GridPro v5.5

User's Guide and Reference Manual

Program Development Corporation

300 Hamilton Ave. Suite 409

White Plains, NY 10601

Tel:(914)761-1732, Fax:(914)761-1735

Email:gridpro@gridpro.com

October 17, 2012

Contents

Part I

Chapters

Chapter 1

Overview

A good general purpose grid (mesh) generator should be at least good on two accounts: 1) Quick and easy to setup typical complex gridding problems: The considerations include the first gridding turn-around time, the subsequent parametric design turn-around time, the modular parametric design (adding and subtracting features) turn-around time and the clustering capabilities (for CFD use); And 2) Good grid quality: such as the grid smoothness, orthogonality, desired grid distribution and surface fidelity. Both accounts are better served through automation with different levels of user selectable controls.

For a multi-block structured grid generator, automation can be classified into four areas: 1) Optimum distribution of high quality grid, 2) Book keeping of topological information, 3) Topology generation, and 4) Surface restructuring and repair.

To this end, GridPro is a general purpose, 3-dimensional, multi-block structured grid (mesh) generator using an advanced smoothing scheme that incorporates many automatic features.

1.1 What is GridPro?

GridPro has achieved full automation in high quality grid distribution and the book keeping of topological information. It partially automates topology generation by reducing the user task to the generation a coarse wireframe of the topology in which only imprecise corner and edge information is required; while the blocks and block faces are automatically generated from the wireframe. It also has a certain capability of automatic surface restructuring and repair, such as auto-stitching of surface gaps between surface patches, and implicit surface trimming and intersection capturing.

The design of GridPro has followed the principles: 1) Minimizing the user input with a strong emphasis on topological template (`COMPONENT`) construction capability and reusability, 2) Maximizing the grid quality, and 3) Optimizing the grid distribution.

The first principle cuts down both the initial setup time and more drastically the subsequent setup time for configuration modifications; The second translates into a higher solution accuracy, and faster convergence for the CFD solvers; And the third reduces the demand for computer resources in terms of both the CPU time and the amount of RAM usage.

Once the block topology is chosen, the process of grid generation using GridPro is accomplished by solving a variationally based system with an iterative updating scheme. In this process, the initial setup of the grid is only a guess to the final grid that is the converged solution of the system. Thus, in a general sense, the final grid (solution) is independent of the initial grid distribution. This results in that only imprecise initial position information is

required and dramatically reduces the amount of required user input while generating grids with excellent quality. On the other hand, multiple sweeps are needed to generate a grid. In this sense, it is more CPU intensive.

GridPro consists of two main modules:

Figure 1.1 shows the relationship among GridPro/`Ggrid` and its different components. The next two sections are devoted to the discussion of this relationship.

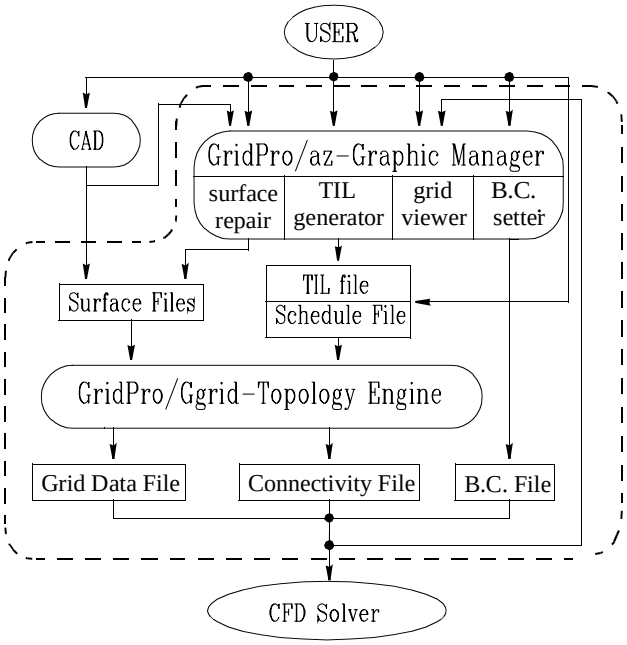


Figure 1.1: Relationship between GridPro/`Ggrid` and its environment.

1) `az`-Graphics Manager (type: '`az <ret>`' to start it).

2) `Ggrid` topology engine that reads in topology, and generates and writes out the grid. This part of the GridPro also includes a suite of other non-graphic utilities.

The media between the two modules is the Topology Input Language (TIL). The `az`-Graphics Manager is effectively a language generator that generates and feeds the TIL codes to `Ggrid` to generate grids. One can also manually write his/her own TIL codes and/or edit them without resort to the `az`-Graphics Manager.

An important point is that every TIL code is a template for the same class of problems.

This manual is about the non-graphic part of GridPro, which includes `Ggrid`, TIL and other GridPro utilities. Since the executable `Ggrid` is the center piece of this volume, we will use the terms GridPro and GridPro/`Ggrid` interchangeably, unless a real distinction is needed.

1.2 What do you put into `Ggrid` ?

For a run-case, the input required from a user has three components: surface specifications, a block topology and a run schedule.

The surface specifications constitute an external component in that they are provided mostly from the outside of the GridPro environment and conform to certain standards. The block topology is the static part of the grid generation process; Grid generation for a changed topology usually requires one to rerun GridPro. In contrast, the run schedule is the dynamic part of the

process. The run schedule should be designed to best use the computer resources and to guide the convergence. You can inspect the grid at the middle of the run and reschedule your run at any time you wish. You can also resume a previously stopped run.

In general, the grid generation process involves several iterations of modifying and enhancing the block topology to achieve the best grid quality and optimal grid distribution. The level to which the enhancements are brought is up to the user.

1.2.1 Surface specifications

Surface specifications are independent of the surface grid generated with GridPro and they can be from different sources and in different formats. The collection of implemented formats for GridPro is still expanding.

However, there are also some particular requirements on the surface conditions and formats. An important requirement is the smoothness of the surface defined. GridPro can handle up to 90^o jumps for the surface normal vector. Such condition can break down on the seam lines of many surfaces generated directly from popular CAD systems.

When necessary, surface geometries should to be restructured to conform to the requirements of GridPro before using them. Such restructuring can be as simple as changing the data format; or as complicated as merging surfaces, and smoothing, modifying and removing small features.

Within the GridPro software package, there are utilities and tools to assist users to create and restructure surfaces. The GridPro/`az`-Graphic Manager can also be used to accomplish these.

1.2.2 Block topology

The phrase *topology* here is defined as the connectivity information of block corners (not blocks!), the surface assignments of corners (and possibly edges and faces) and the initial positions of corners. A special topology input language (TIL) is used to record the topology into files. The topology files must have the file name extension '.fra'.

It is the user's responsibility to design and record a block topology. The design and recording process can be done either manually coding in TIL or using `az`-Graphic Manager. However, to record the design, a user does not need to provide the information about the edges, faces and blocks in the design. GridPro will generate such information automatically! In simple terms, a TIL file contains mainly a sequence of corner definitions, each of which provides an approximate initial position of the current corner, a list of other corners that has a link to the current corner, and a list of surfaces that the corner should be on.

An important feature of TIL is to organize the topology design into COMPONENTs. A COMPONENT works much the same way as a subroutine in, say, FORTRAN. It hides the irrelevant details of the topology and connects to the rest of the topology through interfacial variables. This feature of TIL provides a natural means to build reusable component libraries.

1.2.3 Run schedule

Since GridPro uses an advanced smoothing scheme in which the grid is generated in multiple sweeps just as in the case of the ordinary elliptic grid generation, a schedule for the run must be provided for a better and faster convergence.

A schedule file consists of step lines, each of which lists a sequence of actions that direct the run process of GridPro. The name of a schedule file must have the prefix part same as the corresponding main topology file and end with the file name extension '.sch'.

1.3 What do you get out from `Ggrid` ?

GridPro outputs the block connectivity information and grid data in various formats.

1.3.1 Block grid data

Block grid data is in simple point data format listed block by block.

1.3.2 Block connectivity data

Block connectivity data is written into the file 'conn.tmp'.

1.4 A complete and simple example

Our first example is to generate a grid for a simple 2-d case. The region to be gridded is the area between the circle and the rectangular box as shown in Figure 1.2(a). Through this

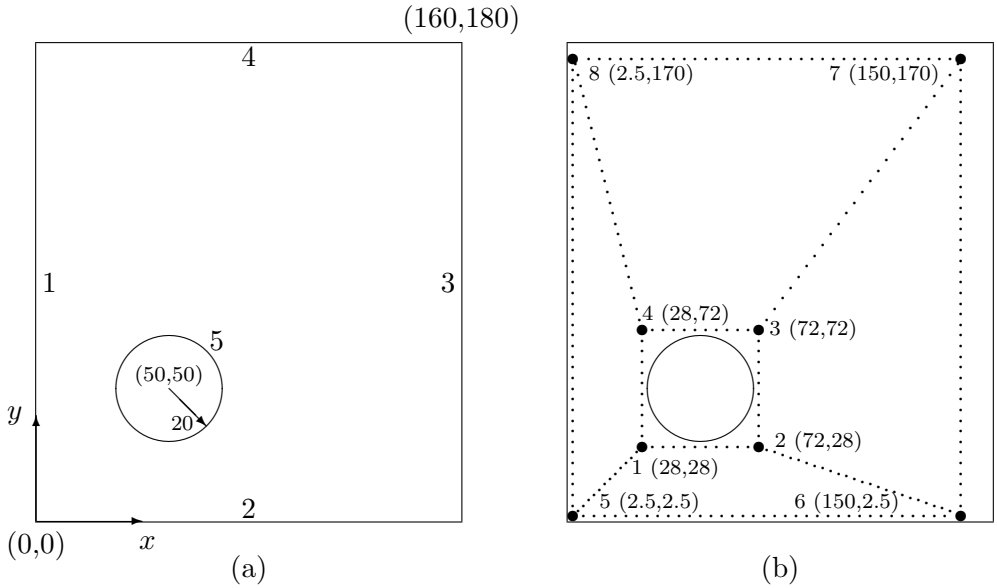


Figure 1.2: (a) A simple region to be gridded. (b) The block topology designed by a user.

example, we demonstrate what the basic steps are for generating a grid using GridPro. Also in this and other examples of this manual, we will focus on the details of coding in TIL; Therefore we will not use `az`-Graphic Manager to prepare the TIL code though it is much simpler to do so for simple cases like this. The advantage of manually coding in TIL becomes obvious when one deals with more complex cases with topologically repetitive structures, or design iterations are required.

1.4.0 Step 0 – Preparing surfaces

For the example we are going to demonstrate in this section, all the surfaces used are built-in implicit analytic surfaces which are simple. For this reason, the task of current step is significantly reduced.

Generally, surfaces need to be prepared to conform to the requirements of GridPro. For instance, GridPro requires a certain degree of smoothness for the surfaces. This sort of requirements and the surface data format requirements forms a tedious, but more or less independent part of GridPro; These, being general geometric items, are much less particular to GridPro. Therefore, a general discussion of this step is left to Chapter 7.

1.4.1 Step 1 – Partitioning and Labelling geometry

The first thing we do is to partition the geometry into surfaces and label the resulting surfaces. The word surface has a specific meaning for GridPro. A surface here is defined as a portion of the geometry on which the distribution of grid points is automatically created by GridPro. It is up to the user to decide how the geometry should be partitioned into different surfaces. Though there is no unique way to perform the division, for most cases there are one or two natural ways to do so. The way the geometry should be partitioned also strongly depends on the block topology in use and the existing geometric features. The surfaces should form a leakless closure of the gridding region. The gridding region must be connected.

For our case, we can consider that there are five surfaces in the problem. The circle is one, and the four sides of the box are the other four surfaces. The surface labels we assigned are marked in Figure 1.2(a) with 1, 2, 3, 4 and 5. In the assignment, each surface should have a unique number to identify it and the order of the numbering is not important. We will also use the phrase *surface id* to mean the number assigned to a surface.

In the above surface division, grid points intended for, say, surface 1 will not be distributed into surface 2. Another choice for the division is to regard surfaces 1, 2, 3 and 4 as one surface and assign it a single label. In this case, GridPro will consider the four sides of the box as a whole in order to decide the distribution of grid points on it.

So far, we have not said anything about the surface data formats and requirements. As said earlier, we leave the details to Chapter 7, except to mention here that as a rule, each of the surfaces should be relatively smooth. The intersections of different surfaces do not need to be explicitly defined; but, generally, the specifications of the surfaces should extend somewhat beyond the intersections. Note also that only the surface portion that is a part of the closure of the gridding region is really used. We will not have this problem here, however, since all the surfaces can be specified analytically in very simple terms as can be seen in the next step.

1.4.2 Step 2 – Designing a block topology with TIL

The second thing we do is to design a block topology for the region to be gridded. This can be done with the GridPro/az-Graphic Manager or simply using a pen on a piece of sketch paper. For the purpose of learning TIL, we will go the later route.

The process of designing the topology is the fun part of the whole grid generation process. It also needs some creativity.

At a simple level, the goal is to cover the region with quadrilaterals (for 2d cases). This covering does not need to be done at a geometric precise level. It needs to be done only at a rather topological level; That is, a surface can be represented as a set of piece-wise linear segments (again for 2d cases) placed not too far from the real surface. Let's take a circle as an example. It can be represented by a square, a pentagon, or an arbitrary polygon, all depending on the needs of your block design. The design will be programmed with the Topology Input Language (TIL) into a file to be compiled and processed by GridPro.

For a configuration of complex geometry, one can design simple components of block topologies, and assemble them into a complex one much the same way as a real airplane or

automobile is built.

For our case, the designed block topology is shown as the dotted lines and big dots in Figure 1.2(b). The solid lines are surfaces. The big dots represent the block corners and the dotted lines connecting the big dots are corner links. We also labelled the block corners from 1 to 8 with its approximate coordinates written next to it in the parentheses. The term *corner label* is also referred as *corner id*. The TIL program for this topology design is written in a file called 'example1.fra' and listed in Program 1.1.

Program 1.1 *File 'example1.fra'*

```
SET DIMENSION 2
SET GRIDDEN   16

COMPONENT circleInBox()
BEGIN
  s 1   -plane  ( 1.0    0  0     0) ; #x1 side
  s 2   -plane  (   0  1.0  0     0) ; #y1 side
  s 3   -plane  (-1.0    0  0 160.0) ; #x2 side
  s 4   -plane  (   0 -1.0  0 180.0) ; #y2 side
  s 5   -ellip  (0.05 0.05 0) -t 50.0 50 0 ; #circle

  c 1   28  28  0  -s 5 ;
  c 2   72  28  0  -s 5    -L 1 ;
  c 3   72  72  0  -s 5    -L 2 ;
  c 4   28  72  0  -s 5    -L 3 1 ;
  c 5   2.5 2.5 0  -s 1 2  -L 1 ;
  c 6   150 2.5 0  -s 2 3  -L 2 5 ;
  c 7   150 170 0  -s 3 4  -L 3 6 ;
  c 8   2.5 170 0  -s 4 1  -L 4 7 5 ;

  g 1 5  32;
END
```

To have an overview of a complete example, only a brief explanation is given here for Program 1.1. A detailed explanation is left to Chapter 2 to 4.

Program 1.1 tells GridPro that the topology is a 2d case (`SET DIMENSION 2`); Edges are initialized to have 16 grid points (`SET GRIDDEN 16`); And the topology consists of one component (`COMPONENT circleInBox()`) in which five surfaces and eight corners are defined. A corner is defined by its initial position, the surfaces it should be on (with the `-s` flag) and the corners it has link to (with the `-L` flag). Note: the coordinates must be specified in 3-d fashion for both 2d and 3d cases.

