Taking Advantage of Advanced User Interface Tools in MSC/XL

Daniel Bryce Natalie Jaco Mark Kenyon

February 15, 1990

Abstract

Version 2 of MSC/XL has many new capabilities. Future releases of MSC/XL will continue to expand both horizontally, to support other MSC analysis packages (as Version 2 supports MSC/EMAS), and vertically, to support each analysis package in more detail. However, no matter how many options are added to MSC/XL, users will always request additional capabilities in order to help them solve their problems more effectively. This paper describes a number of external mechanisms users can take advantage of in order to extend the capabilities of MSC/XL.

This paper uses several simple examples to focus on the introduction of ways in which users may tailor MSC/XL to suit their own needs. A detailed discussion of the underlying components is beyond the scope of this paper. For more information on these concepts refer to the MSC/XL User's Manual or consult your MSC regional support staff.

Contents

List of Figures

1 Introduction

One of the initial design goals of MSC/XL was to build an expandable, easy-to-use system. From a user's point of view this meant that it must be a comfortable system which could simultaneously improve productivity. From an application programmer's point of view this meant that it must be easy to add or to modify capabilities in order to respond to the changing and expanding demands of the users.

In order to accomplish these objectives a set of programming tools was constructed to provide a structured framework for the application programmer. For example, an external menu file structure was defined so that the contents of the various menus and pop-up forms could be easily created and modified. Also, the text output facility was generalized so that not only could it be used for writing analysis input files but it could also be used for writing out the data in other formats. As the development tools mature, information will be provided to MSC/XL users explaining how they may also take advantage of these capabilities.

This paper provides an overview of some of the MSC/XL tools. Many of the tools discussed in this paper were previously available in Version 1 of MSC/XL. Tools new in Version 2 are specially noted throughout this paper. Section 2 reviews capabilities such as spell correcting, aliases, macros, calculator, reading input files, writing parts, and spawning commands. Section 3 presents a number of applications which use these tools in a variety of ways. Section 4 provides a glimpse into the expandability of the menu system and describes the new grouping mechanism. Finally, Section 5 summarizes the contents of this paper. All of the files used throughout this paper are listed in the Appendices.

2 MSC/XL Capabilities

In this section some pertinent features of MSC/XL are reviewed. In particular, overviews appear for spell correcting, command aliasing, macros, calculator expressions, reading of input files, writing of parts using a format file, spawning commands to the operating system, and new extensions to the macro processor. Those familiar with MSC/XL may wish to skim over this section. Section 3 demonstrates how to use the capabilities described in Section 2.

2.1 Spell Correcting

The MSC/XL command interpreter contains a context-sensitive spell corrector. Since the first entry on any command is the verb, the user input is first compared against a dictionary of verbs. After the spell corrector determines the closest match for the verb it then proceeds to process the user input relative to the current verb. Similarly, the input for the noun is only compared against a dictionary of nouns for that specific verb. There are also dictionaries for the verb qualifiers, data field keywords, and data field values.

Although all of the components of the command language (including verbs, verb qualifiers, data keywords, and enumerated data fields) are spell corrected, it is important to recognize that not all user input is spell corrected. In particular, macro names, variable names, and calculator function names must be entered exactly as defined. (Experimentation allowing spell correction of these additional components caused too many clashes with the core command language components.) Also keep in mind that the first character of any word must be correct in order for it to be properly spell corrected.

2.2 Aliases

Aliases allow users to use other terms for verbs and nouns. Since aliases are also included in the spell corrector dictionaries, aliases can be used to enhance MSC/XL's understanding of command abbreviations or common misspellings. For example, it is not possible for the verb spell corrector to understand **Ref** since it matches **Refresh** and **Reflect** equally well. The user can define **Ref** to be a verb alias for refresh by entering the following command.

```
Define VerbAlias Ref Refresh
```

The previous example defined an alias for a verb. Similarly, the user can define aliases for a noun associated with a specific verb. In order to define an alias for all occurrences of a noun, an individual **Define NounAlias** command is required for each verb-noun combination.

2.3 Macros

The macro capability of MSC/XL is essentially a string substitution facility. Whenever the command parser encounters a word (an alphanumeric string) it first looks up the word in the set of macros. If a valid macro definition is located, the body of the macro is substituted into the input string as if the user actually typed in the text. Macros can be simple string substitution or parameterized string substitution. Shown below is the **fm** (Find Model) macro as defined in the standard **Startup.inp** file.

```
Define Macro "fm" "Refresh/Find View"
```

Startup.inp (located in the MSC/XL delivery directory) contains several useful macro definitions. A copy of this file is shown in Appendix A.

Although it is perfectly acceptable to have macros reference other macros, this may not work in all cases because of command line length limitations. No command line input may exceed 1000 characters in length. Since the macro capability operates by substituting the body of a macro for the macro name in the command line, the length of the command line will generally increase with every macro encountered. In order to circumvent this limitation, the user may need to use a combination of macros and input files.

One final note: All of a macro's parameters must be used somewhere in the body of that macro. If any of the parameters are not used, MSC/XL will issue an error message. Several examples of parameterized macros appear later in this paper.

2.4 Calculator

The MSC/XL calculator allows the user to enter an expression in place of a number. A calculator expression may include constants, variables, common arithmetic operations, standard math functions, and special MSC/XL access functions. The complete list of calculator functions appears in the MSC/XL User's Manual. In order to enter a calculator expression, the user simply encloses the expression in parentheses. For example, the following command will list the distance of point 5 from the origin. In this example `PointX`, `PointY`, and `PointZ` are geometry calculator functions which evaluate to the X, Y, and Z coordinates of the point (5) whose identifier is passed as a parameter.

```
List Expression (sqrt(PointX(5)*PointX(5) +
                      PointY(5)*PointY(5) +
                      PointZ(5)*PointZ(5)))
```

Likewise, calculator expressions can be used in conjunction with macros. The next simple example shows the definition of a macro which can be used to define a point midway between two existing points.

```
Define Macro "DefMidPt(Point1Id,Point2Id)" -
             "Define Point -
                ((PointX(Point1Id)+PointX(Point2Id))/2.0) -
                ((PointY(Point1Id)+PointY(Point2Id))/2.0) -
                ((PointZ(Point1Id)+PointZ(Point2Id))/2.0)"
```

The following invocation of the macro will result in creation of a new point between points 3 and 6.

```
DefMidPt(3,6)
```

The calculator also recognizes user-defined variables. `Define Variable` is used to create a variable and to optionally assign an initial value to it. Since the initial value is a real number, the calculator may be used to compute its value from an expression. The following example defines the variable `Theta` with an initial value of 45 (degrees). The variable `DegreesToRadians` is defined with an initial value equal to the factor for conversion of degrees to radians. These two variables are then used in a calculator expression to list the value of `Theta` in radians.

```
Define Variable Theta 45
Define Variable DegreesToRadians (acos(-1)/180.0)

List Expression (Theta * DegreesToRadians)
```

2.5 Reading Input Files

MSC/XL provides an alternative to interactively typing commands by allowing the user to read in a file containing MSC/XL commands. Common methods for creating these input files include using an editor, extracting portions of a history file, or invoking user-written programs. Section 3.5 contains a good example of a user-written program being used to generate a sequence of MSC/XL commands.

In its simplest form, the `Read File` command accepts one parameter, a file name. If a file name is not specified, MSC/XL presents the file tool to prompt the user for the name of the file. (By convention `.inp` is used as the file name extension.)

Anyone who has used MSC/XL has already witnessed an example of the use of `Read File`. When MSC/XL starts up, the following commands are automatically issued.

```
Read File "XL_PATH:Startup.inp"
Read File "USER_PATH:Startup.inp"
Read File "Startup.inp"
```

Since these three `Startup.inp` files are automatically read in on system startup, a user's system-tailoring commands may be added to system initialization by editing a local or a logically-defined `Startup.inp` file.

In Version 2, `Startup.inp` is read in before any screen graphics updates are done. This makes it more convenient for the user to tailor the system by repositioning tiles, modifying label table entries, defining macros, or reading in macro definition files. Input files may contain `Read File` commands, therefore one input file may read in other input files.

Input files may be tailored by passing in parameters separated by commas. Within the input file the names of the parameters are `P1, P2, ..., Pn`. Therefore it is essential that the user avoid creating macros, aliases, or any other command interface entities with the character `P` followed by a sequence of numbers.

