently underway to develop a family of interactive techniques for defining and manipulating three-dimensional geometries of nodes and finite elements.

[2] Version 1 of XBUILD can only handle point loads of fixed magnitude and orientation. To generalize the loading, we plan to define a truck object data file called `truck.dat`, which will contain definitions of standard highway trucks. A set of interactive functions will be developed to initialize and position instances of trucks on the bridge deck. Once a bridge lane object is defined, it should be a relatively straight forward step to iteratively move trucks and cars to all parts of a bridge, enabling moment envelopes to be created.

[**3**] Version 1 of XBUILD runs on SUN Workstation hardware. The decision to develop Version 1 of XBUILD using SunView was largely motivated software "bugs" in early releases of Version 11 of X windows in 1988-1989. Now that X11 is well tested, future versions of XBUILD should support the X window system developed at MIT [10]. This will allow XBUILD to run on a much wider range of hardware products, including workstations manufactured by IBM, Hewlett Packard, Digital Equipment Corporation, Apple, and of course, SUN MicroSystems.

[**4**] Finally, XBUILD needs to be extensively tested by students, faculty, and other personnel at the University of Maryland and at the FHWA test facility. The accumulation and evaluation of user feedback would provide a solid basis for system improvement.

References

[1] Aho A.V., Sethi R., Ullman J.D., **Compilers : Principles, Techniques and Tools**, *Addison-Wesley*, 1985.

[2] American Institute of Steel Construction, **Manual of Steel Construction**, Seventh Edition, 1973.

[3] Austin, M.A., "CSTRUCT: An Interactive Computer Environment for the Design and Analysis of Earthquake Resistant Steel Structures," *Report* No. UCB/EERC-87-13, Earthquake Engineering Research Center, University of California, Berkeley, September, 1987.

[4] Chamberlin D.D. and Boyce R.F., "SEQUEL : A Structured English Query Language," Proc. 1974 ACM SIGMOD **Workshop on Data Description, Access and Control**.

[5] Computech Engineering Services, Proceedings of a Workshop on Research Needs for Short and Medium Span Bridges, Sponsored by the National Science Foundation, Washington, DC., 1986.

[6] Cook, R.D, Malkus, D.S., and Plesha, M.E.,**Concepts and Applications of Finite Element Analysis**, *John Wiley & Sons*, 1974.

[7] Daniels, J.H., et al, "Weigh-in-Motion and Response of Four Inservice Bridges," *Report No.* FHWA/RD-86/045, Federal Highway Administration, Washington D.C., October, 1987.

[8] Elnahal, M.K., Albrecht, P., and Cayes L., "Load Distribution in a Two-Span Continuous Bridge",Report FHWA-RD-89-101, Federal Highway Administration, Washington D.C., 1989.

[9] Foley J.D. and Van Dam A., **Fundamentals of Interactive Computer Graphics**, Addison-Wesley Publishing Company, 1983.

[10] Gettys J., Newman R., and Fera T.D., "Xlib - C language X Interface Protocol Version 10," Digital Equipment Corporation, 1986.

[11] Goodwin N.C., "Functionality and Usability," **Communications of the ACM**, March, 1987, pp. 229-233.

[12] Gould J.D. and Lewis C., "Designing for Usability : Key Principles and what Designers Think," *Communications of the ACM*, Vol. 28, No. 3, March, 1985.

[13] Grubb, M.A., "Prototype Bridge Final Girder Design," American Institute of Steel Construction, November, 1984.

[14] GTICE Systems Laboratory, "GTSTRUDL User's Manual Volume II", School of Engineering, Georgia Institute of Technology, Atlanta, Georgia, 1986.

[15] Hartson H.R. and Hix D., "Human-Computer Interface Development: Concepts and Systems for its Management," *ACM Computing Surveys*, Vol. 21, No. 1, March 1989.

[16] Hutchins, E.L., Hollan, J.D., and Norman, D.A.,**Direct Manipulation Interfaces in User Controlled System Design**, D.A. Norman and S.W. Draper, Eds., Lawrence Associates, Hillsdale, NJ, 1986.

[17] Johnson S.C.,"YACC - Yet another Compiler Compiler," *Computer Science Technical Report 32*, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[18] Katz R.H. et al., "Design Version Management," *IEEE Design and Test*, February 1987.

[19] Kernighan B.W. and Pike R., **The UNIX Programming Environment**, *Prentice-Hall Software Series*, 1984.

[20] Kernighan B.W. and Ritchie R., **The C Programming Language**, *Prentice-Hall Software Series*, 1978.

[21] Kohnke, P.C., "ANSYS Theoretical Manual," Swanson Analysis Systems, Inc., 1977.

[22] "MSC NASTRAN Application Manual", McNeal-Schendler Corp., Los Angelos, California, 1989.

[23] Nutt, R.V., Zokaie, T., and Schamber, R.A., "Distribution of Wheel Loads on Highway Bridges," NCHRP Report, Transportation Research Board, National Research Council, Washington D.C., October, 1987.

[24] Plum T., **Reliable Data Structures in C**, *Plum Hall Inc.*, Cardiff, New Jersey, 1986.

[25] Rersner P., "Formal Grammar and Human Factors Design of an Interactive Graphics System", IEEE Trans. Software Eng, Vol. 7, No 2, 1981, pp. 229-240.

[26] Schniederman, B., **Software Psychology: Human Factors in Computer and Information Systems**, *Winthrop Publishing Company*, Cambridge, Mass., 1980.

[27] Scordelis, A.C., Chan, E.C., Ketchum, M.A., and Van Der Walt, P., "Computer Programs for Prestressed Concrete Box Girder Bridges," Report No. UCB/SESM 85-02, Department of Structural Engineering and Structural Mechanics, University of California, Berkeley, CA 94720, March, 1985.

[28] Sedgewick R., **Algorithms in C**, *Addison-Wesley Publishing Company*, 1990.

[29] "Standard Specifications for Highway Bridges," 13th Edition, American Association of State Highway and Transportation Officials, Washington, D.C., 1983.

[30] Sun Microsystems, "SunView 1 Programmer's Guide," Copyright 1982..1988, Sun Microsystems, Inc.

[31] Thompson, T., "Sun's New Workstation: the Sun386i," Byte, July, 1986, pp. 103-106.

[32] Van Wyk C.J., **Data Structures and C Programs**, *Addison-Wesley Publishing Company*, 1988.

[33] William, K.J., and Scordelis, A.C., "Computer Program for Cellular Structures of Arbitrary Plan Geometry," Structural Engineering and Structural Mechanics Report No. UC-SESM 70-10, University of California, Berkeley, September, 1970.

[34] Wilson E.L., Habibullah A., "SAP 80 : Structural Analysis Programs," *Users Guides*, Computers and Structures Inc, University Avenue, Berkeley, California, 1980.

[35] Zienkiewicz, O.C, Taylor,R.L, **The Finite Element Method**, Chapter 15, *McGraw-Hill Publishing Company*, 1989.