The line starting with the key word 'g' assigns the edge connecting corners 1 and 5 with 32 grid cells.

Any thing beyond the '#' character in a line is ignored by GridPro.

1.4.3 Step 3 – Scheduling your run

Our run schedule must be in the file 'example1.sch'.

To be simple, let's say we want to generate a grid in 100 sweeps and write out the 2d grid to a file called 'blk2d.tmp'. 'example1.sch' will have only two lines as follows:

Program 1.2 *File 'example1.sch'.*

```
step 1: -S 100 -w

write -f blk2d.tmp
```

The schedule section of the file has only one step with two actions. The actions are executed one by one from left to right. The first action '`-S 100`' tells GridPro to run 100 relaxation sweeps; The second, '`-w`' directs GridPro to execute an output grid action.

The details about the data to be outputted is specified in the output section of the same schedule file. This section consists of all the lines beginning with the key word `write`. For our case, it has also only one line, which tells GridPro to write out the grid to a file called 'blk2d.tmp' (`-f blk2d.tmp`).

1.4.4 Step 4 – Generating a grid

Now, we are almost ready to run GridPro. Since some '.tmp' files will be generated automatically in the current directory when running GridPro, it is always a good idea to create a directory for each run case, and place your '.fra', '.sch' and other relevant files in it. You should run GridPro in it too.

GridPro is normally installed in the directory '*SOMEWHERE*/GridPro'. Before running it, make sure the path has '*SOMEWHERE*/GridPro/bin' in it. You can check the path by typing,

```
set | grep path <ret>
```

If it is not set, set it by appending to '~/.cshrc' a line,

```
set path = ( $path SOMEWHERE/GridPro/bin )
```

To generate a grid, type,

```
Ggrid example1.fra<ret>
```

GridPro will read 'example1.fra' once to generate all topology information and schedule the run according to 'example1.sch'. When it finishes, the grid generated is stored in the file, 'blk2d.tmp'.

The data is in a simple point data format. That is, the data is listed block by block starting from block 1. For each block the data can be read from a FORTRAN program as follows:

Program 1.3 *Point data format in FORTRAN*

```
READ(UNIT,*) IMAX,JMAX,KMAX
DO 10 I=1,IMAX
DO 10 J=1,JMAX
DO 10 K=1,KMAX
10  READ(UNIT,*) X(I,J,K), Y(I,J,K), Z(I,J,K)
```

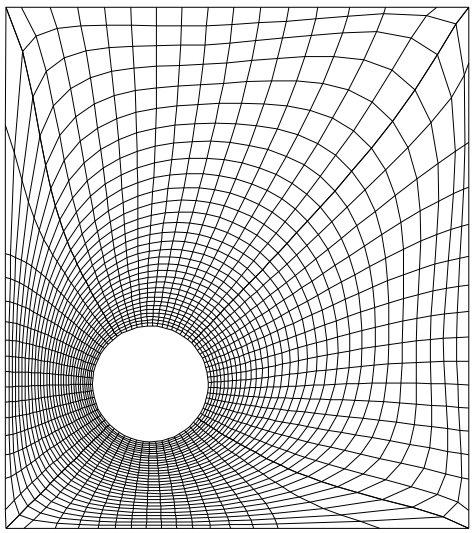


Figure 1.3: A grid generated from Program 1.1.

where `X, Y, Z` are the x, y, and z coordinates of a grid point.

The connectivity information of the blocks is automatically stored in the file 'blk2d.tmp.conn'. For the format of it please read Section 5.3.2.

Now you can examine the grid using the `az`-Graphic Manager by typing,

```
az -v blk2d.tmp <ret>
```

For the details of operating `az`, see the manual volume for GridPro GUI and the on-line help in `az`.

The grid is also shown in Figure 1.3.

So far, we have finished one cycle of the grid generation process. It is likely enough for a simple problem like this. However for a complex problem, the user probably has to go through several cycles of retopologizing and rescheduling to generate a grid to one's liking.

1.5 Organization of this manual

The rest of the manual is arranged as follows: In Chapter 2 , the Topology Input Language (TIL) is introduced and explained through the simple example given in Chapter 1.These are useful for parameterization of the topology. Chapter 3 provides the general procedure of running GridPro. Chapter 4 is devoted to the surface specifications implemented in the current version of GridPro. Chapter 5 is focused on the hints, tips and tricks for generating better grids using GridPro. Utilities and tools to analyze, extract and convert data and grids are discussed in Chapter 6. These include the plugin utilities of GridPro/`Ggrid`. Chapter 7 is a brief discussion of the Property And Boundary Condition Assingnments of Grid. Chapter 8 is a brief discussion of the `az`-Graphic Manager. The details are left to the GridPro GUI manaul and online help of `az`.

Appendix A is scheduling syntax and worked out examples.

Chapter 2

Basics of Topology Input Language (TIL)

Minimum or no knowledge of TIL is required for people using the az-Graphic Manager only to create topology. However, programming with TIL become increasingly important for complex geometries with repetitive sub-topologies, or when design optimization for the geometry is in consideration. In this Chapter, we will use Program 1.1 to illustrate the basics (the minimum knowledge required) of Topology Input Language (TIL).

2.1 TIL program structure

When GridPro processes a TIL program, anything from a '#' character to the end of the line is ignored. A '#' character can be used to introduce comments for that line.

New-line characters and tabs are treated as spaces and consecutive spaces will be truncated to a single space. Thus, the alignment in Program 1.1 is purely for styling purposes. In writing a TIL program, spaces can often be omitted as long as tokens can be read in correctly.

A TIL program can have three sections appearing in the following order:

1) An optional global assignment section.
2) An optional include section.
3) A component definition section.

Program 1.1 has only sections 1) and 3). The missing section 2) is normally used where the topology design is programmed in several files or topology libraries. For the case where the entire topology is contained in a single file, there should not be an include section.

The optional assignment section is used to assign certain global parameters. A parameter not assigned in the assignment section takes a default value.

The first two lines of Program 1.1 form the assignment section,

```
SET DIMENSION 2
SET GRIDDEN   16
```

It specifies the problem to be a 2d case and initializes every edge to have 16 grid cells. The parameter `DIMENSION` can have a value either 2 or 3. The parameter `GRIDDEN` must have a value greater or equal to 3. Without these assignments, `DIMENSION` and `GRIDDEN` take the default values 3 and 8, respectively.

The 'SET GRIDDEN' line can have another syntax as follows:

```
SET GRIDDEN  E_X_axis 16 E_Y_axis 12 CROSS 10
```

Multiple such 'SET GRIDDEN' lines can be used to initialize grid density on edges. Here, E_X_axis, E_Y_axis and CROSS are global edge labels defined in the TIL program, that each may be a collection of more than one edges. A 'SET GRIDDEN' line like this functions as a schedule step before those steps in the .sch file.

GridPro is a general purpose 3d software package. For a 2d case, GridPro will first convert it to a 3d case, then run it as a 3d problem. Thus, for a 2d case, anything in the topology file involving real space positions still has to be specified in a 3d fashion. However, the z coordinate should always be assigned a value, 0. The places where real space positions are involved can be the initial positions of corners, the data to specify surfaces, and possible translation and rotation operators for surfaces and components. We will see it when we go through the details of Program 1.1.

The component definition section is the core of a TIL program. The basic unit of topology specifications in TIL is a COMPONENT and a TIL program consists of at least one COMPONENT. A COMPONENT is composed by a set of declarations and statements bounded by the key words BEGIN and END as follows,

```
COMPONENT comp_name( arg_list )
BEGIN
   declaration
   .
   .
   declaration
   statement
   .
   .
   statement
END
```

Each declaration or statement starts with a key word and ends with the terminal symbol ';'. The first COMPONENT is always the head COMPONENT which does not require any arguments to be passed in and out. GridPro will construct the complete block topology from the head COMPONENT.

For our case, the entire topology design is specified in a single component named circleInBox. The five statements starting with an s define the five surfaces. The eight statements beginning with a key c define the eight topology corners shown in Figure 2.1(b). The last statement assigns a grid density for an edge.

2.2 Defining surfaces

A surface can be in one of three modes depending on how they are used. A surface of the fixed mode is fixed in space by the data specifying the surface; A surface of the periodic mode is used for periodic boundary conditions and is not fixed in space by the data specifying the surface. A surface of the float mode has no fixed position in the space. For most cases, surfaces

are defined in the fixed mode. A fixed surface can be either internal or external depending on whether both sides or only one side of the surface need to be gridded.

Before we proceed further, let us make a distinction between the phrases surface specification and surface definition used in this manual. By a surface specification, we mean the data and data format used to describe the shape of a surface. On the other hand, by a surface definition, we mean a statement in the TIL program which starts with a key word '`s`' and assigns an id and other attributes to a surface. The main function of a surface definition is to make the surface known to other parts of the TIL program.

2.2.1 Surfaces of the fixed mode

All the five surfaces used in Program 1.1 are in the fixed mode. They are also all used as external surfaces. The term external here simply means that the grid region is on one side of the surface, as opposed to an internal surface where blocks must appear on both sides of the surface.

External surfaces

Among the five surfaces, the first four are of type `-plane`; The fifth is of type `-ellip` (for ellipsoid).

There are two groups of surface types used in GridPro. The first group is the explicit surfaces. A surface of this group is usually specified by a fair amount of data stored in a separate file(s). The second group of types are implicit. In this case, a surface is defined as an equal potential surface of a scalar valued analytic function of position vector. Some of the simple forms of the functions are hard wired into GridPro. They are called built-in implicit types. For these types, a surface is specified by providing several parameters in the corresponding surface definition statement in the TIL program. There is no need for a separate specification data file. Both types, `-plane` and `-ellip` are built-in implicit types. (For a complete list of types accepted by GridPro see Chapter 7.)

For a surface of type `-plane`, the four real numbers enclosed in the parentheses (a b c d) specify the plane in such a way that a point on the plane satisfies the equation $ax+by+cz+d=0$ and the plane normal vector (a, b, c) should point into the region to be gridded. To be more specific, let us look at the statement defining surface 3,

```
s 3  -plane(-1.0  0  0  160);
```

This is the right side of the box (Figure 2.1(a)) with the normal vector pointing opposite to the x axis. Therefore the equation for it is $-x + 160 = 0$ and the parameters for the surface appear as `(-1.0  0  0  160)`.

For a surface of type `-ellip`, three real numbers enclosed in the parentheses (a b c) are provided. A point on the ellipsoid satisfies the equation $(ax)^2+(by)^2+(cz)^2-1=0$. For a circle of radius 20, we have $a = 0.05$, $b = 0.05$, and $c = 0$. The center of the circle is translated by the `-t` *translation_vector* operation with *translation_vector* = 50.0 50.0 0. Altogether this appears as,

```
s 5 -ellip(0.05 0.05 0) -t 50.0 50 0 ;
```

Note that the `-t` *translation_vector* operation can be used for any type of surface. In fact, general transformations can be applied to surfaces. And a general transformation can be specified using vector expressions (See Chapter 5).

As we mentioned above for plane surfaces, the surface normal must point into the region to be gridded. This is generally required for all types of external fixed surfaces. If the surface definition is in a wrong orientation, a `-o` flag can be placed in the corresponding surface definition statement to reverse the orientation. Normally, the place to put it is next to the type (and associated parameters or data) flag. For example, the following statement defines the same surface 3 as before,

```
s 3  -plane(1.0  0  0  -160)  -o;
```

Another useful attribute that can be associated with a surface is a targeted average off-wall normal grid spacing. The line,

```
s 3  -plane(1.0  0  0  -160)  -o -c 0.00001;
```

assigns surface 3 a spacing of 0.00001. It means that grid points near surface 3 are intended to be clustered toward surface 3 with a target first layer grid normal spacing 0.00001.

Two things you need to keep in mind. First, the clustering will not take effect until it is turned on in the run schedule. Turning it on or off can be scheduled at any step in the schedule file. Second, turning the clustering on does not always mean the targeted spacing will be reached. GridPro tries to limit the grid growth ratio of the length scales of two consecutive grid cells not larger than 3.

Internal surfaces

An internal surface is a surface for which both sides of the surface are to be gridded. An internal surface can be specified with any surface type that can be used for an external surface. However, the orientation of the surface must be suppressed with the flag **-O** in the surface definition statement. The grids on both sides of an internal surface will be matched on the surface. A typical example is the wake of flow over an airfoil. To obtain a high grid quality, it is desirable to have internal surfaces relatively flat.

2.2.2 Surfaces of the periodic mode

Though periodic boundary conditions are not used in the example programs, it is basic enough to render a discussion here.

A surface of the periodic mode can only be of implicit type (`-implic`, *-xpolar* or `-xyz`. The same `s` statement syntax defines the surface. However, the data used to specify the surface is different.

In this case, a surface specification really specifies a family of infinitely many surfaces and one of them will become the final surface determined by many other factors. For details see, Section 2.5 and 7.4.

2.2.3 Surfaces of the float mode

It can be used to specify spacings for block interfaces. The only valid surface type for this mode is `-float`.

2.3 Defining corners

The eight statements beginning with a key `c` in Program 1.1 define the eight topology corners shown in Figure 2.1(b). For each of the `c`-statements, the number next to the key `c` is the label we assigned to the corner; The next three numbers give an approximate initial position (x, y, z) of the corner. Two more pieces of information for defining a corner are the fixed boundary conditions of the corner and the linkages to other corners.

By a boundary condition, we simply mean a logical association or assignment of a corner, edge or face to a certain surface. GridPro uses the associations to determine the automatic distribution of grid points on corresponding surfaces.

The boundary conditions of the fixed mode for a corner are specified through a list of surface labels following `-s` and the linkages are specified by a list of corner labels following `-L`. All referenced corners or surfaces following the `-s` flag and `-L` flag should be defined before the current statement. For example, consider the statement,

```
c  6     150  2.5  0    -s  2  3    -L  2  5;
```

This says that corner 6 is on surface 2 and 3 in the final grid (`-s 2 3`), and has links to corner 2 and 5 (`-L 2 5`) . Notice, in Figure 2.1(b) corner 6 has also a link to corner 7, however by the rule, only those corners defined before corner 6 will be listed for corner 6. Thus the link from corner 6 to corner 7 will only appear in the link list for corner 7.

The initial position of corner 6 is at (150, 2.5, 0) which is not and does not need to be on surface 2 and 3. This is another point we would like to make: The surface assignments for a corner is meant for the final grid; the initial position of a corner does not have to be placed on the surfaces assigned to the corner. In fact, the initial positions of corners can be specified at a very imprecise level and, in theory, the final grid is independent of the initial positions. On the other hand, the initial positions can not be entirely arbitrary in order to have a convergent final grid.

A good rule of thumb is to put the links intended for a surface on the outside of the surface if it is a closed surface, and on the convex side of the surface if it is not, and more carefully place the corners that are near the high curvature region.

2.4 Assigning grid densities

Without an explicit assignment of grid density, an edge is assigned the default value of 8 cells. There are four means to change the assignment. They are listed with increasing priorities as follows,

1) Global assignment without edge labels: We have discussed it earlier in Section 2.1. An example is a line such as,

```
SET   GRIDDEN   3
```

in the header of the main TIL program file.