The following example demonstrates the use of an input file to define a point midway between two other points.

1. Create DefMidPt.inp with the following contents.

```
Define Point -
        ((PointX(P1)+PointX(P2))/2.0) -
        ((PointY(P1)+PointY(P2))/2.0) -
        ((PointZ(P1)+PointZ(P2))/2.0)
```

2. Now to create a point between points 3 and 6 enter the following command.

```
Read File DefMidPt.inp 3,6
```

2.6 Writing Parts

Once the user has created a model in MSC/XL, the model information may be written out for use by external programs. The MSC/NASTRAN bulk data format represents one obvious format for writing out the finite element data. However, the bulk data format may not be the best format for other applications. Also, the bulk data file does not contain all of the information available in MSC/XL. Most notably, none of the geometry (point, curve, surface, and solid) information is written to the bulk data file. The `Write Part` command provides a tool with which the user may access all fields of the entities stored in parts. Furthermore, the user has some control over the actual format of the file.

In order to use the `Write Part` command the user must first create a format file. A format file contains a sequence of entries, each entry having three main components. The first component is the name of the MSC/XL entity of interest. The second component is a C-style format string. The third component describes the parameters for the format string and consists of an integer and a number of variables. The integer defines the number of variables to follow. The variables are the names of fields in the particular entity specified by the first component. One example of a format file is the file `Bulk.fmt` located in the MSC/XL delivery directory. (Note, this is a special type of format file which is used to write an MSC input file. This file can not be used directly with the `Write Part` command.) The file `XLayout.ddl` in the MSC/XL delivery directory describes all of the valid variables for each entity recognized by MSC/XL.

The following format may be used to generate an input file with MSC/XL commands suitable for recreating the geometric information associated with GRIDs and QUAD4s.

```
Grid
    "Edit/Create Grid/%d X=%g Y=%g Z=%g CP=%d CD=%d
"    6 Identifier X Y Z CP CD
Quad4
    "Edit/Create Quad4/%d G1=%d G2=%d G3=%d G4=%d
"    5 Identifier G1 G2 G3 G4
```

Assuming the name of above format file is `GenInp.fmt`, the following `Write Part` command can be used to create an input file capable of recreating the GRIDs and QUADs in Part 0.

```
Write Part/0 Format="GenInp" File="InputFile.inp"
```

Limitations: The format file must be in the local directory, and its name must use the `.fmt` extension.

2.7 Spawning a Command

Combinations of the capabilities presented above provide the basis for performing sophisticated operations. When features such as **Write Part** and **Read File** are used in conjunction with **Spawn Command**, the range of capabilities available to the user is greatly expanded. For example, the user may first write out the entities associated with a part in an easy-to-process format. Second, the user may write a program which accepts that format file as input and which creates an MSC/XL input file. Finally, the user may read in the generated input file. **Spawn Command** provides the tool which allows all three steps to be done without leaving MSC/XL. **Spawn Command** requires one parameter, the operating system command to be executed.

The following command lists the contents of the user's local directory on a UNIX system.

```
Spawn Command "ls"
```

Section 3.5 provides a good example of how **Spawn Command** can be used to invoke a user-written application program.

2.8 Vector Operations in Macros

The macro facility in MSC/XL Version 2 includes the ability to pass in character strings as parameters and to force macro parameter substitution independent of the context. In Version 1, a word was only recognized as a parameter if appropriately delimited by spaces. In Version 2, a parameter may appear in the body of a macro if it is enclosed by back quotes (‘).

For example, the following macro definition may be used to define vectors as three calculator variables.

```
Define Macro "VectorDefine(VecName, V1, V2, V3)" -
        "Define Var ‘VecName‘1 (V1); -
        Define Var ‘VecName‘2 (V2); -
        Define Var ‘VecName‘3 (V3)"
```

Now, the following commands may be used to define vectors a and b.

```
VectorDefine(a,1,1,0)
VectorDefine(b,1,4,(sqrt(7)/2.0))
```

Appendix B lists several vector operations which will be used later in this paper.

3 Applications

The previous section reviewed some basic components of MSC/XL. This section will demonstrate, through the use of several examples, how these components may be used. The techniques presented in these examples purposely demonstrate a variety of approaches applicable to many different situations.

Each example application contains four steps. First, the problem is defined. Second, the tools used in the solution are listed. Third, a solution overview is presented. Finally, the solution description is expanded to show the commands needed for its implementation. In many cases, there are several good solutions to the problem, some, in fact, better than the one offered in the example. The intent is to present approaches (although not always obvious ones) which may be applicable to more complex cases.

3.1 Breaking Up a Model into Parts Corresponding to the PID

Problem Setup: Given a plate model containing only QUAD4, TRIA3, and BAR elements which all belong to Part 0, split up the model into parts 101, 102, 103, 104, and 105 corresponding to those entities with PID 1, 2, 3, 4, and 5, respectively.

Tools Used in the Solution:

- `Write Part`
- `Read File`

Solution Overview:

1. Create a format file containing `Put Element` commands.
2. Create the needed parts.
3. Use `Write Part` with the newly created format file to create an input file.
4. Use `Read File` to read in the input file.

Solution Steps:

1. Create `Put.fmt` as shown below.
2. `Edit/Create Part/101 To 105`
3. `Write Part/0 Format="Put" File="Put.inp"`
4. `Read File "Put.inp"`

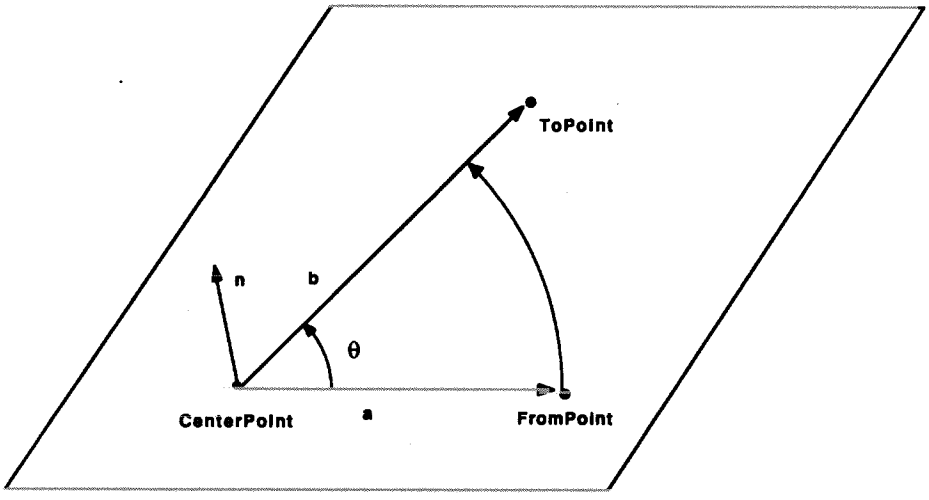

Figure 1: Circular Arc From Three Points

```
File Put.fmt

Quad4
"Put/NoUpdate Element/%d Into=(100+%d)
"        2 Identifier PID
Tria3
"Put/NoUpdate Element/%d Into=(100+%d)
"        2 Identifier PID
Bar
"Put/NoUpdate Element/%d Into=(100+%d)
"        2 Identifier PID
```

3.2 Creating a Circular Arc Using Three Points

Problem Setup: Define a section of a circular arc using three points as shown in Figure 1. The first point defines the center of the circular arc, the second point defines the radius of the circular arc and the origin of the arc, and the third point (together with the first two) defines the plane in which the circular arc lies and the line at which the circular arc will terminate.

Tools Used in the Solution:

- `Define Macro`

- Calculator Variables
- Calculator Geometry Access Functions
- Other Calculator functions

Solution Overview:

1. Define vectors $\vec{a}$ and $\vec{b}$.
2. Define vector $\vec{n}$ (normal to the plane) as $\vec{a} \times \vec{b}$.
3. Define the angle between $\vec{a}$ and $\vec{b}$ as $\arccos(\vec{a}.\vec{b})$.
4. Issue the corresponding `Sweep Point` command.

Solution Steps:

The Center2Point (C2P) macro accomplishes all of the above steps and is shown below.

```
Define Macro "C2P(CenterId, FromId, ToId, NumOfCurves)" -
    "Echo Off;-
    Define Var a1 (PointX(FromId) - PointX(CenterId));-
    Define Var a2 (PointY(FromId) - PointY(CenterId));-
    Define Var a3 (PointZ(FromId) - PointZ(CenterId));-
    Define Var b1 (PointX(ToId) - PointX(CenterId));-
    Define Var b2 (PointY(ToId) - PointY(CenterId));-
    Define Var b3 (PointZ(ToId) - PointZ(CenterId));-
    Define Var ra (sqrt(a1*a1 + a2*a2 + a3*a3));-
    Define Var rb (sqrt(b1*b1 + b2*b2 + b3*b3));-
    Define Var theta (acos( (a1*b1 + a2*b2 + a3*b3) / (ra*rb)));-
    Define Var nx (a2*b3 - a3*b2);-
    Define Var ny (a3*b1 - a1*b3);-
    Define Var nz (a1*b2 - a2*b1);-
    Sweep/Points NumOfCurves Point/FromId -
        (PointX(CenterId)) (PointY(CenterId)) (PointZ(CenterId)) -
        (PointX(CenterId) + nx) -
        (PointY(CenterId) + ny) -
        (PointZ(CenterId) + nz) -
        (theta/NumOfCurves);-
    Echo On"
```

Note: The equivalent operation could have been done using an input file. To set up the operation in this way, the body of the macro must be placed in an input file and the macro parameters CenterId, FromId, ToId, and NumOfCurves must be changed to `P1, P2, P3,` and `P4` respectively. Then, instead of invoking the operation by typing in `C2P(13, 2, 3, 2)`, the operation could be accomplished by typing in the following.

```
Read File C2P.inp 13, 2, 3, 2
```

3.3 Vector Operations

Problem Setup: As an alternative to the basic tools used in the previous operation, rework the problem using a combination of macros, vector operations, and `Read File`.

Tools used in the Solution:

- Macros
- Vector Operations
- `Read File`

Solution Overview:

1. Create a file with the equivalent functionality as the `C2P` macro shown in Section 3.2.
2. Read in an input file defining the vector macros.
3. Create a macro C2P which reads in the input file.

Solution Steps:

1. Create `C2PVector.inp` as shown below.
2. `Read File "Vector.inp"`
3. ```
   Define Macro "C2P(CenterId, FromId, ToId, NumOfCurves)" -
   "Read File C2PVector.inp CenterId, FromId, ToId, NumberOfCurves"
   ```

File `C2PVector.inp`:

```
VectorFromPoints(a, P2, P1)
VectorFromPoints(b, P3, P1)
Define Var ra VectorLength(a)
Define Var rb VectorLength(b)
Define Var theta (acos( VectorDot(a,b) / (ra*rb)))
VectorCross(n, a, b)
Sweep/Points NumOfCurves Point/FromId -
    (PointX(CenterId)) (PointY(CenterId)) (PointZ(CenterId)) -
    (PointX(CenterId)+n1) -
    (PointY(CenterId)+n2) -
    (PointZ(CenterId)+n3) -
    (theta/NumOfCurves)
```
```

3.4 Aligning Parts

Problem Setup: Given two parts, move one of them so that it lines up with the other part. For example, consider two parts, one representing a straight pipe and the other an elbow pipe. Figure 2 shows the initial and the desired configurations for these two parts.

Tools Used in the Solution:

- Vector Operations
- Calculator
- `Read File`
- `Translate Part`
- `Rotate Part`

Solution Overview:

1. Partition the model into the desired two parts.
2. Create the six reference points, if they do not already exist.
3. Create an input file which performs the following operations
   (a) Translate Part 1 to make Points 2 and 3 coincide (Figure 3a).
   (b) Rotate Part 1 to align the lines defined by Points 2-4 and 3-5 (Figure 3b).
   (c) Rotate Part 1 to align the planes defined by Points 2-4-6 and 3-5-7 (Figure 3c).

Solution Steps:

1. Create `Align.inp` as shown in Appendix E.
2. `Read File "Align.inp" 1, 2,5, 3,6, 4,7`

3.5 Applying Pressure Loads to a Pipe

Problem Setup: Given a pipe composed of solid HEXA elements, apply a uniform pressure load to faces along the inside of the pipe as shown in Figure 4.

Tools Used in the Solution:

- `Check Element`
- `Write Part`

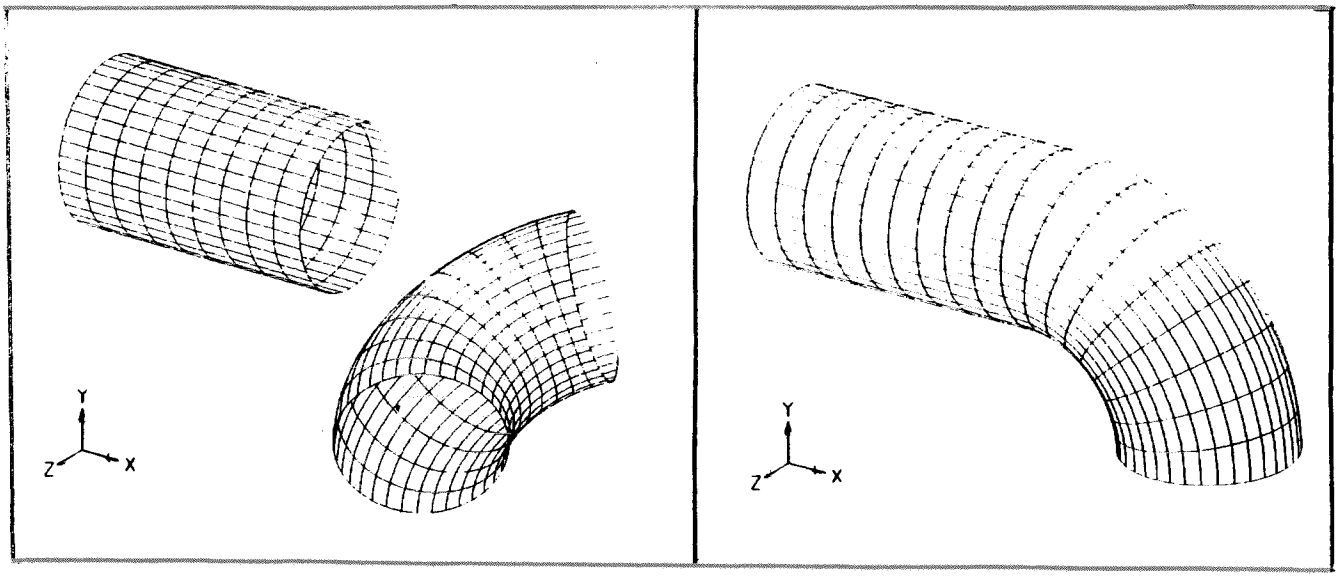

Figure 2: Aligning Two Parts

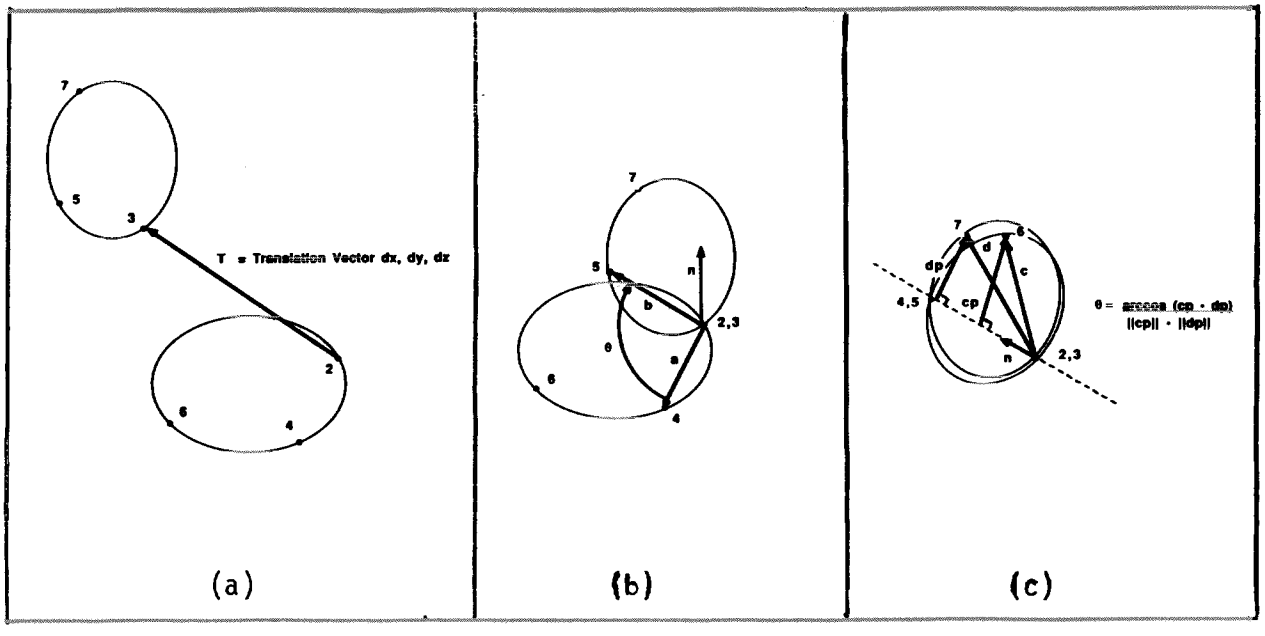

Figure 3: Steps in Aligning Two Parts

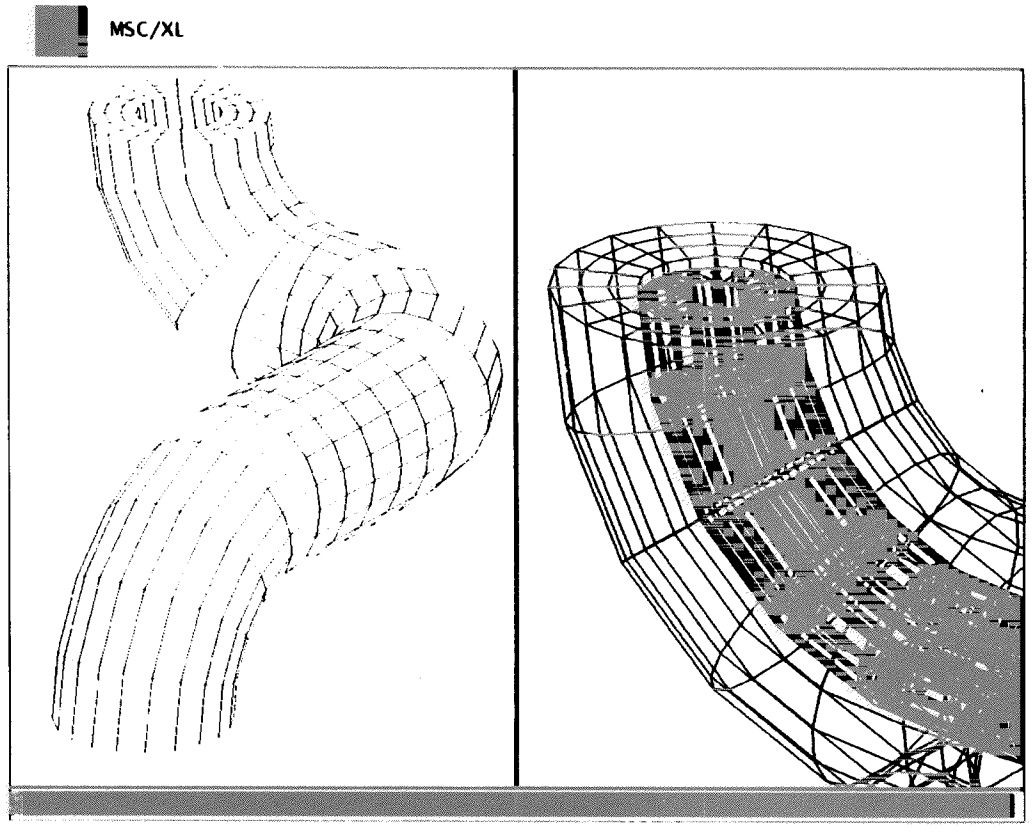

Figure 4: Applying Pressure Loads

- `Spawn Command`
- `Read File`

Solution Overview:

1. Find the Free Faces.
2. Write out the GRIDS and Free Faces.
3. Execute a user-written program which accepts as input the GRIDS, the FACES, the identifier of the face of interest, and a maximum angle. The output of this program will be an input file containing the necessary `Edit PLOAD4` commands. Starting with a set containing only the given face of interest, this program will compute the transitive closure of faces for which the normal differs by no more than the maximum angle.
4. Read in the `PLOAD4.inp` file.

Solution Steps:

1. `Check Element FreeFace`
2. `Write Part/0 Format="Faces" File="Faces.dat"`. Create a format file to write out the GRIDS and FACES (Faces.fmt is shown in Appendix F.1).

3. `Spawn Command "GenPressures Faces.dat 15 45 n"`

   (Invoke the user-written program given that 15 is the identifier of a free face on the inside of the pipe, 45 degrees is a good maximum angle, and the normal of face 15 points outward. `GenPressures.c` is shown in Appendix F.2.)

4. `Read File "PLoad4.inp"`.

4 Selected New Features in MSC/XL Version 2

The previous section demonstrated how several basic components of MSC/XL can be effectively combined to provide creative solutions for specific user problems. Extensions or modifications to the standard MSC/XL user interface can make access to these solutions even easier. New tools in MSC/XL Version 2 simplify access to the large number of entities manipulated in real-world problems and their solutions.

Section 4.1 demonstrates how the user interface may be extended to allow immediate access to user-defined capabilities. Section 4.2 overviews the new grouping mechanism which allows a user to more simply manipulate a large subset of differently related entities.

4.1 Extending the User Interface

Although the steps involved in changing the user interface will not be detailed, modified MSC/XL screens will be presented. In particular, we will show the results of adding the MidPt and C2P macros to the QAMs (Quick Access Menus).

Figure 5 shows the usual MSC/XL screen with two modifications. First, the QAMs contain two lines instead of the usual single line. Second, the QAMs have a scroll bar. Also notice that MidPt and C2P appear as options on the second line of the QAM. Figures 6 and 7 show the pop-up forms which are presented when the user selects the MidPt and C2P options respectively. They appear and work the same as all standard MSC/XL forms. The only difference is that instead of directly accessing internal operations, these options invoke macros.

The input file and the menu text files required for this user interface extension are shown in Appendix G. The input file (`ExtendUI.inp`) defines the calculator variables used in the pop-up forms and the macros which relate the menu forms to their respective parameterized macros. The menu text file QAM.txt adds MidPt and C2P as options to the QAM tile. (Additional system commands are required to activate the new QAM contents and the scroll bars.) The last two menu text files (`MidPtPop.txt` and `C2PPop.txt`) actually define the pop-up forms which will allow the specification of parameters required by the `MidPt` and `C2P` macros.