2) Local static assignment: It is done through the `g`–statements in the components of a TIL program. Assume we have the following line in component `circleInBox` of Program 1.1,

```
g   1 5 32    1 2 25    1 3 30    6 5 20    2 3 1;
```

Here each triplet of numbers following the key word, `g`, defines an edge grid density assignment. Let us use Edge(1,5) to mean the edge connecting corners 1 and 5. Thus, 1 5 32

means assigning 32 grid points to Edge(1,5). This assignment will propagate through all the parallel edges, that is Edge(2,6), Edge(3,7) and Edge(4,8) in this case.

Three rules apply here, a) All the corners referenced (i.e. corners 1, 2, 3, 5 and 6 in this case) must be defined before this statement. Otherwise, it is a syntax error. b) If an edge referenced does not exist(e.g. Edge(1,3) here) or the grid density is less than 3 (e.g. in 2 3 1), the triplet is ignored. c) If an edge group is affected more than once, the assignment with more grid points takes precedence (e.g. 1 2 25 takes precedence over 6 5 20). This last rule also applies over different `g`–statements in different components.

3) Global static assignments with edge labels: Act as if it is the first schedule step. Again, see Section 2.1 for an example.

4) Dynamic assignment: Grid density can also be dynamically changed in a similar fashion as for static assignments. It is done through `-g` actions in the schedule file. Changes can be scheduled to happen at different steps. For details, see Appendix A.

2.5 Periodic boundary conditions

`i` statements are used to define periodic boundary conditions. A periodic boundary is set by identifying corner pairs through a period with respect to a periodic surface.

Note that: A corner on a periodic surface should NOT be assigned to the surface through the '`-s`' flag in the corner definition statement!

Consider a simplified example of turbo blade cascades in 2d shown in Figure 2.1(a). Figure 2.1(b) is the topology design for one of the blades. The solid lines are surfaces in the fixed mode. The two dashed lines here are surfaces in the periodic mode. They both have the same surface label, 4, since they are periodic to each other with respect to surface 4.

Then, what do we mean by 'periodic with respect to a surface'? Let's explain it with the example in mind.

First, on the two dashed lines we must have the same number of corners and every corner on one dashed line is paired with one and only one corner on the other dashed line. The neighbouring relationship of corners on one dashed line must be preserved through the pairing process (that is, the two sides have the same topology). The pairing of two corners is also called an identification of the corners since the two corners are identical in the sense that a corner on the lower dashed line of blade 2 is the paired corner on the upper dashed line of blade 1. With a correct identification between corners on the two dashed lines, all the grid points on the two dashed lines are automatically identified (or paired). In our case, the identification is: corner 1 to corner 13, corner 2 to corner 14, corner 3 to corner 15, and corner 4 to corner 16.

Second, a surface specified in the periodic mode defines a coordinate transformation from the physical space (x, y, z) to some working space, say, (u, v, w), such that in the new coordinate system (u, v, w), two corners having an identifying relation have the same values for v and w, and a given and fixed difference δu in u.

Third, it is the user's responsibility to choose the coordinate transformation for which certain consistency must be maintained. In terms of the new system, any surfaces intersecting both dashed lines (surfaces 2 or 3) must be periodic in the same fashion. That is, in the (u, v, w) space and using the above example, if a point $(u0, v0, w0)$ is on surface 2 and near the lower-left intersection between surface 2 and surface 4 where the final grid is expected, the point $(u0 + \delta u, v0, w0)$ must be on the surface that intersects surface 4 on the upper left-intersection, which, in this case, happens to be the same surface 2.

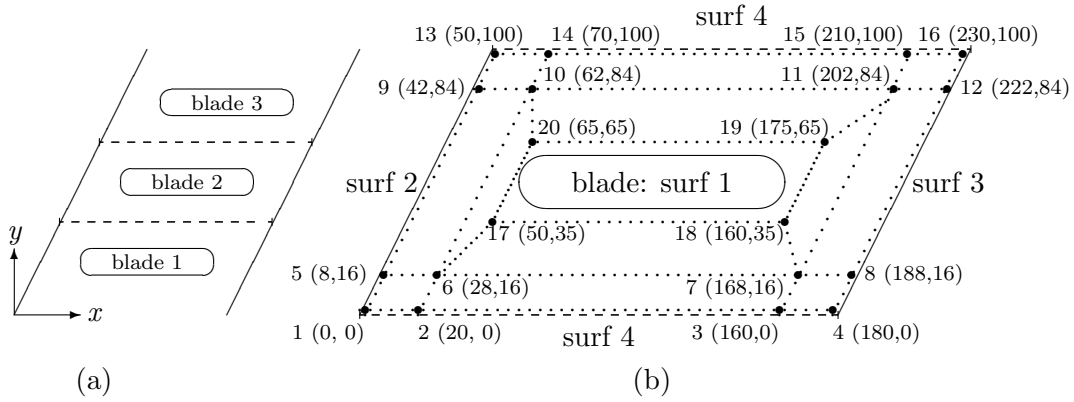


Figure 2.1: A turbo cascade in 2d.

In our case, the choice of coordinate transformation used for surface 4 is

$$u = x + 2y$$

$$v = 2x - y$$

$$w = z$$

Surface 2 and 3 are,

$$2x - y = 0 \qquad (\ v = 0\ )$$

and

$$-2x + y + 360 = 0 \qquad (\ -v + 360 = 0\ )$$

The TIL program for this topology is listed below,

```
COMPONENT blade()
BEGIN
  s 1 -linear "blade.dat";
  s 2 -plane ( 2 -1 0 0);
  s 3 -plane (-2  1 0 360);
  s 4 -implic "periodSurf.h" 250.0; # define the coordinate
                                    # transformation

  c 1    0   0 0 -s 2 ;
  c 2   20   0 0      -L  1;
  c 3  160   0 0      -L  2;
  c 4  180   0 0 -s 3 -L  3;

  c 5    8  16 0 -s 2 -L  1;
  c 6   28  16 0      -L  2  5;
  c 7  168  16 0      -L  3  6;
  c 8  188  16 0 -s 3 -L  4  7;

  c 9   42  84 0 -s 2 -L  5;
  c 10  62  84 0      -L  6  9;
  c 11 202  84 0      -L  7 10;
```

```
  c 12 222  84 0 -s 3 -L  8 11;

  c 13  50 100 0 -s 2 -L  9;
  c 14  70 100 0      -L 10 13;
  c 15 210 100 0      -L 11 14;
  c 16 230 100 0 -s 3 -L 12 15;

  c 17  50 35  0 -s 1 -L  6;
  c 18 100 35  0 -s 1 -L  7 17;
  c 19 175 65  0 -s 1 -L 11 18;
  c 20  65 65  0 -s 1 -L 10 19 17;

  i  (1 13  4);
  i  (2 14  4);
  i  (3 15  4);
  i  (4 16  4);
END
```

The last four statements set up the periodic boundary condition for the two dashed lines. An `i`-statement has the following syntax:

`i` (*cid1 cid2 sid*);

Here *cid1* and *cid2* are two corner ids, *sid* is a surface id. The statement identifies corner *cid2* with corner *cid1* through surface *sid*. Here, one has to make sure that surface *sid* is one that can be used for a periodic BC. In other words, it is one of the types `-xpolar`, `-xyz`, or `-implic`. And remember that not every `-implic` surface can be used for a periodic BC. More precisely, for the final grid in the new coordinate system (u, v, w) defined in surface *sid*, the periodic boundary condition requires,

$$(u_2, v_2, w_2) \text{ - } (u_1, v_1, w_1) = (period,0,0)$$

where (u_1, v_1, w_1) and (u_2, v_2, w_2) are for corners *cid1* and *cid2* respectively. Notice, the period, which is given when the surface is defined, is specified in terms of the u coordinate of the new system. The sign of *period* is irrelevant, since internally the sign is recalculated from the initial positions of relevant grid points. See Chapter 7, for the details of constructing an '.h' file for a periodic surface.

2.6 Topology building rules

In designing a block topology, one needs to keep in mind the automatic rules that GridPro uses in the topology construction, then, follow the user rules to build a valid topology.

2.6.1 Automatic rules

Automatic rules are rules that GridPro uses to build the final topology from the topology input program (TIL code).

1) *The rule of object building:* Unless explicitly overruled, a link forms an edge, a closed 4-edge loop forms a face (or a block for 2d cases) and six closed faces form a block.

This rule may generate false faces or blocks. GridPro provides a means to correct them either automatically or manually. In the case of Program 1.1, since the quadrilaterals defined by corners 1, 2, 3 and 4, and by corners 5, 6, 7 and 8 form two 4-edge loops, two blocks will be generated. However, they are not intended to be blocks. Therefore, they can be explicitly excluded with the `x`-statement such as,

```
x f  1 3  5 7 ;
```

where a quadrilateral is referenced by a pair of its diagonal corner labels. More discussion on statements like this appears in Chapter 4.

These loops will be also automatically excluded from forming blocks as long as the surfaces are all correctly assigned to corners.

2) *The rule of surface assignments:* Unless explicitly overruled, the surface assignments of an edge (face) are derived from the surface assignments of its two (four) boundary corners (edges) with the possible modifications by the next two rules.

Again, in Program 1.1, this rule sets the surface assignments for all edges. GridPro will determine that the edge connecting corners 7 and 8 is on surface 4 since both corners 7 and 8 are on surface 4; Further, since this edge is the only edge on surface 4, GridPro will automatically distribute the grid points of this edge onto the part of surface 4 bounded by the intersections of surface 4 with surface 1 and surface 3. By the same token, the edges defined by corners 1 and 2, corners 2 and 3, corners 3 and 4, and corners 4 and 1 are all on surface 5; GridPro will automatically distribute the grid points on all of these edges as a whole on the circle – surface 5.

3) *The rule of overlapping surfaces:* When a face is assigned either automatically or manually to two or more fixed surfaces, these surfaces are overlapping surfaces. The data specifying these overlapping surfaces must actually overlap each other over the area where the face may locate. In the part of topology that has surface overlaps, they are considered as one surface, thus, the intersections of the overlapping surfaces will not be sought.

Surface overlap is used for avoiding surface confusion for some tricky surfaces. In those cases, one has a pre-knowledge as to where a face of concern will or will not go in the final grid generated by GridPro, and by some reason, the face did not attach to the correct part of the surface. One can divide the surface of concern into several surfaces, and overlap them on the area where no confusion is likely. Note that, the concept of surface overlap here is not merely a physical space overlap of surfaces; It also attaches topological requirements. Two surfaces overlap only when at least one face is located on both of them. Note also, that overlap is only a local concept, in that, two surfaces may overlap in one region and intersect in another.

4) *The rule of reduction of surface assignments:* It is an over supply of surfaces if a corner is assigned to more than 3 surfaces, or a edge is assigned to more than 2 surfaces, or a face is assigned to more than 1 surface. GridPro will make a proper reduction of such over assignments. To generate a good grid, it is required that the surfaces of every possible reduction produce the same intersection. For example, if an edge is assigned to 3 surfaces, the 3 intersection curves generated from 3 possible selections of 2 surfaces out of 3 should be the same. If a corner is assigned to 4 surfaces, the 4 surfaces should intersect at one point. One should note that these requirements are not generally satisfied since in general 3 surfaces do not intersect at one point and 4 or more surfaces do not have intersection. Therefore, special care must be taken to insure the requirements are satisfied.

This rule can be overruled by using exclude(`x`), and add(`a`) -statements to eliminate manually the over assignments for the relevant corners, edges and faces.

2.6.2 Rules of valid topology

The rules of valid topology are the rules that the user must follow to design a valid topology. In order to follow these rules, one must first understand the automatic rules used in GridPro (see previous subsection).

A topology that GridPro can successfully parse and compile is a valid topology. GridPro accepts very general topology input. However, GridPro may reject certain seemingly valid topologies for various reasons.

1) *The rule of irreducibility:* With a valid topology, the grid region must be decomposed into full face matching blocks without dangling, unused, or redundant corners, edges or faces.

2) *The rule of connectedness:* All parts of a topology must be connected. That is, a valid topology is one that can not be divided into disconnected parts. If a topology can be divided into disconnected parts, it usually means that the region to be gridded is composed of disconnected subregions. In this case, one should grid each of the subregions independently.

3) *The rule of singularity:* GridPro will reject any topology that has an edge that has more than 8 blocks to it. This rule stems from the grid quality considerations.

4) *The rule of surface closure:* In the topology, the faces assigned to external surfaces must form a full closure of the grid region. That is, these faces should fully separate the region to be gridded from the rest of the world. If internal surfaces are involved, the faces on non-float surfaces is better to cut fully the grid region into disjoint subregions.

These rules are the minimum requirements for a valid topology. However, a valid topology does not always guarantee a grid, let alone a good grid, since the rules here concern only the topological acceptability of the blocking. Many other factors affect whether a grid will be generated. A class of these factors concerns the physical space properties of the involved surfaces and initial corner positions. In the following, we give a few typical situations where a valid topology may lead to bad grids or no grids at all.

1) *Incompatible topology*: The block topology is not suited for the physical shape of the region to be gridded.

2) *Bad initial corner positions*: The initial positions of corners can be generous, but should not be too wild.

3) *Incompatible surfaces*: The physical space properties of surfaces must be compatible with topological requirements. For example, if the topology requires two surfaces intersect, they must intersect truly in the real space. A true intersection of two surfaces is a curve and at every point on the curve, the two normals of the two surfaces are distinct.

4) *Bad internal surfaces*: A bad internal surface can mean either that the surface has too much non-uniformity of curvature, or that the neighbourhoods on the two sides of the surface are too different.

Chapter 3

Running GridPro

3.1 Run options

Before running GridPro, make sure GridPro is properly installed (see Appendix D for installation). Assume that GridPro is installed in the directory `/usr/local/GridPro`.

In particular, your path should contain `/usr/local/GridPro/bin`. You can check the setting by typing,

```
set | grep path <ret>
```

and

If the path is not set properly, set it by adding the following line to your `~/.cshrc` file,

```
set path = ( $path /usr/local/GridPro/bin )
```

GridPro can be invoked through the commands, `Ggrid` and `Sgrid` for the double precision version and `ggrid` and `sgrid` for the single precision version. They are located in `/usr/local/GridPro/az3000`.

To make things easier to explain, let's say the topology is in the file `topo.fra` and the schedule is in the file `topo.sch`. Always remember that the topology file must have the name extension `.fra`; the schedule file must have the name extension `.sch`; and, the name prefix part must be the same for both files.

3.1.1 Generating new grids

`Ggrid` serves for multiple purposes when running with different command line option flags. Among them, the most important one is to generate grids. This includes generating a new grid or resuming a previously stopped run.

To generate a new grid, one can simply type,

```
Ggrid  topo <ret>
```

or

```
Ggrid  topo.fra <ret>
```

GridPro will first generate the topology from `topo.fra`, then, schedule the run according to `topo.sch`. If `topo.sch` does not exist, GridPro will create a default `topo.sch` file for you. You may edit it or use it as is. If a syntax error or a topology error is found, the run will print out the type and place of the error, and either stop the run or switch to the debug mode.

3.1.2 Resuming a run

To continue a previously stopped run, a binary dump file or an ASCII file containing block data from the previous run must exist. The dump file has a default name `dump.tmp`. It is generated with a line,

```
write -D 0 -f dump.tmp
```

in the output section of the schedule file `topo.sch`. `dump.tmp` contains the grid data and run parameters for a run. Normally, depending on the schedule file, `dump.tmp` is periodically updated during a run.

The command line to continue a previous run is as follows,

```
Ggrid  topo -r data_file <ret>
```

or

```
Ggrid  topo -r <ret>
```

In the second form, the `data_file` name is assumed to be `dump.tmp`.

In running the command above, if `dump.tmp` or `data_file` does not exist, GridPro will prompt the user to input a new file name.