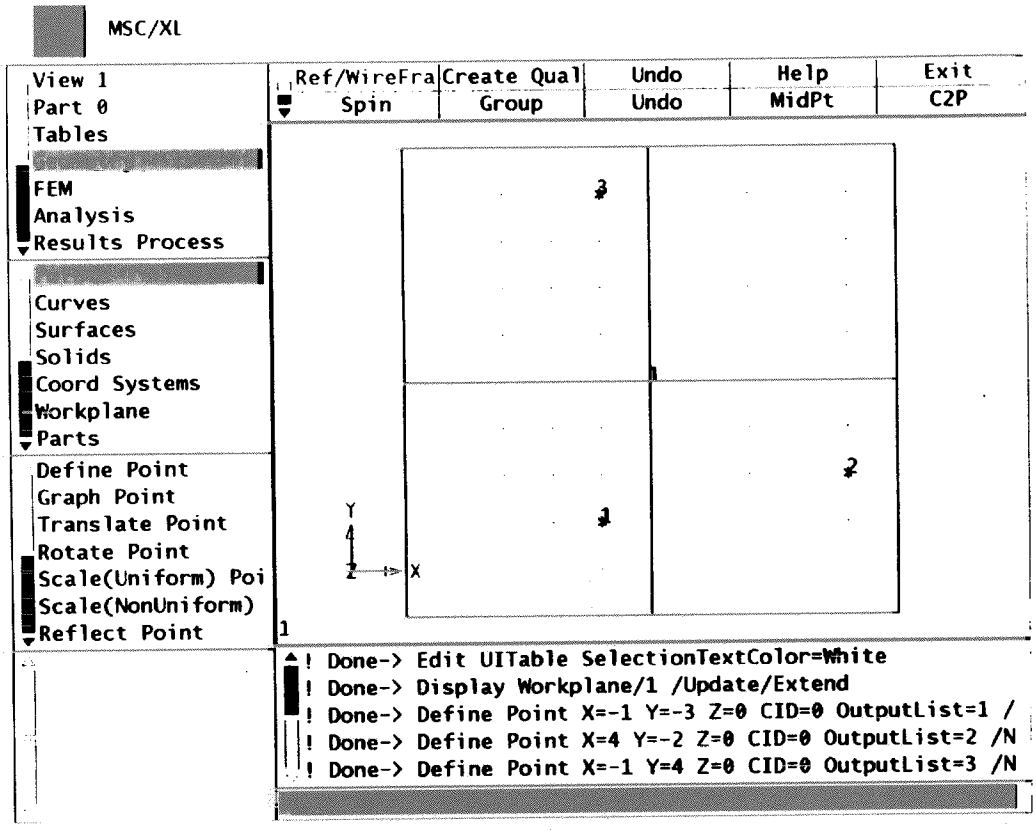

Figure 5: Extended MSC/XL User Interface

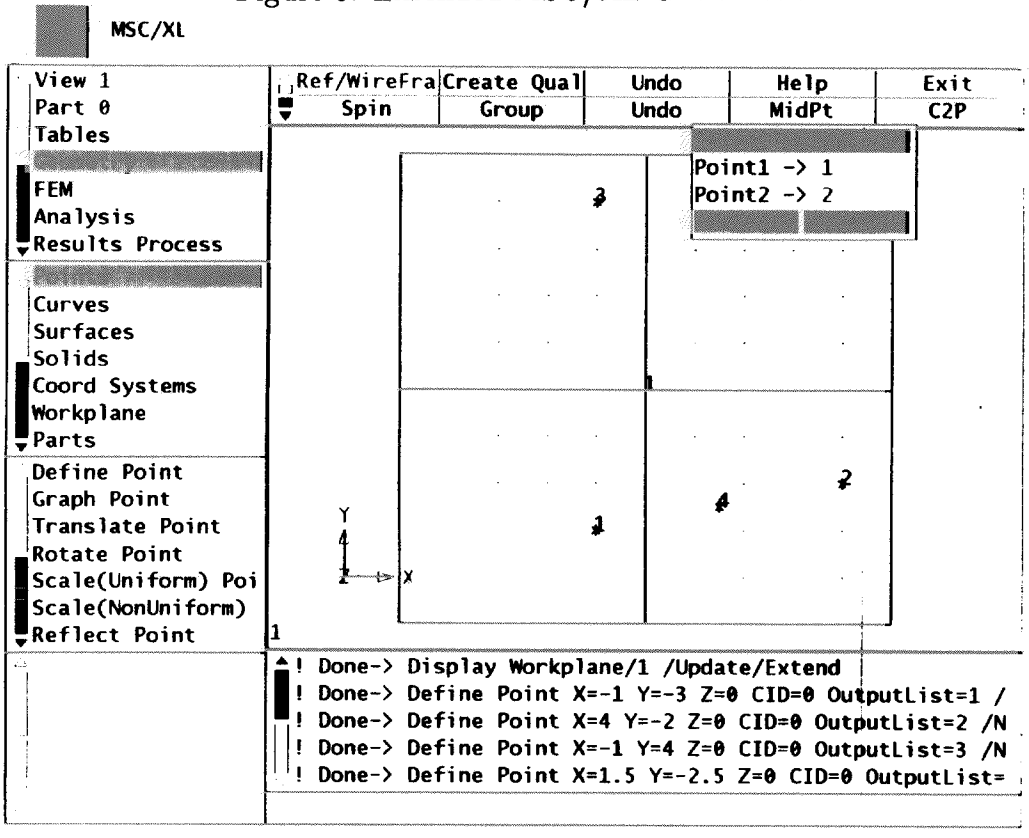

Figure 6: Use of the MidPoint Macro

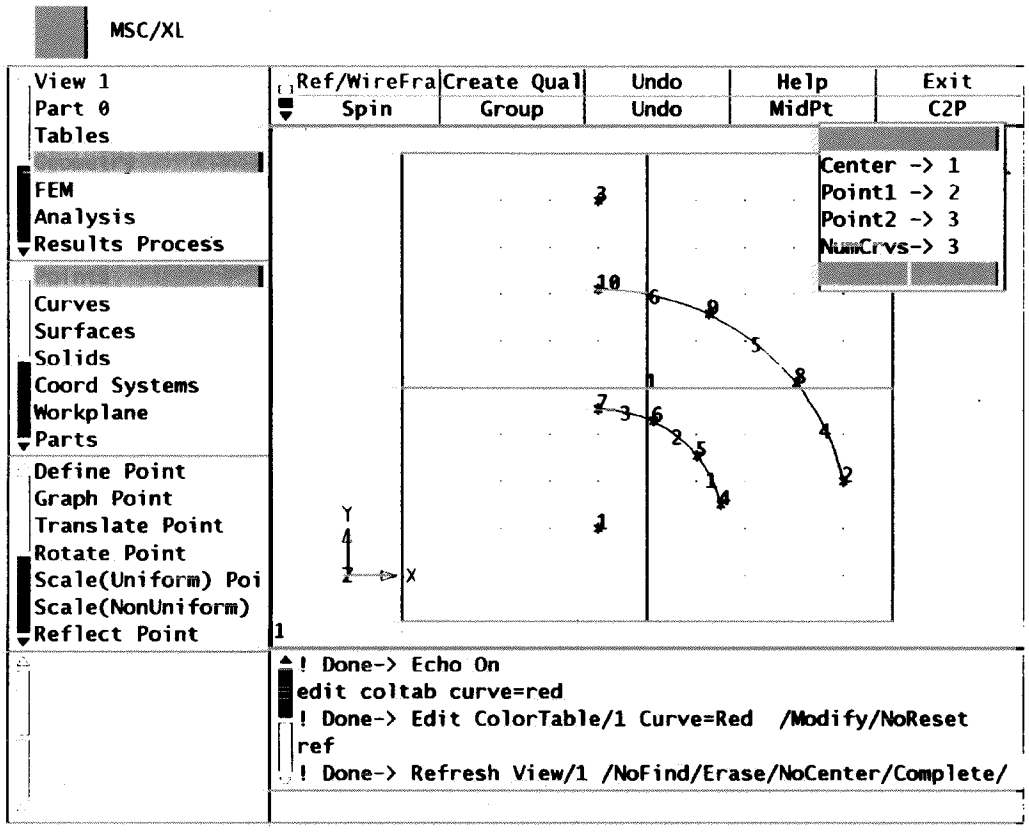

Figure 7: Use of the C2P Macro

4.2 Grouping

Grouping is one particularly important tool introduced in MSC/XL Version 2. The grouping tool can be used to construct groups of element identifiers by:

- specifying specific Point, Curve, Surface, Solid, Grid, or Element identifiers,
- selecting a region on the screen by using a rectangular or polygon window,
- specifying an arbitrary criterions (such as PID=1), and
- intersecting and unioning existing groups.

Once a group is defined, it may be used where an `IDList` is required. Therefore, the user may easily delete, display, and erase elements whose identifiers are included in a group, and the user may also easily define parts based on groups.

A complete list of enhancements contained in Version 2 will be delivered with the system. All of the files listed in the appendices will also be included in the MSC/XL Version 2 delivery directory.

5 Summary

This paper presented several basic tools which can be used to extend the immediate capabilities of MSC/XL. Some of the tools were present in Version 1, and others were added in Version 2. The application problems presented above illustrated how a user can perform operations which do not appear to be readily available. In particular, the `Write Part/Read File` solution shown for splitting a model into parts (refer to Section 3.1) represents one way to address such a problem. Because of the additional capabilities of Version 2, there is a much more straightforward solution to this particular problem. However, the idea of using format files applies to many other situations.

MSC/XL Version 1 provided an interface to MSC/NASTRAN. MSC/XL Version 2 provides an interface to MSC/NASTRAN and MSC/EMAS. Future releases will expand to provide a common interactive user interface to the full range of MSC analysis packages. As this expansion occurs, the variety of applications will require different set of tools. MSC/XL will not only provide bundled tools to help users solve specific problems, but will also continue to provide user tools allowing expansion or tailoring of the system to meet specific requirements.