In fact, the `data_file` may be an ASCII grid data file generated with a schedule line such as:

```
write   -a -f blk.tmp
```

In this case, the run is scheduled to start at step 1, since the ASCII grid data file does not contain any information on run status.

For a resumed run, the topology file will be read again. Any inconsistency in the topology design is considered an error. Furthermore, you cannot change the order of the corner definition statements in a component; Neither, can you change the order of corners listed in the link list of a corner. However, certain changes to the topology file are allowed. Some of them will affect the resumed run; Some of them will not. Things that can be safely changed are:

1) Surface definitions: You may completely respecify a surface. But, you should not add or delete a surface definition. If an `.h` file is involved in a change, you must use `Sgrid` to regenerate the executable `Ugrid`. A resumed run will take the new definitions of surfaces. An example is that for a slightly different surface, you do not have to rerun the whole process. You can simply take the grid generated for the first surface, and then resume the run for the changed surface. In this way, you save the CPU time. Of course, the changes should not be too large. Otherwise, the run may collapse.

2) `PRINT` and `OUTPUT` statements: They can be added or deleted as you wish. These changes will only affect the screen and certain output files of the run.

3) `g, < >` statements and the corner's initial positions: They can be added, deleted or modified. But, changes have no effect on the resumed run.

3.1.3 Setting up initial grid with input block data

Setting up the initial grid with input block data is very similar to resuming a run, except that the run is always scheduled to start at step 1 and the grid densities on edges are the same as those specified in the topology file, rather than those of the input block data.

The input block data can be either a binary dump file or an ASCII data file generated by GridPro for the same topology. The procedure and requirements are very similar to those in the previous subsection as well.

The command line is now,

```
Ggrid  topo -R data_file <ret>
```

or

```
Ggrid  topo -R <ret>
```

3.1.4 Debugging your topology

The second function of `Ggrid` is to debug topologies. If GridPro detects a topology error when running `Ggrid` for generating grids, the run will switch to a topology debug session automatically. The debug session can also be invoked with the `-d` flag as follows,

```
Ggrid  topo -d <ret>
```

A debug session is indicated by the screen line

```
+++Begin debug session+++
```

and the screen prompt 'input obj:'. In a debug session, one can walk through the topology and inspect various aspects of the topology.

The current debugger is still very primitive. The main function of it is to display topology information in terms of corner tracers using information given in terms of internal labels and vice versa. The error messages from GridPro are usually given in terms of internal labels and the debugger provides a means to translate these labels to those in terms of the original topology design.

With GridPro, every corner in a final topology is associated with a tracer which labels the component path from which the corner is created.

For example in a debug session, if you type,

```
c 60 <ret>
```

you may see following lines displayed on the screen

```
c 60 -2.14881 -3.4931    100  -s 6  -L  6 56 62 64 181
c 60 <- clp2(4:1:1:2)
        lk   6 <- A3(7)          lk 56 <- clp1(4:1:1:1:2)
        lk  62 <- clp3(4:1:2)    lk 64 <- clp1(4:2:1:2)
        lk 181 <- clp2(4:1:1:2)
```

The first line is the internal definition of corner 60. The syntax is the same as a `c` statement in TIL, except that all references to surfaces and corners are internal labels; the initial position is given with a fixed vector; and the link list has no order restrictions. The other lines above give the tracers of the corners defined or referenced. The second line says that corner 60 is originated from a component called `clp2`. The tracer of corner 60 is (4:1:1:2), i.e. corner 60 is corner 2 of `INPUT` 1 of `INPUT` 1 of `INPUT` 4 in the head component. Of course, the first '`INPUT` 1' is also the originating component `clp2`. The next three lines provide the same information for all linked corners.

In the line

```
c 60 <ret>
```

the character `c` is called a debug key and the number 60 is an internal label for a corner.

A list of examples of debug key usage is as follows:

`h` — help messages.

`q` — quit GridPro.

`t 1:2 3:4` — list information about corners with tracers 1:2 and 3:4.

`c 60 50 71` — list information about corners 60, 50 and 71.

`e 60 50 71` — list information about edges 60, 50 and 71.

`f 60 50 71` — list information about faces 60, 50 and 71.

`b 60 50 71` — list information about blocks 60, 50 and 71.

`E 60 50 71 80` — list information about Edge(60,50) and Edge(71,80).

`F 60 50 71 80` — list information about Face(60,50) and Face(71,80).

`B 60 50 71 80` — list information about Block(60,50) and Block(71,80).

`s 5 3 0` — list information about surface 5, 3 and 0.

`sc 0 1` — list corners on surface 0 and 1.

`se 0 1` — list edges on surface 0 and 1.

`sf 0 1` — list faces on surface 0 and 1.

`sx 0 1` — list surfaces that topologically intersect surface 0 or 1.

3.1.5 Parameter settings

There are many parameters that can be set when running GridPro. The following sequence provides the setting process with increasing priority (a later setting always overwrites an earlier one).

1) Internal defaults.
2) File '`GridPro/az3000/az3000.def`'. (static)
3) File '`$HOME/.az3000.def`'. (static, optional).
4) File '`./az3000.def`'. (static, optional).
5) TIL program. (static)
6) Schedule file. (dynamic)
7) Per sweep parameter file. (`.par` file, dynamic, optional)

3.1.6 Topology with non-builtin implicit surfaces

'Sgrid' is used to generate user surface libraries for the cases where some of the surfaces are user defined implicit surfaces.

In running `Sgrid`, a subdirectory 'work.tmp' will be created in the current directory if it does not already exist. 'work.tmp' is used to collect certain working files generated with `Sgrid`.

To generate and link a user library, type,

```
Sgrid  topo.fra <ret>
```

or

```
Sgrid  topo <ret>
```

A link to the executable called `Ugrid` will be created in the current directory and it will be used to generate grids. The procedure of generating grids follows the discussion in the previous subsections, except the word, `Ggrid` should be replaced by `Ugrid`.

In general, whenever an implicit surface is modified internally (file content changes) or externally (transformation changes in the topology file), or the topology is changed, `Sgrid` should be rerun. For any other changes in the topology file or the schedule file, `Ugrid` does not need to be regenerated.

3.2 Schedule capabilities

A schedule is necessary because the grid generation with GridPro is accomplished in multiple sweeps. It can be rather CPU intensive. A typical Euler grid of 50,000 points can be generated in less than 10 CPU minutes on an IBM RS6000 320h machine without much carefulness in terms of scheduling. However, a viscous grid of the same scale may take more than 2 CPU hours to generate, depending on how well the run is scheduled.

A schedule file consists of two sections: A mandatory schedule section and an optional output section. A line is continued to the next by ending with a '\' character and a '#' character starts a comment line.

3.2.1 How to reference geometric objects in the schedule file

A geometric object means either a corner, a surface, an edge, a face or a block defined in and generated by the TIL program. There are two ways to reference a geometric object in the corresponding schedule file: One is to use an object label ; the other is to use the internal id assigned by GridPro;

An object label is a literal string beginning with a letter. Before an object label can be used in the schedule file, it must be already defined in the TIL file with the '`LABEL`' statement syntax. The same label can be defined multiple times. The overall result is the union of the objects of all the definitions. A label is, in effect, a convenient way to refer to a group of objects of the same type. 'ALL' (or 'all') and 'DEF' (or 'def') are predefined edge labels to mean all edges and edges with default grid density respectively. Identical names can be used for objects of different types. In each such case, they are treated just as if they have different labels.

If an internal id is used, the map between it and its original id can be obtained through running GridPro in the debug mode or with a `PRINT` statement in the topology file.

Let's take Figure 1.2(b) as an example. Suppose we want to increase the number of the wrap around grid lines for the lower circle, but not the upper circle. Also, suppose we want to

do this after 100 sweeps instead of at the beginning of the run. Then, this can only be done in the schedule file with a '-g' action. This task can be accomplished by increasing grid density on Edge(1,cb:1) in COMPONENT circle of Program ?? for the lower circle.

To find out the internal ids for corners 1 and cb:1 of the lower circle, we first insert a PRINT line in the component circle as follows,

```
COMPONENT circle( cIN  cb[1..4] )
BEGIN
  s 1   -ellip (0.05 0.05 0) ;

  c 1   -20 -20 0   -s 1   -L cb:1 ;
  c 2    20 -20 0   -s 1   -L cb:2 1 ;
  c 3    20  20 0   -s 1   -L cb:3 2 ;
  c 4   -20  20 0   -s 1   -L cb:4 3 1 ;
  LABEL E_GRP = e(1 2 2 3);
  LABEL E_GRP = e((1,2),(3,4),(1,3)); #note: '()' and ',' can be used.
  PRINT("internal label maps: 1->%m, cb:1->%m \n",1,cb:1);
END
```

The label E_GRP is defined twice for edges. It contains valid edges in both of the definitions, namely, Edge(1,2), Edge(2,3), and Edge(3,4). Since corners 1 and 3 do not define an edge they will be ignored. The label name can be any string with a mix of letters, numbers and '_' beginning with a letter.

The only function of a PRINT statement is to print information. It will not in any way change the topology. Therefore, one can feel free to add and delete them in the topology file. Now we can run GridPro with

```
Ggrid  topo <ret>
```

and kill the run after the topology is completed. On the screen, one will see two lines like this,

```
.
.
internal label maps: 1->31, cb:1->22
internal label maps: 1->35, cb:1->29
.
.
```

Each is printed when a component circle is processed by GridPro. Since the lower circle is INPUT 2 and the upper circle is INPUT 3 in the component circ2InBox, the first of the above two lines is for the lower circle. Thus, using the internal labels, the intended edge, Edge(1,cb:1), is referred to as Edge(31,22).

3.2.2 Schedule section

The schedule section is composed of a sequence of steps with the syntax as follows:

step *num*: *actions*

where *num* is a step label and *actions* is a sequence of actions that will be executed from left to right. The steps are executed one by one. If a step is in the gap of the steps listed in the

schedule file, it is implied that the actions for this step are the same as those of the next nearest step explicitly specified in the file. For an itemized action list, see Appendix A.

This section is focused on the following example which is a schedule file used to generate a viscous grid for a 3-element airfoil case,

Program 3.1 *File 'A3.sch'*

```
#schedule section follows
step  1:  -R CTRL.SINGULAR 1.35
step  2:  -g  EDGE_GRP1 24 EDGE_GRP2 32 EDGE_GRP3  16  -S 100
step 20:  -S -w
step 21:   -g all 2x  -S -w
step 23:  -ca all 1.0 -S -w

#output section follows
write -D 0 -f dump.tmp
write      -f  blk.tmp         #all blocks
```

A detailed explanation is given below,

Step 1

Actions:

1) (`-R CTRL.SINGULAR 1.35`): Singularity parameter is set to 1.35.

Step 2

Actions:

1) (`-g EDGE_GRP1 24 EDGE_GRP2 32 EDGE_GRP3 16`): The grid densities on the edges collected in label `EDGE_GRP1` are set to 24 and so on so forth. `EDGE_GRP1`, `EDGE_GRP2` and `EDGE_GRP3` should be defined in the TIL program. They can be multiple defined. An undefined label will be ignored in schedule.

2) (`-S 100`): Set *sweep_length* to 100 sweeps, and perform 100 sweeps.

Step 3 to step 20

Actions:

1) (`-S`): Perform *sweep_length*(=100) many sweeps

2) (`-w`): Output grid.

Step 21:

Actions:

1) (`-g all 2x`): Double grid density on every edge. `all` in GridPro is a predefined label for edges. It has all the edges.

2) (`-S`): Perform *sweep_length*(=100) many sweeps.

3) (`-w`): Output grid.

Step 22 to 23:

Actions:

1) (`-ca all 1.0`): Switch on the clustering. `all` means for all surfaces that has a `-c` flag in the surface definition statement in the TIL code. `1.0` means the off-wall grid spacing spec is multiplied by the factor `1.0`.

2) (`-S`): Perform *sweep_length*(=100) many sweeps.

3) (`-w`): Output grid.

For a complete list of actions, see Appendix A.

3.2.3 Output section

The output section determines the grid data to be outputted when a '`-w`' action is encountered in the schedule section. The output section is composed of lines beginning with the key word '`write`' . The most frequently used output section is similar to the one used in Program 3.1. It appears in the form,

```
write -D 0 -f dump.tmp
write      -f  blk.tmp          #all blocks
```

The first line sends the dump data to the file 'dump.tmp'. It is useful when a resumed run is expected. The second line outputs grid data for all blocks and sends it to the file 'blk.tmp' in the simple point data format. (see Program **??**). The file 'blk.tmp' is normally passed to the grid viewer for display or to a flow solver for CFD solutions. For a complete list of options, see Appendix A.

In fact, the above output section is equivalent to the two lines below,

```
write -D 0
write
```

if you have not redefined the default file name settings.

An alternative for outputting grid data is to use `OUTPUT` statements in the topology file. It is most useful for debug purposes when a complex geometry and multiple levels of components are involved. The main advantage is that the corner references are done in terms of the original labels, instead of internal labels. This makes it much easier to locate the blocks intended for output.

3.3 Output of GridPro

The output of GridPro has two elements: The block grid data and the block connectivity data. They are stored in separate files in the GridPro formats. Tools (`chfmt, mrgb..`) are available for format conversions and data restructurings.

3.3.1 Block grid data

The block grid data is in a simple point format in which the grid is listed block by block starting from block 1. For each block the data can be read from a FORTRAN program as follows:

Program 3.2 *Point data format in FORTRAN*

```
      READ(UNIT,*) IMAX,JMAX,KMAX
      DO 10 I=1,IMAX
      DO 10 J=1,JMAX
      DO 10 k=1,KMAX
   10   READ(UNIT,*) X(I,J,K), Y(I,J,K), Z(I,J,K)
```

where `X, Y, Z` are the x, y, and z coordinates of a grid point. Each of x, y, and z in the data has at least 10 significant digits.

For 2-d data, `KMAX` is set to 1 and `Z(I,J,K)`=0 for all `I,J,` and `K`.

Note that the number of blocks is not explicitly specified in the data file. The advantage here is that two or more grid files can be simply concatenated together to form a valid grid file.

3.3.2 Connectivity Information

Whenever `Ggrid` or `Ugrid` is run, the connection information of the blocks is automatically stored in the file `blk.tmp.conn`. The data format again is for general 3-d cases.

First let's introduce several terminologies.

Index space of a block: This is a space where a grid point of the block has the coordinates (i,j,k) where i, j and k are the array indices of the point in the block. The index space of a block provides a local coordinate system for a grid point.

Index axis: They are the i-axis, j-axis and k-axis. The i-axis for a block is an axis in the index space along the direction of increasing i. The j-axis and k-axis are similarly defined.

The connection data is specified in terms of index spaces and index axes. They convey the information about the topological structure of the block design and are independent of the real space locations of grid points.

For a grid point common to two neighbouring blocks, the two index spaces overlap and define a unique map from one index space to the other. This map is a simple whole axis to whole axis map. That is, an index axis of a block is mapped to an index axis of the other block, either in parallel or in anti parallel. It is unique also in the sense that all the overlap points of the two blocks define the same map.

Let's now explain the format for the connectivity file `blk.tmp.conn`. A `blk.tmp.conn` file consists of three data sections: (1) block connectivity, (2) block labels and (3) surface labels. The first line of each section gives the number of items in the section, and each data item takes one line.

The following is a part of the connectivity file `blk.tmp.conn` for a typical run case,

```
79     blocks
B 1 b 0 2 123  b 0 16 423 p  5 9 243 s 7 0 000 s 8 0 000 s 9 0 000 1
B 2 b 0 3 123  b 0 13 513 p -5 1 123 s 3 0 000 s 8 0 000 s 9 0 000 1
.
.
.
B 79 ...
0      block labels
3      surf  labels
FAR   7
FAR   2
ELEM  1
```

The first line in `blk.tmp.conn` gives the total number of blocks. The connectivity information for the blocks follows sequentially starting from block 1. For each block, the connectivity data takes one line which starts with the word `B` and a block id followed by 6 groups of 3 numbers led by an interface type flag. These 6 groups of numbers are the connection data for the I-begin, I-end, J-begin, J-end, K-begin and K-end sides of the block respectively.

Each group has the following format:

interface_type surface_id block_id axis_map

where

interface_type — The type of the neighbouring object on this face of the current block. It takes the following values:

`b` – Simple inter block interface: There is no surface assignment for the face, and the other side is a block.
`s` – Fixed surface-block interface: The face is on a fixed surface with or without a block on the other side.
`p` – Periodic surface-block interface: The face is on a periodic surface and the other side is a periodic block.

surface_id — For type 'b', the value of this number can be ignored.

For type 's' or 'p', this number must be non-zero. It is the signed id of the surface that the current block face is on with the sign indicating which of the two surface sides the current block is on. Note that, a block face with a surface assignment can be in one of two groups depending on which side of the surface it is on; The two sides of a surface are somewhat arbitrarily labelled as positive and negative. Note also that, the current block is the one given by the id following the key word '`B`' at the beginning of the line. Again, all ids here are the internal ids plus 1. Internal ids start from 0. Also for external fixed surfaces, all relevant blocks are on the positive side of the surfaces.

block_id — For type 's', this number can be zero, if there is no block on the other side of the current face.

For type 'b' or 'p', this number must be positive. It is the id of the block on the other side of the face.

axis_map — This is a three digit number where each digit has a value from 1 to 6 (or 0 if not used). The left, middle and right digits define, respectively, the axes on the neighbour block that the i, j, and k-axis of the current block map to. Here, i_axis, j_axis, k_axis, -i_axis, -j_axis and -k_axis of the neighbour block are assigned labels from 1 to 6 respectively. A value of 0 indicates that the map here is irrelevant. This happens when the neighbour is not a block, but a surface. The following is a complete list of values for the *i_axis_map*:

0 – Does not care.
1 – Maps to i_axis of the neighbor block.
2 – Maps to j_axis of the neighbor block.
3 – Maps to k_axis of the neighbor block.
4 – Maps to -i_axis of the neighbor block.

5 – Maps to -j_axis of the neighbor block.

6 – Maps to -k_axis of the neighbor block.

In this data format, there is quite a bit of information redundancy. However, with the repetition, one gains clarity.

We now explain the above example. The first line indicates that there are 79 blocks in the topology. Skip the second line and consider the third line which defines the connection data for block 2.

The first group of tokens (b 0 3 123) is for the I-begin side of block 2. It says that it is not on any surface and the neighbour is block 3, and the axis map is an identity map (123). Block 2 here is block 1 (=2-1) in terms of the internal label of GridPro. The second group of tokens (b 0 13 513) is for the I-end side of block 2 and it says that this face is not on any surfaces and the neighbour is block 13, and the axis map from block 2 to block 13 is (513 = i-axis to -j-axis, j-axis to i-axis and k-axis to k-axis). The third group of tokens (p -5 1 123) is for the J-begin side of block 2. It has a periodic boundary condition with respect to surface 4(=5-1). The minus sign means that this face of block 2 is on the minus side of surface 4. The second number, 1, indicates that with this periodic boundary condition, this face is connected to block 1 for which the axis map is given by the third number, 123, which is again an identity map. The fourth group of tokens (s 3 0 000) is for the J-end side of block 2. It means that this side is on surface 3 and there is no block on the other side of the face. The three zeros can be any other values and they are ignored. Surface 3 here is surface 2 (=3-1) internally in GridPro. The last two groups of tokens are for the K-begin and K-end sides of block 2. They can be similarly interpreted.

For a 2-d case, the only useful information is about the i-axis and j-axis. To be consistent, all k-axes map to k-axes as shown in the above example. Furthermore in 2-d, the k-axes of all blocks are parallel (as opposed to anti-parallel), and that the i-axis, j-axis and k-axis for a block form a right hand system.

Data sections 2 and 3, list all the block and surface labels defined in the TIL program with the LABEL statements. Each of the sections should have at least one line which is the line that contains the number of data items. A multiple defined label is listed multiple times, one for each block or surface.

3.4 Understanding the screen display

As GridPro is running, large amounts of information about the run status can be displayed on the screen and recorded in a log file called 'log.tmp'. It is very important to understand what is being displayed in order to fine tune your run. There are five sections of information listed in the order that the corresponding processing phase runs. They are:

3.4.1 Processing TIL input

This section of information lists the TIL file and component names being processed. Some important parameter settings are also displayed.

3.4.2 Generating topology

There are mainly three lists in this section: The first is a list of numbers of different objects generated. The second is for the statistics of singularities. And, the third is a list of all labels defined in the TIL code with the originating component name, the object type, and the internal object ids given.

3.4.3 Loading surfaces

This section lists all surfaces loaded by GridPro in the order of their internal ids.

3.4.4 Initializing grid

(not important, omit)

3.4.5 Scheduling grid generation

This is the most important section of information, also the longest. It is mainly an action-reaction listing. The actions are from the schedule file.

The most important information is the per-sweep information. By default, the per-sweep information for each sweep takes one line starting with the key 'swp' followed by the sweep number, and four or less pieces of run status information:

1) **Run phase status:** GridPro is run with 3 phases, named phase 0, 1 and 2. A run always starts with phase 0. A successful run always ends up in phase 2. This piece of information displays the number of surfaces and the number of remaining grid points in phase 0 and 1. If all the surfaces are in phase 2, nothing will be displayed.

2) **Maximum measures:** The key word here is 'max'. The worst occasions for four grid quality measures during the latest sweep are provided. They are:

Relative residue: Indicated by a letter 'r'. It is the ratio of the length of the residue vector and the length of the shortest side of the cell.

Relative surface curvature: Indicated by a letter 'c'. Relative surface curvature is only defined for surface grid points. It is the angle (in degree) between the surface normal vectors of two neighbouring surface grid points.

Spacing ratio: Indicated by a letter 's'. It is the length ratio of two consecutive grid line segments along an index direction.

Aspect ratio: Indicated by a letter 'a'. It is the length ratio of the longest and the shortest sides of a cell.

3) **Run load status:** This piece of information has a key word 'act' to mean active. It displays the percentages of blocks, volume grid points, and surface grid points that have been updated during the current sweep. Setting and resetting the various STOP parameters will affect the numbers here.

4) **Critical status:** This piece of information has a key word 'cri'. It provides the counts on sharp points and overshoot surface grid points in the last sweep. 'Sharpness' and 'overshoot' are different measures of tangent turnings along a grid line. A good grid should not have any overshoots and not have too many sharp points.

3.5 File usages and name conventions

In running GridPro , many different input and output files are involved. Here is a summary of the file usages and name conventions. Remember that not all files are needed (or generated) to run GridPro.

TIL program: (input) A TIL program can be spread into several files. The head file name must have the extension `.fra`.

Schedule: (input) A schedule file name must have the same name as the corresponding TIL program except that the file name extension must be `.sch`.

Grid data: (output) The default file name for grid data is `blk.tmp`.

Binary dump: (output-input) Used to resume a stopped run. The default file name for binary dumping is `dump.tmp`.

Connectivity: (output) The default file name for connectivity is `blk.tmp.conn`.

Log file: (output) The default file name for the log file is `log.tmp`. If an older `log.tmp` exists, it is renamed to `log.tmp. 1` first.

Position data: (input) Used for TIL I/O to input the initial setup positions and vectors through the READ statements.

Per user default parameter settings: (input) Used for the customized run parameters for a user. The file name must be `.az3000.def` and located under the HOME directory.

Per case default parameter settings: (input) Used for the customized run parameters for a run case. The file name must be `az3000.def` and located under the current case directory.

Per sweep parameter settings: (input) Used for dynamically adjusting run parameters. It is more flexible and less capable than the schedule.

Init twisted blocks: (output) the wire frames of twisted blocks in initial setup. The default file name is `badB.tmp`.

Chapter 4

Surface Specifications

This Chapter will be mainly concerned with surface specifications and their relation to surface definitions in the TIL programs. Some topics related to surfaces, such as, internal surfaces, overlapping surfaces and surface assignment rules will not be discussed here (see Chapter 2).

4.1 Surface classifications

4.1.1 Surface types

Every surface definition statement in a TIL program contains a flag for surface type. Surface types are used to indicate the format and structure of surface data. There are seven surface types in the current implementation of GridPro. A surface type belongs to one of the two group types used in GridPro.

The first group type is implicit (analytic) type. In this case, a surface is defined as an equal potential surface of a scalar valued analytic function of the position vector (x, y, z). Some of the simple forms of the functions are hard wired into GridPro. They are called built-in implicit types and are listed below,

```
-plane       --- plane surfaces.
-ellip       --- (super) ellipsoid surfaces.
-xpolar      --- used for periodic BC in polar coordinates.
-xyz         --- used for periodic BC in cartesian coordinates.
```

For non-builtin implicit surfaces, there is a type,

```
-implic      --- general implicit  surfaces.
```

The second group type consists of explicit surfaces. This group type can also be called the parametric type for the reason that a surface point can be determined by the values of a set of parameters.

In current implementation, it contains the following surface types,

```
-linear     --- a) single patch bilinear parametric surfaces.
                b) surfaces composed of multiple patches of bilinear
                   parametric surfaces.
-quad       --- surface of unstructured quadrilateral elements.
-tria       --- surface of unstructured triangular elements.
-tube       --- surface of revolution around a center curve.
```

A surface of this group is usually specified by a fair amount of data stored in a separate file(s).

4.1.2 Boundary modes

Boundary modes distinguish between different constraints on surfaces. Boundary modes are not explicitly specified in TIL programs. Whether a surface is of a certain boundary mode is determined by how the surface is used in GridPro. There are three boundary modes in GridPro: the fixed-surface mode, the periodic-surface mode and the float surface mode.

A surface of the fixed-surface mode has a fixed position in space. Most surfaces are used in this mode. They can be further divided into external surface mode and internal surface mode (see Chapter 2).

A surface in the periodic-surface mode has no fixed position in space. These surfaces are used to provide periodic boundary conditions. A surface in such a mode must be of an implicit type; that is, periodic boundary conditions have to be provided through analytic functions.

A surface in the float surface mode is an internal surface without location constraints. It is not a surface in the conventional sense. It is only a convenient way of grouping block faces, so that clustering can be done for them.

4.2 Fixed-surfaces – Implicit types

Under the fixed surface mode, an implicit surface is a surface specified by an equation in the form,

$$u(x, y, z) = 0$$

In general, $u(x, y, z)$ may be any function that can be programmed in C and satisfies the following conditions:

1) $u(x, y, z)$ is defined in a neighbourhood of the surface $u(x, y, z) = 0$. The neighbourhood should cover a domain larger than that expected for the initial positions of the corners which are assigned for the surface.

2) At any point in the intended physical domain, $u(x, y, z)$ should have a well defined gradient vector pointing to or away from the surface. The dependency of the gradient vector on (x, y, z) should be smooth.

3) The gradient vector of $u(x, y, z)$ should point into the region to be gridded.

4.2.1 Built-in implicit surfaces

Certain simple forms of implicit surfaces are built in with GridPro. They are:

The plane surface (`-plane`)

Example surface definition statement:

```
s  3  -plane  (0.1  0.2  0.3  0.4);
```

This statement defines surface 3 as a plane surface specified by the equation $0.1x + 0.2y + 0.3z + 0.4 = 0$ with the surface normal vector given by $(0.1, 0.2, 0.3)$. Notice that the parameter values to specify a given plane surface are not unique. They can be scaled by any fixed positive or negative constant without changing the surface. However, a negative scaling changes the direction of the normal vector (i.e. the orientation) of the plane.

The (super) ellipsoid surface (`-ellip`)

Example surface definition statement:

s 3 -ellip (0.1 0.2 0.3 4.0);

This statement defines surface 3 as a super ellipsoid specified by the equation $|0.1x|^4 + |0.2y|^4 + |0.3z|^4 - 1 = 0$. The gradient of the surface function $u(x, y, z) = |0.1x|^4 + |0.2y|^4 + |0.3z|^4 - 1$ is pointed to the outside of the super ellipsoid. The user must supply an orientation change operator '-o' when the gridded region is inside the ellipsoid (i.e. when the ellipsoid is an outer boundary). For a regular ellipsoid, the value for the power parameter is 2 (instead of 4 in the above example). The statement can be

s 3 -ellip (0.1 0.2 0.3 2.0);
or
s 3 -ellip (0.1 0.2 0.3);

4.2.2 Non-builtin implicit surfaces

A non-builtin implicit surface can be specified in the form of a '.h' file with the syntax of the C preprocessor. The up side of this is the flexibility of using different functional forms; The down side is that extra steps must be taken to compile a user generated function library in C and link it to the main module of GridPro before running GridPro.

Example surface definition statement:

s 3 -implic "torus.h";

This statement defines surface 3 to be an implicit surface as specified in the file "torus.h".

Let us say that we want "torus.h" to define a torus as follows: The central circle of the torus is on the y-z plane and is centered at origin with a radius 1.5. The cross-sectional circle of the torus has a radius of 0.5. The region to be gridded is inside the torus. The file "torus.h" is listed below,

Program 4.1 *File "torus.h"*

```
#define FUNCU ya=sqrt(y*y+z*z)-1.5, \
          1.0 - (ya*ya + x*x)*4  /* Define torus surface*/
#define Ulen    -1.0             /* -1.0 for no period  */
```

For those who do not have experience in the C programming language, three comments are in order: 1) A matching pair of '`/*`' and '`*/`' with everything in between will be ignored by the C compiler. They can therefore be used to make comments. 2) A '\' character at a line end continues this line to the next. 3) Each '`#define`' line defines a macro with the name of the macro and the content of the macro following.

If you want to define a different surface, you can copy this file and change the contents of the two pre-named macros, `FUNCU` and `Ulen`.

For `Ulen`>0, '`FUNCU` modulo `Ulen` = 0' specifies the surface; Otherwise '`FUNCU` = 0' specifies the surface.

In defining these macros, (x, y, z) are the input variables. Also available are 11 sets of auxiliary variables, (xa, ya, za), $(x0, y0, z0)$, .. , $(x9, y9, z9)$. They can be used to construct a surface function. These intermediate variables provide the opportunity to segment the surface formulation and to, thereby, achieve clarity. The segments are separated by the "," operator.

4.3 Fixed-surfaces – Explicit types

4.3.1 Surfaces of quadrilateral elements

Single patch bilinear parametric surface (`-linear`)

Example surface definition statement:

s 3 -linear "surf1.dat" ***[boundary_conditions]*** **;**

This defines a local bilinear parametric surface with certain *[boundary_conditions]*. The surface is specified by an IxJ array of grid points for some I and J in the file "surf1.dat". The bilinear interpolation is used to determine the surface points within the array cells. No collapsed cells or merged data points are allowed.