A Startup.inp

```
Define Macro "fm" "Refresh/Find View"
Define VerbAlias ref refresh
Define NounAlias line connect points
!
Define NounAlias MSCNASTRANInput Read  MSCInput
Define NounAlias MSCEMASInput   Read  MSCInput
Define NounAlias MSCNASTRANInput Write MSCInput
Define NounAlias MSCEMASInput   Write MSCInput
!
Define Macro "XAxis" "0 0 0 1 0 0"
Define Macro "YAxis" "0 0 0 0 1 0"
Define Macro "ZAxis" "0 0 0 0 0 1"
!
Define Macro "AtGrid(g1)"   "(GridX(g1)) (GridY(g1)) (GridZ(g1))"
Define Macro "AtPoint(PointId1)"  "(PointX(PointId1))
                                   (PointY(PointId1))
                                   (PointZ(PointId1))"
Define Macro "MidPt(PointId1,PointId2)"
              "((PointX(PointId1)+PointX(PointId2))/2.0) -
               ((PointY(PointId1)+PointY(PointId2))/2.0) -
               ((PointZ(PointId1)+PointZ(PointId2))/2.0)"
!
```

```
Define Macro "OnCurve(CurveId,T)" "(CurveXat(CurveId,T)) -
                                  (CurveYat(CurveId,T)) -
                                  (CurveZat(CurveId,T))"
Define Macro "OnSurface(SurfaceId,U,V)" "(SurfaceXat(SurfaceId,U,V)) -
                                         (SurfaceYat(SurfaceId,U,V)) -
                                         (SurfaceZat(SurfaceId,U,V))"
Define Macro "OnSolid(SolidId,U,V,W)" "(SolidXat(SolidId,U,V,W)) -
                                       (SolidYat(SolidId,U,V,W)) -
                                       (SolidZat(SolidId,U,V,W))"
!
Define Macro "gamma(g)" "1 (g / (1-g)) 1"
Define Macro "pratio(p)" "p (1-p) (1-p) p"
!
Define Macro "Side1" "0 1 0 0 0 0"
Define Macro "Side2" "1 1 0 1 0 0"
Define Macro "Side3" "0 1 1 1 0 0"
Define Macro "Side4" "0 0 0 1 0 0"
Define Macro "Side5" "0 1 0 0 1 1"
Define Macro "Side6" "1 1 0 1 1 1"
Define Macro "Side7" "0 1 1 1 1 1"
Define Macro "Side8" "0 0 0 1 1 1"
Define Macro "Side9" "0 0 0 0 0 1"
Define Macro "Side10" "1 1 0 0 0 1"
Define Macro "Side11" "1 1 1 1 0 1"
Define Macro "Side12" "0 0 1 1 0 1"
Define Macro "Face1" "0 0 0 0 0 1 1 0 0 0 1 1"
Define Macro "Face2" "1 1 1 1 0 1 1 0 0 0 1 1"
Define Macro "Face3" "0 1 1 0 0 0 0 0 0 0 1 1"
Define Macro "Face4" "0 1 1 0 1 1 1 1 0 0 1 1"
Define Macro "Face5" "0 1 1 0 0 0 1 1 0 0 0 0"
Define Macro "Face6" "0 1 1 0 0 0 1 1 1 1 1 1"
```

B Vector Operation Macros

```
Define Macro "VectorDefine(VecName, V1, V2, V3)" -
        "Define Var `VecName`1 (V1); -
        Define Var `VecName`2 (V2); -
        Define Var `VecName`3 (V3)"

Define Macro "VectorFromPoints(VecName, Origin, Terminus)" -
        "Define Var `VecName`1 (PointX(Terminus) - PointX(Origin));-
        Define Var `VecName`2 (PointY(Terminus) - PointY(Origin));-
        Define Var `VecName`3 (PointZ(Terminus) - PointZ(Origin))"
```

```
Define Macro "VectorAdd(C, A, B)" -
        "Define Var 'C'1 ('A'1 + 'B'1); -
        Define Var 'C'2 ('A'2 + 'B'2); -
        Define Var 'C'3 ('A'3 + 'B'3)"

Define Macro "VectorSub(C, A, B)" -
        "Define Var 'C'1 ('A'1 - 'B'1); -
        Define Var 'C'2 ('A'2 - 'B'2); -
        Define Var 'C'3 ('A'3 - 'B'3)"

Define Macro "VectorDot(A, B)" "('A'1*'B'1 + 'A'2*'B'2 + 'A'3*'B'3)"

Define Macro "VectorCross(C, A, B)" -
        "Define Var 'C'1 ('A'2*'B'3 - 'A'3*'B'2); -
        Define Var 'C'2 ('A'3*'B'1 - 'A'1*'B'3); -
        Define Var 'C'3 ('A'1*'B'2 - 'A'2*'B'1)"

Define Macro "VectorLength(A)" -
       "( sqrt('A'1*'A'1 + 'A'2*'A'2 + 'A'3*'A'3))"

Define Macro "VectorUnit(U, A)" -
        "Define Var LengthOf'A' -
                (sqrt('A'1*'A'1 + 'A'2*'A'2 + 'A'3*'A'3)); -
        Define Var 'U'1 ('A'1 / LengthOf'A'); -
        Define Var 'U'2 ('A'2 / LengthOf'A'); -
        Define Var 'U'3 ('A'3 / LengthOf'A')"
```

C C2P Macro Definition

```
Define Macro "C2P(CenterId, FromId, ToId, NumOfCurves)" -
        "Echo Off;-
        Define Var a1 (PointX(FromId) - PointX(CenterId));-
        Define Var a2 (PointY(FromId) - PointY(CenterId));-
        Define Var a3 (PointZ(FromId) - PointZ(CenterId));-
        Define Var b1 (PointX(ToId) - PointX(CenterId));-
        Define Var b2 (PointY(ToId) - PointY(CenterId));-
        Define Var b3 (PointZ(ToId) - PointZ(CenterId));-
        Define Var ra (sqrt(a1*a1 + a2*a2 + a3*a3));-
        Define Var rb (sqrt(b1*b1 + b2*b2 + b3*b3));-
        Define Var theta (acos( (a1*b1 + a2*b2 + a3*b3) / (ra*rb)));-
        Define Var nx (a2*b3 - a3*b2);-
        Define Var ny (a3*b1 - a1*b3);-
```

```
        Define Var nz (a1*b2 - a2*b1);-
        Sweep/Points NumOfCurves Point/FromId -
                (PointX(CenterId)) (PointY(CenterId)) (PointZ(CenterId)) -
                (PointX(CenterId) + nx) -
                (PointY(CenterId) + ny) -
                (PointZ(CenterId) + nz) -
                (theta/NumOfCurves);-
        Echo On"
```

D C2PVector Input File

```
VectorFromPoints(a, P2, P1)
VectorFromPoints(b, P3, P1)
Define Var ra VectorLength(a)
Define Var rb VectorLength(b)
Define Var theta (acos( VectorDot(a,b) / (ra*rb)))
VectorCross(n, a, b)
Sweep/Points NumOfCurves Point/FromId -
        (PointX(CenterId)) -
        (PointY(CenterId)) -
        (PointZ(CenterId)) -
        (PointX(CenterId)+n1) -
        (PointY(CenterId)+n2) -
        (PointZ(CenterId)+n3) -
        (theta/NumOfCurves)
```