`The data format in file` "`surf1.dat`": The first line of the data file lists I and J. The subsequent lines list the three coordinates of a surface grid point in the order of increasing J, then I. In terms of FORTRAN, the data can be read by a program as follows:

```
READ(UNIT,*) IMAX,JMAX
DO 10 I=1,IMAX
DO 10 J=1,JMAX
10   READ(UNIT,*) X(I,J),Y(I,J),Z(I,J)
```

The *[boundary_conditions]* in the example line specifies how the boundaries of the surface piece should be treated by GridPro. The options are:

`+i` – i=0 link to i=0 side
`-i` – periodic in I direction
`+I` – i=IMAX link to i=IMAX side
`+j` – j=0 link to j=0 side
`-j` – periodic in surface J direction
`+J` – j=JMAX link to j=JMAX side

At most, one of the `+i,-i,+I` options and one of the `+j,-j,+J` options can be used for a surface definition statement.

Instead of the inline *[boundary_conditions]*, a connectivity file named `surf1.dat.conn` can exist to provide connectivity information. In any case, if there is no inline *[boundary_conditions]* or there is the file `surf1.dat.conn`, the surface is treated as a special case of multi-patch surfaces, discussed in the next subsection.

The natural surface orientation is defined by the normal vector $\hat{i} \times \hat{j}$ where $\hat{i}$ and $\hat{j}$ are the unit vectors along the i index and j index directions respectively for any of the array points that determine the surface. If the natural orientation is not the same as expected (i.e. the positive side of the surface should be facing the region to be gridded), the reversing orientation flag `-o` can be placed in the surface definition statement as follows,

s 3 -linear "surf1.dat" ***[boundary_conditions]*** `-o`**;**

Note that, a surface here is specified by a grid which, however, is not the same as the surface grid generated with GridPro. The relation between them is that the generated surface grid will be on the surface which is specified by yet a different grid.

Multiple patch bilinear parametric surfaces (`-linear`)

A surface can be specified by multiple patches. The type for it is again `-linear`. Two files are involved to specify a surface of this type; namely, a data file which contains a sequence of surface patches, and a connectivity file which provides the information as to how the patches in the data file are connected. The format for each of the patches in the data file is the same as the single patch surface.

The requirements on the connections between different surface patches are very mild. They only need to be nearly-full face matchings. To understand it, you can imagine that a surface is composed of full face matching quadrilateral patches. However, the data specifying the surface on a patch does not need to exactly cover that patch. There can be gaps between adjacent surface patches. Preferably, the gap scales are smaller than the element (cell) scale of the surface patches near the gaps. However, no overlaps are allowed.

Example surface definition statement:

s 3 -linear "surf1.dat";

File "`surf1.dat`" is also called the data file. An associated connectivity file that has the name "`surf1.dat.conn`" may or may not exist. If "`surf1.dat.conn`" does not exist, GridPro will use the surface data to generate it by some default rules.

"`surf1.dat.conn`" contains the information how the surface pieces are connected. An example of the connectivity file consists of the following lines,

```
1   6  1   6  -1   2  2   0  0
2   3  1   3  -1   1  2   0  0
3   2  1   2  -1   4  2   0  0
4   5  1   5  -1   3  2   0  0
5   4  1   4  -1   6  2   0  0
6   1  1   1  -1   5  2   0  0
```

Each line here defines a surface piece and its connection to other pieces. It has the following format,

sp_id nxi sdi nxI sdI nxj sdj nxJ sdJ

sp_id – An id number that labels the surface piece. It should appear sequentially and correspond to the data piece in the data file.

nxi sdi – Boundary condition for the i = IMIN side of current surface piece. *nxi* provides the neighboring surface piece id. *nxi=-1* means no neighbor surface piece. *sdi* indicates which side of the neighbor surface piece is connecting to the i = IMIN side of current surface piece. Values *sdi=1,-1,2*, and *-2* mean the i=IMIN side, i=IMAX side, j=JMIN side, and j=JMAX side respectively. If *nxi*$\leq$*0*, the value of *sdi* is irrelevant.

nxj sdj – Boundary condition for the j = JMIN side of current surface piece.

nxI sdI – Boundary condition for the i = IMAX side of current surface piece.

nxJ sdJ – Boundary condition for the j = JMAX side of current surface piece.

The orientation of the surface will be synchronized to that of the first piece.

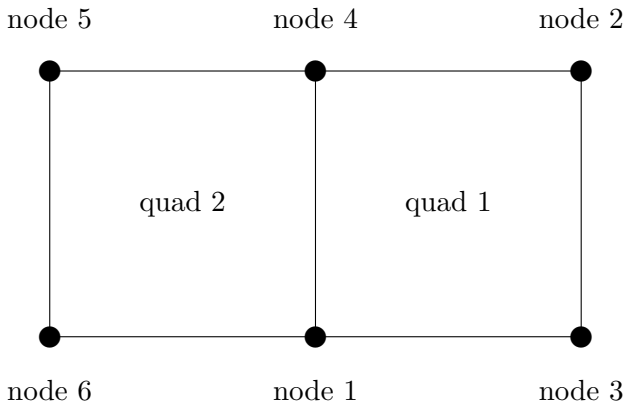


Figure 4.1: An example of a surface of quads with two quads.

Surfaces of unstructured quad elements

A surface point is specified by bi-linearly interpolating on the quad elements.

Example surface definition statement:

s 3 -quad "surf1.dat";

GridPro will interpret the data format according to the contents of the data file. It may be in one of the two implemented formats. The first is the small field Nastran bulk data fixed or free format, where the GRID entry is used to define a node coordinate position, and the CQUAD4 or CQUADR entry is used to specify a quad element (for detail see MSC/NASTRAN Quick Reference Guide).

The implementation is a subset of MSC/NASTRAN's specification. In particular, the replication capability is not implemented. That is, data entries should not contain the '=' and '*' operators. For the free format, GridPro requires that data entries are separated by commas.

The second data format is the GridPro format used by GridPro. As an example, '`surf1.dat`' may consist of the following lines:

```
6
801.97693   -7.0479345   -23.675423
886.42688    14.096904    -15.55965
942.72565    28.191708   -10.122647
999.00952    42.283035   -4.6956768
1055.3088    56.378807   0.71006197
1111.5261    70.471634    5.0577559
2
1 3 2 4 0
6 5 4 1 0
```

It is a node list followed by a quad list. The first line specifies that there are 6 nodes in this file. It is followed by the coordinates (x y z) of the 6 nodes (one node per line). The next line indicates that there are 2 quads in the data. Then, each of the following lines specifies the 4 node numbers and 1 property id for that quad. The node numbers are ranging from 1 to 6 corresponding to the nodes in the node list. In this case, quad number 1 is formed by nodes 1, 3, 2, and 4, and quad number 2 is formed by nodes 6, 5, 4, and 1. The property id is not used

in the graphical manager and Ggrid. It is used in some utility calculations. Therefore, the value of property id can be arbitrary here.

The listing order of node ids for a quad should be circling about the quad in either of the two possible directions. No more than one of the 4 sides of a quad may be collapsed either by having the same node id or by having a single space position. Small holes are allowed on the surface.

The natural orientation of a quad surface is determined by the first quad in the quad list. The positive side of the surface is the side where when one faces the first quad, the listing order of nodes that defines the quad rotates anti-clock wise (right hand rule).

4.3.2 Surfaces of triangular elements (`-tria`)

There is one type in this category, namely '`-tria`'. The data formats are similar to that of unstructured quad surfaces. Mainly, the differences will be pointed out here.

1) The Nastran data format entries used here are GRID, CTRIA3, and CTRIAR.

2) In the GridPro format, the element is defined by 3 nodes.

Example surface definition statement:

s 3 -tria "surf1.dat";

In the GridPro format, 'surf1.dat' may consist of the following lines:

```
4
886.42688     14.096904     -15.55965
942.72565     28.191708    -10.122647
999.00952     42.283035    -4.6956768
1055.3088     56.378807    0.71006197
2
1 3 2 0
4 1 2 0
```

Here we have 4 nodes and 2 triangles. No degenerate sides are allowed for any of the triangular elements.

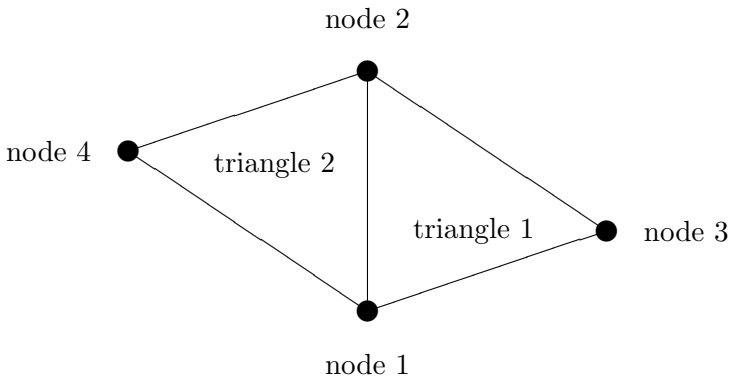


Figure 4.2: An example of a surface of triangles with two elements.

4.3.3 Surfaces of revolution (`-tube`)

A surface of this type is specified by the revolution around a digitized center curve. Therefore, it is a more general type of surface of revolution.

Example surface definition statement:

s 3 -tube "surf1.dat";

Here 'surf1.dat' contains the data for the digitized center curve. It can be read via,

```
READ(UNIT,*) IMAX
DO 10 I=1,IMAX
10  READ(UNIT,*) X(I),Y(I),Z(I),R(I)
```

Where `X(I),Y(I),Z(I)` are the coordinates of the $\texttt{I}^{th}$ center curve data point and `R(I)` is the radius of revolution for the data point.

The center curve point, the center curve tangent and the the radius of revolution are linearly interpolated from the center curve data.

The revolution for each point on the center curve is performed in the normal plane of that point with respect to the tangent of the curve. Thus, this type represents tube - like surfaces with curved center line and variable radius.

To avoid ill-specified surfaces, the circular disks bounded by the circle generated by the revolution for any two center curve data points should not intersect each other.

The natural orientation is pointed to the outside of the tube section defined by the first two data points on the center curve. Therefore the first two data points should not have the same position in space. However, in general two center curve data points *can* have the same position in space, as long as the radii are different.

An interesting example is a torus surface. There are two ways to represent it with surface type '`-tube`'. The first is to digitize the center circle of the torus, and use a constant radius for all of the center curve data points. The second way is to digitize the axis of rotational symmetry, and use a variable radius to represent the cross-sectional circle of the torus.

For both cases, the '`-i`' flag must be given to the surface definition statements to indicate that the center curves have a periodic boundary condition on them.

4.4 Periodic surfaces

A periodic surface is a surface used in the periodic mode; that is, it is used to determine a periodic boundary condition for the grid to be generated. The word "periodic" here should not be confused with the the word "periodic" used to indicate the periodic boundary condition for a surface. A surface may have loops that are closed with such conditions (instead of, for the grid to be generated).

A surface used to specify a periodic boundary condition has different and stronger requirements on the surface functions. More precisely, a periodic boundary condition is specified by providing a non-singular coordinate transformation in the form,

$$U = u(x, y, z)$$
$$V = v(x, y, z)$$
$$W = w(x, y, z)$$

A surface with the given periodic boundary condition is selected by GridPro among all the surfaces satisfying the condition,

$$\begin{array}{rcrcc} u(x,y,z) & - & u(x_{id},y_{id},z_{id}) & = & period \\ v(x,y,z) & - & v(x_{id},y_{id},z_{id}) & = & 0 \\ w(x,y,z) & - & w(x_{id},y_{id},z_{id}) & = & 0 \end{array}$$

on every pair of grid points (x, y, z) and (x_{id}, y_{id}, z_{id}), for which the identifying relation is defined by `i`-statements in the TIL program.

4.4.1 General implicit surfaces (`-implic`)

Similar to a non-builtin implicit surface in the fixed surface mode, an implicit surface in the periodic surface mode can be specified in the form of a '`.h`' file with C control-line syntax. In terms of their appearance as `s`-statements, there is no difference, except the period must be provided on the surface definition line in the TIL code. For example,

s 3 -implic "polar_for_z.h" 30.0;

where `30.0` indicates the period.

However, in the '`.h`' file, 9 pre-named macros are used.

`FUNCU, Ulen, FUNCV, Vlen, FUNCW,` and `Wlen` are used to define the forward transformation from (x, y, z) to (u, v, w) and `FUNCX, FUNCY,` and `FUNCZ` are used to define the inverse transformation from (u, v, w) to (x, y, z). At run time, the consistency of the inversion is checked using the initial positions of all the grid points assigned to the involved surface.

The way to define `FUNCU, Ulen, FUNCV, Vlen, FUNCW,` and `Wlen` is the same as for a non-builtin implicit surface in the fixed surface mode. However, to define `FUNCX, FUNCY,` and `FUNCZ`, one needs to use (u, v, w) as input variables instead of (x, y, z). For the details, see the next subsubsection.

4.4.2 The polar periodic BC (`-xpolar`)

One of the hard wired implicit surfaces is for the periodic boundary condition on the angular coordinate in the polar coordinate system with x-axis as the rotation axis. The angle is measured in degrees.

Example surface definition statement:

s 3 -xpolar 30.0;

where `30.0` indicates the period is 30 degrees. There is no other data specification needed.

The transformation used is,

$$\begin{array}{rcc} U & = & \text{atan}(\frac{z}{y}) \cdot 180/\pi \\ V & = & x \\ W & = & \sqrt{z^2+y^2} \end{array}$$

Two points identified by this periodic surface condition will have the same V and W values and a fixed difference in U.

The corresponding '`.h`' file would appear as,

Program 4.2 *File* 'period.h'

```
#define FUNCU ((180/PI)*atan(z/(y+dsig(y)*1.0e-30))+((y<0)? 180 : 0))
#define Ulen  ( 360 )

#define FUNCV  x
#define Vlen  (-1.0)

#define FUNCW  sqrt(z*z+y*y)
#define Wlen   (-1.0)

#define FUNCX   v
#define FUNCY  (w*cos(u*PI/180))
#define FUNCZ  (w*sin(u*PI/180))
```

Here, certain things are added in to prevent an over flow condition. `PI` and `PI2` are predefined. At the end of `FUNCU`, '`((y<0)? 180 : 0))`' means that if $y < 0$ then the value is 180; otherwise it is 0. This is just some C programming syntax.

Rotation and translation operators can be applied in the s-statement to change the rotation axis of the polar system. For the polar system with z-axis as the rotation axis, one has,

```
s 3 -xpolar 30 -R 0 0 1   0 1 0   1 0 0;
```

4.4.3 The cartesian periodic BC (-xyz)

The other hard wired implicit surface is for the periodic boundary condition on the x-axis in the cartesian coordinate system.

Example surface definition statement:

s 3 -xyz 2.5;

where `2.5` indicates the period is 2.5. There is no other data specification needed.

The transformation used is,

$$\begin{aligned} U &= x \\ V &= y \\ W &= z \end{aligned}$$

Two points identified by this periodic surface condition will have the same V and W values and a fixed difference in U (hence x).

The corresponding '`.h`' file would appear as,

Program 4.3 *File* 'period.h'

```
#define FUNCU (x)
#define Ulen  (-1.0 )

#define FUNCV (y)
#define Vlen  (-1.0)

#define FUNCW (z)
#define Wlen  (-1.0)
```

```
#define FUNCX  u
#define FUNCY  v
#define FUNCZ  w
```

Rotation and translation operators can be applied in the s-statement to change the rotation axis of the polar system. For the cartesian system with z-axis as the periodic axis, one has,

```
s 3 -xyz 2.5 -R 0 0 1   0 1 0   1 0 0;
```

4.5 Float surfaces

As we said earlier, the purpose of float surfaces is to provide a convenient way of grouping block faces, so that clustering can be applied for them. Unlike surfaces in other modes, there is no location constraints on a float surface. An example statement is:

s 1 -float +c 0.01;

Here the float mode is indicated by the type parameter `-float`, and the average spacing on either side of the surface is 0.01. Other than that, there is no location constraint and it can be internal to the topology (that is, blocks can be on both sides of the surface), the use of a float surface is very similar to that of a fixed surface. In particular, corners can be assigned to the surface; And the surface can be added to or deleted from a corner, an edge, or a face. The clustering is turned on or off along with the corresponding clustering group. However, when one assigns corners or other objects to a float surface, they must satisfy a consistent requirement. That is, all the block faces on a float surface must be in the same face sheet. In turn, a face sheet is defined as a side of a block sheet. Note that one can have a part of a face sheet assigned to a float surface.

Clustering may also be applied for only one side of the surface. A statement such as,

s 1 -float 0 +c 0.01;

will apply clustering for the side 0. The other side is side 1. The determination of side 0 or 1 is best done by experiment.

4.6 Surface transformations

In the TIL code, a surface can be linearly transformed and translated with the corresponding option flags and arguments in the s-statement. The vector expression form of such transformation is explained in Section 2.4 of Chapter 5.

4.7 Surface conditions

In this section, we will discuss certain requirements that GridPro imposes on the surfaces.

4.7.1 Smoothness

A surface used by GridPro must be "smooth" . However, whether a given surface is smooth or not strongly depends on the topology you choose for the blocks on that surface.

To illustrate the concept of smoothness here, let's use a simple example. Consider an airfoil with a sharp trailing edge. Suppose that the airfoil is specified by a surface of type `-linear` with a dense enough supply of points. Then, most of the places on the surface are smooth for GridPro. However at the sharp trailing edge, when one goes from one surface data point to another, the surface tangent vector will turn by a very large amount (close to 180°).

For the case where a C-cut topology is used, a block boundary is forced to go to this point. Therefore, the sharp trailing edge will not cause a problem and the surface will be regarded as smooth.

On the other hand, if an O-type topology is chosen, GridPro will automatically distribute grid points taking the airfoil as a seamless whole. In this case, the surface point on the sharp trailing edge will be treated the same as any other surface point and the large tangent turn will be regarded as non-smooth.

For a non-smooth surface, it can be restructured by adding surface data points to round off those large tangent turns. In doing so, the surface shape may be changed slightly, but since it can be done at a very fine scale, the changes should not affect the flow field solutions.

One should also avoid specifying a surface with too many data points since it wastes both memory and CPU time. In terms of tangent turning rates, limiting to less than 1° turns will give very good surface specifications, even though GridPro can handle tangent turns as large as 90°, provided that they do not form clusters or sharp points.

4.7.2 Intersections

In general, if two surfaces are supposed to intersect, the surface specifications should be provided to extend somewhat beyond the intersection. GridPro will determine the actual intersection automatically once the extensions are in place.

Chapter 5

Generating Better Grids

At this point, you should have most of the bits and pieces of GridPro in your hands. This Chapter concerns how you can better use them. We also provide some real live experiences, tips, tricks and hints.

5.1 Designing better topologies

5.1.1 Topological singularities

In 3-d, a regular grid point in a structured grid has 6 neighbouring grid points which form a well defined index coordinate system. For multi-block structured grids, most of the grid points are regular. However, singular grid points are introduced for steering the grid lines to better fit a complex geometry.

Singularities can be classified on the topological level, instead of for grid points. There are two types of topological singularities: singular block corners, and singular block edges. A singular grid point is either on a singular corner or on a singular edge.

For 3-d cases, the term singularity usually means singular edges. On the other hand, for 2-d, singular corners are the only type of singularities.

A singularity is ranked with the number of neighbouring blocks. For internal edges (i.e. edges not on surfaces) in 3-d, a regular one has 4 neighbouring blocks. An internal edge having three neighbouring blocks is called a singularity of degree 3. Similarly, an internal edge having five neighbouring blocks is called a singularity of degree 5. It is more difficult to generate good grids around a singularity as the degree moves farther from the regular number 4.

This subsection is concerned mostly about internal singularities. Singularities on surfaces are discussed in the next subsection.

GridPro restricts the degree of a singularity to be either 3, or 5 to 8. However, one should almost exclusively limit his singularities to degree 3 and degree 5.

A singularity of degree higher than 5 can always be split into several singularities of degree 5 using a general procedure without completely redesigning the topology. The trade off is increasing the number of blocks.

A simpler procedure will work most of the time. For a 2-d case, Figure 5.1 is a local view showing how a singularity of degree 6 can be split into two singularities of degree 5. The simple procedure is described as follows for 2-d cases: First, find a simplest partition along the block boundaries that satisfies the conditions: a) It passes through the singularity to be split; b) It splits any corner not more than once; c) At any split point, there are more than two block edges on each side unless it is a singularity of degree 3; and d) It cuts the topology into disjoint parts.

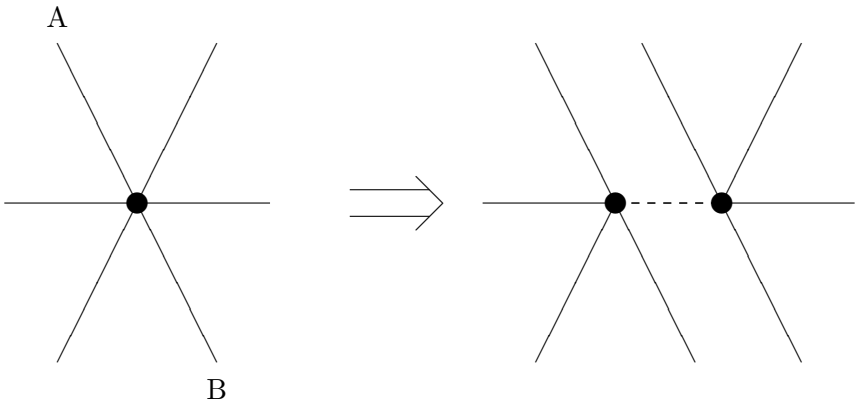


Figure 5.1: A singularity of degree 6 is split into two of degree 5.

Second, for any corner split in the partition, make a corner link (the dash line in Figure 5.1) between the two corners created.

This procedure puts a layer of blocks into the partition gap. It creates a valid topology with a reduced severity for any singularity of a degree higher than 5.

One may try one's own way to reduce the singular degree of a corner. The goal is to do it locally without making other corners worse.

Singularities of degree higher than 5 are used for the cases where their use will significantly reduce the number of blocks. When a sufficiently high degree of symmetry is present in the surface geometry and in the block topology near these singularities, the blocks around a singularity will be more or less uniformly distributed.

5.1.2 Wrapping around surfaces

One should avoid putting singularities on surfaces. In other words, one should try his or her best to design a topology that wraps around the surfaces.

The following situations are considered to be regular on surfaces (3-d): an edge on a single surface having two neighbouring blocks; and, an edge on the intersection of two surfaces having only one neighbouring block. Any other surface edge is a singular surface edge and should be avoided in the design. For 2-d cases, things said about edges can be said about corners. Thus, the topology design in Figure 1.2 has 4 singularities on the surfaces. They are the 4 corners of the outer box. A better design should avoid them and it is shown in Figure 5.2.

In some situations, insisting on using regular corners or edges on surfaces may result in bad topology. However, one does not have to use singularities on surfaces to solve the problem. An alternative is to redo the surface division.

5.1.3 Testing components

Cleverly using components can save human time for a topology design and make the design easier to maintain and modify.

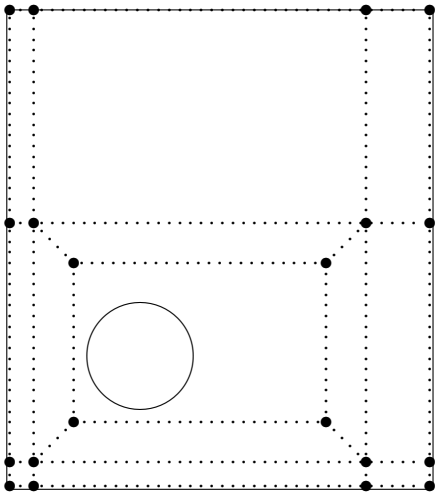


Figure 5.2: A better topology for Figure 1.2(a).

The underlying principle is to take advantage of any symmetry either in the surface geometry or in the topology.

For a complex geometry, the grid generation process should also involve multiple levels. Topologies for components should be debugged individually to localize any possible topological error. Since a cycle of grid generation may take as long as 15 CPU hours for a complete complex problem, it is always good to first go through the grid generation processes for those components for which the topologies are trickier to design and may require several redesigns.

To do this, you need to write a head component for testing purposes that takes the component to be tested as an input component and defines the proper surfaces for the far field. The ideal far field should, more or less, emulate the real environment that may be encountered by the tested component. However, for many cases, a far field set carelessly may do equally well in real situations.

5.2 Designing a distribution of grid points

In general, one has no control over which grid point goes where with GridPro. The necessity of a control at such a fine level is precisely the weakest point of all other multi-block grid generators.

For almost all cases, one does not and should not care whether a particular grid point is located at a particular position. Instead, it is and should be much more important to have the grid points reach the desired density, orthogonality and smoothness in certain regions of the problem. Often, the grid density is the first thing one wants to have control over. An optimal distribution is one where certain sensitive regions are distributed with a very dense coverage of grid points. On the other hand, such high densities must not propagate to the regions where there is not much action going on.

A traditional C type grid for a typical airfoil with clustering to the wake line is not optimal in terms of the grid point distribution since the dense grid points near the trailing edge of the airfoil propagate to the far field, which is considered a waste of computing resources.

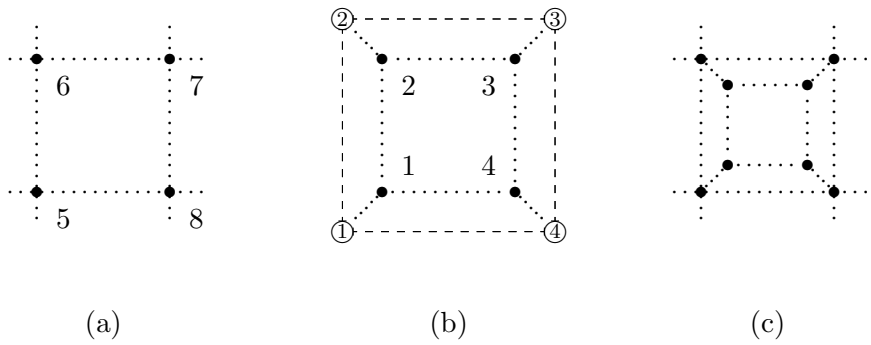


Figure 5.3: Compact enrichment for an internal block. (a) The internal block. (b) The component `hypCube` used to do the enrichment. (c) The local view of the final topology.

GridPro provides three means to control the distribution of grid points to various degrees. They are 1) Edge grid density assignments; 2) Cluster parameters; and 3) enhanced topologies.

5.2.1 Changing the grid density on edges

The edge grid density can be set initially by the global parameter `GRIDDEN` and more specifically with the `g` statements in a TIL program. They can also be dynamically changed in the schedule file.

By itself, this is the weakest method to control the distribution of grid points. In general, a change of the grid density on an edge is not local. The effect will smooth out over a region limited by the local topology, which, in many cases, can be quite large. On the other hand, it is a useful tool when one desires such an effect. In general, it is used to assign the overall grid densities for run cases and for fine tuning the grid distributions. It is also useful when used in conjunction with the other two methods.

5.2.2 Using the cluster parameters

Clustering is not a general means for controlling the grid point distribution, but, it can have a very strong effect on particular types of regions such as those near surfaces and singularities.

The singularity clustering can be turned on and off in the schedule file and it is controlled by a strength parameter in the schedule. The parameter takes a value between 0 and 1 with 0 meaning no clustering at all. A value between 0.25 and 0.35 gives a good clustering for most cases.

The cluster parameter for a surface is a targeted off-wall normal spacing of the first layer of grid points.

5.2.3 Enhancing your topology

Topology enhancements provide a general and strong method to control the grid densities. In essence, it is the same as redesigning the topology. However, it is done locally with a particular underlying concept, **compact enrichment**, and by following a well-developed procedure. These make it a powerful tool more than simply redesigning the topology.

We will use 2-d examples to illustrate the concept of compact enrichment. Consider that the block (corners 5 to 8) in a topology as shown in Figure 5.3(a) is to be enriched. We construct the component `hypCube` shown in Figure 5.3(b) to do the job.

The local view of the final topology created by putting `hypCube` into the block in Figure 5.3(a) is shown in Figure 5.3(c). The dashed lines in Figure 5.3(b) give the environmental topology expected by `hypCube`. Other symbols have the same meanings as before.

In terms of TIL programming, we first define the new component, `hypCube`

```
COMPONENT hypCube(cIN c[1..4])
BEGIN
  c 1 @ 0.9*<c:1>+0.1*<c:3> -L c:1;
  c 2 @ 0.9*<c:2>+0.1*<c:4> -L c:2 1;
  c 3 @ 0.9*<c:3>+0.1*<c:1> -L c:3 2;
  c 4 @ 0.9*<c:4>+0.1*<c:2> -L c:4 3 1;
  x f c:1 c:3;
END
```

Then add the following `INPUT` statement in the TIL program for the topology in Figure 5.3(a),

```
.
.
INPUT 11 hypCube( cIN (5..8) );
.
.
```

The input number may not be 11. It is used here just to give you an example.

Now, you can increase the grid density in the local region shown in Figure 5.3(a) by increasing the grid density on Edge(1,c:1). The effect of it is to create looping grid lines without sending them to other blocks.

This procedure may create new singularities on corners 5 to 8 in Figure 5.3(a). In general, the positions of singularities have a pinning effect to a certain degree. That is, the dependence on the edge grid density is much weaker than that of the regular grid points without them.

The compact enrichment concept takes advantage of two aspects of a carefully redesigned local topology: 1) Making the creation of local looping lines possible; and 2) Using the pinning effects of singularities.

If a simple compact enrichment cannot reach the desired density, one may gradually increase the complexity of the local topology and do it in a multiple level style.

However, one should avoid trivial enrichments. One such case is to put another `hypCube` in Block(1,3) of Figure 5.3(b) since this results in an unchanged topology after certain block merges. Thus, if the desired grid distribution could not be achieved before, it will not be achieved now.

Certain cautions should be taken in doing compact enrichment. As we have seen, local topology changes can turn some corners and edges into singularities of higher degrees. One should make sure that the singularities created are of degrees 3 and 5.

The next example shows you how a simple compact enrichment on a surface block can be done (Figure 5.4). The TIL program is listed as follows,

```
COMPONENT clamp(sIN s, cIN c[1..4])
BEGIN
  c 1 @ 0.9*<c:1>+0.1*<c:2> -s s  -L c:1;
  c 2 @ 0.9*<c:2>+0.1*<c:1> -s s  -L c:2 1;
  c 3 @ 0.9*<c:3>+0.1*<c:1>       -L c:3 2;
  c 4 @ 0.9*<c:4>+0.1*<c:2>       -L c:4 3 1;
```

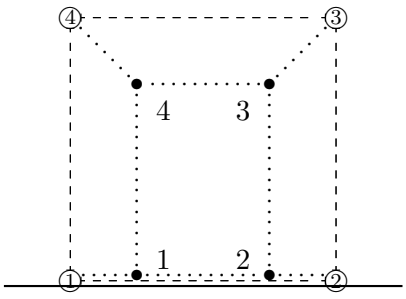


Figure 5.4: Compact enrichment for a surface block (Component `clamp`).

```
  x e c:1 c:2;
END
```

In this case, the singularity degrees for corners ① and ② are not changed.