E Aligning Two Parts Input File

```
! This file requires 7 parameters
!  P1 - The part which will be moved
!  P2, P4 and P6 are identifiers of points in Part P1
!  P3, P5 and P7 are identifiers of points which will be used
!   to align Part P1. Upon completion
!  * Point P2 will be coincident with Point P3.
!  * The line defined by P2 and P4 will be aligned with the line defined
!    by P3 and P5.
!  * The plane defined by P2, P4 and P6 will be aligned with the plane
!    defined by P3, P5 and P7
! Note: file Vector.inp must be read in before this file is read in
! in order to define the Vector macros such as VectorCross.
!
! Step 1 Make Points P2 and P3 Coincide
```

```
!
Define Var Dx (PointX(P3) - PointX(P2))
Define Var Dy (PointY(P3) - PointY(P2))
Define Var Dz (PointZ(P3) - PointZ(P2))
Translate/Modify/Update Part/P1        (Dx) (Dy) (Dz)
!
! Step 2 Align the line defined by P2 and P4 with the line defined
! by P3 and P5.
!
VectorFromPoints(a, P3, P4)
VectorFromPoints(b, P3, P5)
Define Var ra VectorLength(a)
Define Var rb VectorLength(b)
Define Var theta (acos( VectorDot(a,b) / (ra*rb)))
VectorCross(n,a,b)
Rotate/Modify/Update Part/P1 -
        (PointX(P3)) (PointY(P3)) (PointZ(P3)) -
        (PointX(P3) + n1) (PointY(P3) + n2) (PointZ(P3) + n3) -
        (theta)
!
! Step 3 Align the plane defined by P2, P4, and P6 with the plane defined
! by P3, P5, and P7.
!
VectorFromPoints(n,P3,P5)
VectorUnit(n,n)
VectorFromPoints(c,P3,P6)
VectorFromPoints(d,P3,P7)
Define Var nDotc VectorDot(n,c)
Define Var cp1 (c1 - nDotc*n1)
Define Var cp2 (c2 - nDotc*n2)
Define Var cp3 (c3 - nDotc*n3)
Define Var nDotd VectorDot(n,d)
Define Var dp1 (d1 - nDotd*n1)
Define Var dp2 (d2 - nDotd*n2)
Define Var dp3 (d3 - nDotd*n3)
Define Var rcp VectorLength(cp)
Define Var rdp VectorLength(dp)
Define Var theta (acos( (cp1*dp1 + cp2*dp2 + cp3*dp3) / (rcp*rdp)))
VectorCross(n,cp,dp)
Rotate/Modify/Update Part/P1 -
        (PointX(P3)) (PointY(P3)) (PointZ(P3)) -
        (PointX(P3) + n1) (PointY(P3) + n2) (PointZ(P3) + n3) -
        (theta)
```

F Files Required for Applying Pressure Loads to Interior of a Pipe

F.1 Faces.fmt

```
Grid
    "GRID    %d %g %g %g

" 4 Identifier X Y Z
Face
    "FACE    %d %d %d %d %d %d %d %d %d %d %d
" 11 Identifier ParentId1 NumPrimaryGrids G1 G2 G3 G4 G5 G6 G7 G8
```

F.2 GenPressures.c

An example of basic C program to accomplish this task is shown below. Since this program uses fixed size arrays there are some obvious limitations on the size of the model which can be handled by this program. A more general program would use better utilities to remove the size restrictions.

```
#include <stdio.h>
#include <math.h>

#define PI            3.14159265358979323846264338327950
#define PI_OVER_2   1.57

/*
 * Define some constants for maximum allowable dimensions
 * of the grid and face data structures.
 */

#define MAX_GRID_ID 5000
#define MAX_FACE_ID 5000
#define MAX_NUM_FACES_ASSOCIATED_WITH_A_GRID 10

FILE *OutputFilePtr;

/*
 * For each grid remember the global rectangular coordinates.
 * We will build up the list of faces which reference each
 * grid.
 */
```

```
struct Grid {
    int Identifier;
    float X, Y, Z;
    int AssociatedFaces[MAX_NUM_FACES_ASSOCIATED_WITH_A_GRID]
};


/*
 * For each grid remember the basic definition.
 * We will compute the normal for each face.
 */

struct Face {
    int Identifier;
    int ParentId;
    int NumPrimaryGrids;
    int Grids[8];
    int Visited;        /* used in the recursive search routine */
    int ReversedNormal; /* used in write out routine            */
    float Normal[3];
};


/*
 * This program was written to stand on it's own and therefore
 * we use only basic data structures.  (We do not use the
 * Set tools from MSC/XL.)
 */

struct Grid GridSet[MAX_GRID_ID + 1];
struct Face FaceSet[MAX_FACE_ID + 1];

double            FlipNormal = 1.0;

main(argc, argv)
int argc;
char *argv[];
/*****************************************************************\
*
* GenPressures - read in the definition of some faces and find the
* transitive closure of all faces which the normals differ by less
* than some given angle.
*
* Optional Parameter sequence is
```

```
*   1 FileName
*   2 Identifier of the Face of interest
*   3 MaxAngle
*   4 Flip normal (y,n)
*
* Author Change Log:
* 7/16/89   DJB            Created.
*
\*****************************************************************/
{
    char            FileName[100];
    char            TempString[10];
    int             FaceOfInterest;
    float           MaxAngle;

    FILE           *FilePtr;

/*
 * If the parameters were passed in then
 *        copy them into local variables
 * otherwise
 *        prompt for them
 */

    if (argc >= 5) {
        strcpy(FileName, argv[1]);
        sscanf(argv[2], "%d", &FaceOfInterest);
        sscanf(argv[3], "%f", &MaxAngle);
        sscanf(argv[4], "%s", TempString);
    }
    else {
        printf("Enter File Name: ");
        gets(FileName);
        printf("Enter Face Id: ");
        scanf("%d", &FaceOfInterest);
        printf("Enter Max Angle: ");
        scanf("%f", &MaxAngle);
        printf("Do you want to flip the normals? ");
        scanf("%s", TempString);
    }

    if (toupper(TempString[0] == 'y'))
        FlipNormal = -1.0;
```

```
/*
 * Convert the angle to radians
 */

    MaxAngle *= PI / 180.0;

/*
 * Open the file
 */

    FilePtr = fopen(FileName, "r");
    while (FilePtr == NULL) {
        printf("Enter File Name: ");
        gets(FileName);
        FilePtr = fopen(FileName, "r");
    }

    OutputFilePtr = fopen("PLoad4.inp", "w");

/*
 * Read in the data
 */

    ReadInTheData(FilePtr);

/*
 * Compute the normals for the faces and
 * build up the list of associated faces with each grid.
 */

    FaceSetComputeValues();

/*
 * Find and process the set of faces associated with the
 * given face of interest.
 */

    FaceFindConnectedFaces(&FaceSet[FaceOfInterest], MaxAngle);

/*
 * Close the files
 */

    fclose(FilePtr);
```

```
    fclose(OutputFilePtr);
}

ReadInTheData(FilePtr)
FILE            *FilePtr;
{
    char            Name[100];
    int             Id;

    while (fscanf(FilePtr, "%s %d", Name, &Id) == 2) {
        if (strcmp(Name, "GRID") == 0) {
            GridSet[Id].Identifier = Id;
            GridSet[Id].AssociatedFaces[0] = -1;;
            fscanf(FilePtr, "%f %f %f",
                   &(GridSet[Id].X),
                   &(GridSet[Id].Y),
                   &(GridSet[Id].Z));
        }
        else if (strcmp(Name, "FACE") == 0) {
            FaceSet[Id].Identifier = Id;
            FaceSet[Id].Visited = 0;
            FaceSet[Id].ReversedNormal = 0;
            fscanf(FilePtr, "%d %d %d %d %d %d %d %d %d %d",
                   &(FaceSet[Id].ParentId),
                   &(FaceSet[Id].NumPrimaryGrids),
                   &(FaceSet[Id].Grids[0]),
                   &(FaceSet[Id].Grids[1]),
                   &(FaceSet[Id].Grids[2]),
                   &(FaceSet[Id].Grids[3]),
                   &(FaceSet[Id].Grids[4]),
                   &(FaceSet[Id].Grids[5]),
                   &(FaceSet[Id].Grids[6]),
                   &(FaceSet[Id].Grids[7]));
        }
    }
}

FaceSetComputeValues()
{
    int             i, j;
    float           v1[3], v2[3];

    for (i = 0; i <= MAX_FACE_ID; i++) {
        if (FaceSet[i].Identifier == i) {
```

```
            for (j = 0; j <= FaceSet[i].NumPrimaryGrids; j++) {
                if (FaceSet[i].Grids[j] > 0)
                    GridAddFace(&(GridSet[FaceSet[i].Grids[j]]), i);
            }

            VectorSub(v1,
                      &GridSet[FaceSet[i].Grids[1]].X,
                      &GridSet[FaceSet[i].Grids[0]].X);

            VectorSub(v2,
                      &GridSet[FaceSet[i].Grids[3]].X,
                      &GridSet[FaceSet[i].Grids[0]].X);

            VectorCross(FaceSet[i].Normal, v1, v2);
        }
    }
}

GridAddFace (GridPtr, FaceId)
struct Grid *GridPtr;
int FaceId;
{
    int             i;

    for (i = 0; i < MAX_NUM_FACES_ASSOCIATED_WITH_A_GRID; i++)
        if (GridPtr->AssociatedFaces[i] < 0) {
            GridPtr->AssociatedFaces[i] = FaceId;
            GridPtr->AssociatedFaces[i + 1] = -1;
            return;
        }
}

FaceFindConnectedFaces(FacePtr, MaxAngle)
struct Face *FacePtr;
float MaxAngle;
{
    int             i, j;
    float           Angle;
    float           FaceNormalAngle();

    struct Face             *AdjacentFacePtr;

    FacePtr->Visited += 1;
```

```
    if (FacePtr->ReversedNormal)
        fprintf(OutputFilePtr,
                "Edit/Create PLoad4/1 %d 10 10 10 10 %d %d 0 %g %g %g\n",
                FacePtr->ParentId,
                FacePtr->Grids[2],
                FacePtr->Grids[0],
                - FlipNormal * FacePtr->Normal[0],
                - FlipNormal * FacePtr->Normal[1],
                - FlipNormal * FacePtr->Normal[2]);
    else
        fprintf(OutputFilePtr,
                "Edit/Create PLoad4/1 %d 10 10 10 10 %d %d 0 %g %g %g\n",
                FacePtr->ParentId,
                FacePtr->Grids[0],
                FacePtr->Grids[2],
                FlipNormal * FacePtr->Normal[0],
                FlipNormal * FacePtr->Normal[1],
                FlipNormal * FacePtr->Normal[2]);

    for (i = 0; i < FacePtr->NumPrimaryGrids; i++) {
        if (FacePtr->Grids[i] > 0) {
            for (j = 0;
                 GridSet[FacePtr->Grids[i]].AssociatedFaces[j] > 0;
                 j++) {
                AdjacentFacePtr =
                &FaceSet[GridSet[FacePtr->Grids[i]].AssociatedFaces[j]];
                if (AdjacentFacePtr->Visited == 0) {
                    Angle = FaceNormalAngle(FacePtr, AdjacentFacePtr);
                    if (Angle <= MaxAngle) {
                        AdjacentFacePtr -> ReversedNormal =
                            FacePtr->ReversedNormal;
                        FaceFindConnectedFaces(AdjacentFacePtr, MaxAngle);
                    }
                    else if ((Angle > PI_OVER_2) &&
                             ((PI - Angle) <= MaxAngle)) {
                        AdjacentFacePtr -> ReversedNormal =
                            !(FacePtr->ReversedNormal);
                        FaceFindConnectedFaces(AdjacentFacePtr, MaxAngle);
                    }
                }
            }
        }
    }
}
```

```
float FaceNormalAngle(Face1Ptr, Face2Ptr)
struct Face *Face1Ptr, *Face2Ptr;
{
    float           DotProduct;
    float           Mag1, Mag2;
    float           Angle;

    VectorDot(&DotProduct, Face1Ptr->Normal, Face2Ptr->Normal);

    Mag1 = sqrt(fabs(Face1Ptr->Normal[0] * Face1Ptr->Normal[0] +
                     Face1Ptr->Normal[1] * Face1Ptr->Normal[1] +
                     Face1Ptr->Normal[2] * Face1Ptr->Normal[2]));

    Mag2 = sqrt(fabs(Face2Ptr->Normal[0] * Face2Ptr->Normal[0] +
                     Face2Ptr->Normal[1] * Face2Ptr->Normal[1] +
                     Face2Ptr->Normal[2] * Face2Ptr->Normal[2]));

    if ((Mag1 < 0.000001) || (Mag2 < 0.000001))
        Angle = PI_OVER_2;
    else if (DotProduct / (Mag1 * Mag2) > 1.0)
        Angle = 0.0;
    else if (DotProduct / (Mag1 * Mag2) < -1.0)
        Angle = PI;
    else
        Angle = acos(DotProduct / (Mag1 * Mag2));

    return (Angle);
}

VectorSub(A,B,C)
float A[], B[], C[];
{
    A[0] = B[0] - C[0];
    A[1] = B[1] - C[1];
    A[2] = B[2] - C[2];
}

VectorCross(A,B,C)
float A[], B[], C[];
{
    A[0] = B[1] * C[2] - B[2] * C[1];
    A[1] = B[2] * C[0] - B[0] * C[2];
    A[2] = B[0] * C[1] - B[1] * C[0];
```

```
}

VectorDot(Dot,B,C)
float *Dot;
float B[], C[];
{
    *Dot = B[0] * C[0] + B[1] * C[1] + B[2] * C[2];
}
```

G User Interface Files

G.1 ExtendUI.inp

```
Define Var CenterId 1
Define Var FromId 2
Define Var ToId 3
Define Var NumOfCurves 1
Define Macro "MidPtMenu" "Define Point MidPt( (FromId), (ToId))"
Define Macro "C2PMenu" -
    "C2P( (CenterId), (FromId), (ToId), (NumOfCurves))"
Define Macro "C2P(CenterId, FromId, ToId, NumOfCurves)" -
    "Echo Off;-
    Define Var a1 (PointX(FromId) - PointX(CenterId));-
    Define Var a2 (PointY(FromId) - PointY(CenterId));-
    Define Var a3 (PointZ(FromId) - PointZ(CenterId));-
    Define Var b1 (PointX(ToId) - PointX(CenterId));-
    Define Var b2 (PointY(ToId) - PointY(CenterId));-
    Define Var b3 (PointZ(ToId) - PointZ(CenterId));-
    Define Var ra (sqrt(a1*a1 + a2*a2 + a3*a3));-
    Define Var rb (sqrt(b1*b1 + b2*b2 + b3*b3));-
    Define Var theta (acos( (a1*b1 + a2*b2 + a3*b3) / (ra*rb)));-
    Define Var nx (a2*b3 - a3*b2);-
    Define Var ny (a3*b1 - a1*b3);-
    Define Var nz (a1*b2 - a2*b1);-
    Sweep/Points NumOfCurves Point/FromId -
        (PointX(CenterId)) (PointY(CenterId)) (PointZ(CenterId)) -
        (PointX(CenterId) + nx) -
        (PointY(CenterId) + ny) -
        (PointZ(CenterId) + nz) -
     (theta/NumOfCurves);-
Echo On"
```

G.2 QAM.txt

```
QAM
QAM
        "%s/%s"                          TASK_QAM_REFRESH -
                QAMRefresh
                CurrentRenderMode
        "Create Qual"                    TASK_QAM_POPUP CrtDef
        "Undo"                           TASK_MENU_UNDO -
        "Help"                           TASK_MENU_HELPLOADSTRING -
        "%s"                             TASK_QAM_EXIT -
                QAMExitMode
        "Spin"                           TASK_QAM_POPUP Spin
        "Group"                          TASK_MENU_DEFINEGROUP -
        "Undo"                           TASK_MENU_UNDO -
        "MidPt"                          TASK_QAM_PICK MidPtPop
        "C2P"                            TASK_QAM_PICK C2PPop
```

G.3 MidPtPop.txt

```
MidPtPop
OneLine2PickBox     "No Help"
        "MidPoint Creation"              TASK_UNSET -
        "Do It"                          TASK_MACRO_EXECUTE MidPtMenu
        "Help"                           TASK_MENU_HELPLOADSTRING -
        "Point1 -> %g"                   TASK_QAM_UNSET Point
                FromId
        "Point2 -> %g"                   TASK_QAM_UNSET Point
                ToId
```

G.4 C2PPop.txt

```
C2PPop
OneLine2PickBox     "No Help"
        "Center 2 Point"                 TASK_UNSET -
        "Do It"                          TASK_MACRO_EXECUTE C2PMenu
        "Help"                           TASK_MENU_HELPLOADSTRING -
        "Center -> %g"                   TASK_QAM_UNSET Point
                CenterId
        "Point1 -> %g"                   TASK_QAM_UNSET Point
                FromId
        "Point2 -> %g"                   TASK_QAM_UNSET Point
                ToId
```

```
"NumCrvs-> %g"                    TASK_UNSET -
        NumOfCurves
```