Compact enrichment can be used on more than one local block. Three new components `clamp2`, `clamp3` and `clamps` are constructed for two and three connected blocks in which one is on a surface.

```
COMPONENT clamp2(sIN s, cIN c[1..6])
BEGIN
  INPUT 1 clamp( sIN (-1), cIN (c:6 c:3..5), cOUT (1..4) );
  c 1 @ 0.9*<c:1>+0.1*<c:2> -s s  -L c:1 1:1;
  c 2 @ 0.9*<c:2>+0.1*<c:1> -s s  -L c:2 1:2 1;
  x e c:1 c:2;
END

COMPONENT clamp3(sIN s, cIN c[1..8])
BEGIN
  INPUT 1 clamp2( sIN  (-1), cIN (c:8 c:3..7),
                  cOUT (1 2 1:2..4 1:1) );
  c 1 @ 0.9*<c:1>+0.1*<c:2> -s s  -L c:1 1:1;
  c 2 @ 0.9*<c:2>+0.1*<c:1> -s s  -L c:2 1:2 1;
  x e c:1 c:2;
END

COMPONENT clamps(sIN s, cIN c[1..6])
BEGIN
  INPUT 1 clamp2(sIN  (s), cIN  (c:1..6),
                           cOUT (1 2 1:2..4 1:1));
  INPUT 2 clamp (sIN  (s), cIN  (1:1..3 1:6),
                           cOUT (1..4) );
  INPUT 3 clamp3(sIN  (s), cIN  (2:1..3 1:3..6 2:4));
END
```

The topologies are shown in Figure 5.5. Here, `clamp2` is built upon `clamp` and `clamp3` is built upon `clamp2`. Without much trouble, a more complicated component `clamps` is built upon `clamp`, `clamp2` and `clamp3`.

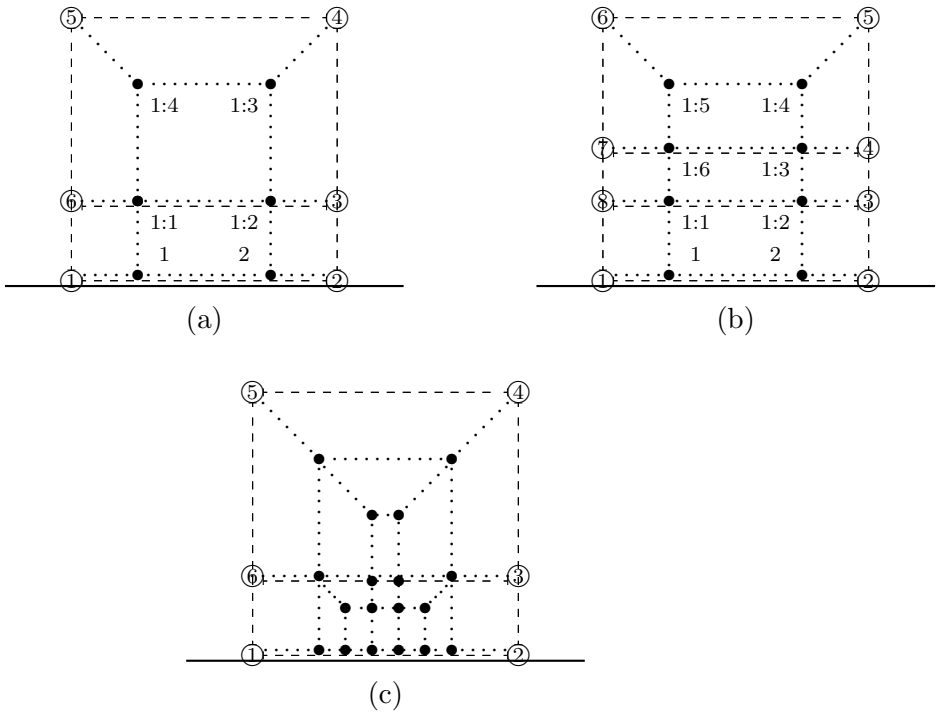


Figure 5.5: Compact enrichment for surface blocks. (a) Component `clamp2`.(b) Component `clamp3`. (c) Component `clamps`.

In fact, compact enrichment for a case of two blocks with one on a surface can have virtually infinite possibilities without severing the singularity degree of imported corners.

The general procedure for the topology design is:

1) Design a simple topology first and generate a grid without any clustering. You also want to place strategically regular corners on sensitive regions for future compact enrichment. An ideal situation is to be able to grab two blocks for each case of compact enrichment since it allows for many levels of compact enrichment as shown in the example above.

2) Inspect the grid generated, then place in local topologies for compact enrichment. Probably the most used local topologies for compact enrichment are those clamps discussed. You may create many different varieties of them. Now, you can generate a grid again. If you are not satisfied, adjust the compact enrichments until a grid to your liking is generated.

3) Turn on all the clusterings, and continue the run until the final grid is generated.

In this process, changes of edge grid densities are used in every step.

5.3 Using Internal Surfaces

Using additional internal surfaces, other than those defined with the original gridding problem, is a good means to satisfy the user requirements of grid point distribution with great robustness. This method requires the construction of extra surfaces as a trade off for a more complex topology. These internal surfaces can mostly be free formed without demanding much carefulness except for certain portions of it, such as those places that are near the sharp tailing edge of an air wing.

In general, whenever, the grid in a region is over stretched or the geometry contains great disparity in length scales and curvatures, internal surfaces among other means can be used to

separate the regions of disparity and get better grid distribution.

5.4 Better Schedules

Schedule is used mainly to speed up the convergence rate.

5.4.1 Poor man's multigrid

A multi-grid version of the grid solver is under consideration for future releases of GridPro. Until then, the speed can be a concern for large scale grid generation. A grid for Boeing 747-200 that contains about 600,000 grid points may take 15 CPU hours to generate on a IBM/RS6000 320h machine. The performance of the 320h machine is approximately 10 MFLOPS.

However, the ability to change edge grid densities dynamically in the schedule relieves the problem to a certain degree. A poor man's multigrid can be used to speed the convergence. In this strategy, one can run a number of sweeps for the coarsest grid possible for the problem. Then, gradually one increases the densities to the desired level.

5.4.2 Scheduling user specified block acceleration

The `-a` action can be used to accelerate the convergence of individual blocks, if so desired. This is particularly helpful on blocks where some block converges more slowly than the rest due to small cell scales.

Chapter 6

Utilities for GridPro

The utility codes that come with GridPro can be used to analyze, extract, convert, print and weld data or grids. The utilities are shipped only to provide the users some convenience. Also note that, for different reasons, some of the utilities listed below may not be shipped with a particular shipment. The az-Graphic Manager is briefly introduced in the next chapter and is documented in detail in a separate manual volume.

Before we proceed further, some general notes are in order:

1 Unless explicitly specified, the data format for grid in these utilities is the simple point format (GridPro format) used in GridPro for both 2d and 3d cases.

2 Some of the utilities use the connectivity information of blocks.It should be in the same format as GridPro would generate.

3 "command_name <ret>" shows the help message for the command in most of the cases.

The following are brief descriptions of the utilities:

6.1 Surface generating and restructuring tools

6.1.1 Controlnetsurf tool

Usage : controlnetsurf < $QuadFileName$ > [Options]

Description :

This program converts a given control net into a surface. There are schemes to do the same. The control net is specified via a quad file.

The three schemes available are

1 Interpolating Scheme

This scheme generates an interpolating surface through the given set of control points However, this would not do a good job if you want to generate a more tube like or sphere like surface. One would end up with regions of relatively tight curvature around the regions where the control net surface has a large change in normal. If you would like a interpolating scheme with a more gentle curvature, then you should think of using the sphere like scheme, which is also an interpolating scheme.

2 Approximating scheme

This scheme generates a surface following the general shape of the control net. The options-NormalsFile and -FrozenCornersFile are not valid for this scheme. However, if they are specified,they ignored.

3 Spherelike scheme

This scheme also generates an interpolating surface but with gentler curvatures as against the first scheme. If you are looking generate a tube like or sphere like interpolating surfaces, this is a better scheme than the first one

-Sid -SchemeId=< 1|2|3 >

Specify the appropriate scheme from the above list. The default is 3

-l -NumberOfLevelsOfRefinement=[1..10]

Specify the number of levels of refinement. Please note that the 'n' levels of refinement generates $4\hat{}n$ times the number of initial number of faces. The default value is 5.

-nf -NormalsFile

This option can be used to specify the tangent planes to the surface at various control points on the given control net The format of the file is as follows :

n #Number of Points at which the normals are specified
x y z nx ny nz #Location of the point(x,y,z)
#and normal of the tangent plane at the specified point (nx,ny,nz)
....

-lf -LinesFile

This option can be used to specify the lines along which the corner has to lie.

-F -FrozenLinesFile

This option can be used to specify a list of corners to mark the edges Through which surface has to pass through. i.e. if the end points of an edge appear in this file, then the final surface passes through the edge. If all the corners of a given face appear in this file, then the face would considered 'frozen'. What 'frozen' means is that this part of the surface would not respect the slope continuity along its boundaries. However, if the neighbouring face is not 'frozen', you could end up with a smooth transition or have a crease along the common edge if otherwise.

The format of the file is as follows :

n #Number of frozen corners
x y z #coordinates of the corners
...

-U -UnFrozenCornerFile

This option primarily helps to let the program know that the points in this file can be moved to a location such that the final surface is smoother than it would otherwise be.This option would be handy when you use the corners to specify a surface, but you do not necessarily want the surface to pass through this corner.

-t -tension

Flatness parameter. default is 0

-sm -SmoothType

1 for Spring analogy
2 for laplacian smoothing

-ns -NumberOfSmoothingIterations

Number of times to run the smoothing algorithm on the unfrozen corners of the controlnet

-o -OutPutFileName

Specify the output file name. The default is "tmp.quad"

-e $< QuadFileCreatedViaControlNetSurf >$ -ExtractControlNet
$< QuadFileCreatedViaControlNetSurf >$
Extract the controlnet used to create the surface
-cnnsf $< CornerandNormalSpecFile >$ -CurveAndNormalSpecFile
$< CornerandNormalSpecFile >$

6.1.2 offset tool

Usage : "offset [Options] "

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension'*.fra '.	-
-s	Surface Id	List of surface ids.	None
-sg	Surface Group	Surface group.	None
-or	Offset ratio	Offset ratio	0.1
-ns	Num Smoothings	Number of normal smoothings.	10
-fg	Feature Group	Feature corners of the surfaces	None
-frg	Frozen Group	Frozen corners of the surfaces	None
-ofn	Output File Name	Output file name with extension'*.fra '.	-

Syntax :
"offset -fn $< inputfilename >$ -s $< sid1 >< sid2 >$ ... -sg $< surfacegroup >$
-or $< offset\_ratio >$ -ns $< numsmooths >$ -fg $< featuregroup >$
-frg $< frozengroup >$ -ofn $< outputfilename >$ "
Purpose :
Create an offset of the given surface. It is used to keep the grid points close to the geometry.
Example : offset -fn airfoil.fra -s 1 -or 0.1 -ns 4 -ofn offset_out.fra
Applications :
1. To keep the grid points close to the geometry

6.1.3 cap_tube tool

Usage : "cap_tube [Options] "

Options	Expansion	Description	Default value
-ifn	Input File Name	The name of the input tube file with an extension '*.tube'.	-
-ofn	Output File Name	The name of the output tube file with an extension '*.tube'	-

Syntax :
"cap_tube -ifn $< inputfile >$ -ofn $< outputfile >$"
Purpose :
Close the tube on both sides.
Example : cap_tube -ifn open.tube -ofn closed.tube

Notes :

1 It is the utility that can be used to close both the open ends of the tube. It works only on surfaces that are created using 'make tube 'command. Please refer the utilities help pdf file to understand more about its uses.

6.1.4 gen_curve tool

Usage : "gen_curve [Options] "

Options	Expansion	Description	Default value
-fn	File Name	Input file name with the extension '*.fra '.	-
-rg	Reference Group Id	The reference corner group id.	None
-g	Group Id	The corner group from which the curve has to be generated.	None
-p	Prefix	Prefix to the newly created curves. i.e.Surface label	$\_new\_surf\_$
-ui	Use Interpolation	Use interpolating scheme. This makes the curve to pass through all the corners.	Approximating scheme
-nr	Num Refinements	The number of refinements.	4
-rt	Remove topology	The topology used for generating the curves will be removed after the curvecreation.	False

Syntax :

"gen_curve -fn $< inputfilename >$ -rg $< referencegroupid >$ -g $< gid >$ -p $< prefix >$ -ui -nr $< numofrefinements >$ -rt"

Purpose :

Create linear curves from the given topology.

Example : gen_curve -fn curve.fra -g 1 -p curve -nr 3

Applications :

1 To create internal surface for sharp feature geometries in 2D.

2 To create internal surface for refining a particular area of the grid.

3 To create 2D geometries in GridPro.

6.1.5 feature_edge tool

Usage : "feature_edge [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with the extension '*.fra '.	-
-s	Surface Id	List of surface ids.	None
-ta	Threshold Angle	The threshold angle for feature edges.	30
-ib	Include Boundary	On the boundary of the surface, it links the corners and forms an edge.	FALSE
-ofn	Output File Name	Output file name with the extension '*.fra '.	-

Syntax:

"feature_edge -fn < $inputfilename$ > -s < $surface\_id$ > -ta < $angle$ > -ib -ofn < $outputfilename$ >"

Purpose:

Create corners on the surface based on the feature angle.

Example: feature_edge -fn wing.fra -s 1 -ta 40 -ib -ofn wing.feature_out.fra

Notes:

1 It calculates the feature angle of each node on the surface and creates corners at the nodes wherever it exceeds the given feature angle.

Applications:

1 To capture the sharp features of the geometry while creating the internal surface.

6.1.6 ribbon tool

Usage : "ribbon [Options] "

Options	Expansion	Description	Default value
-fn	File NameInput	file name with extension '*.fra'.	-
-pg	Path Group	The corners in the group should be assigned to at least one polysurface.	None
-sg	Special Group	The corners in the group retain their normal orientation.	None
-isg	Invert Surface Group	The corners in the group retain their normal orientation.	None
-w	Width	Ribbon Width.	0.01
-ns	Num Smooths	Number of levels of Laplace smoothing.	1000
-ofn	Output File Name	Output file name with extension '*.fra '.	-

Syntax :

"ribbon -fn < $inputfilename$ > -pg < $pathgroup$ > -sg < $specialgroup$ > -isg < $invertsurfacegroup$ > -w < $width$ > -ns < $numsmooths$ > -ofn < $outputfilename$ >"

Purpose :

Create a layer of corners either normally inwards or normally outwards to the given set of corners with the given width, based on the given corners and its assignments.

Example : ribbon -fn wing_ribbon.fra -pg 1 -w 0.2 -ns 1000 -ofn wing_ribbon_out.fra

Applications :

1 To create an internal surface which orthogonally cuts through the sharp feature of the geometry

6.1.7 intersection tool

Usage : "intersection [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-is	Intersecting Surfaces	Specify the ids of all intersecting surfaces.Intersecting surfaces are automatically evaluated.	None
-sp	Surface Pairs	Pairs of intersecting surfaces. E.g. 1 3 0 10	None
-ofn	Output File Name	Output file name with extension '*.fra '.	-

Syntax :

"intersection -fn < $inputfilename$ > -is < $sid1$ >< $sid2$ > ... -sp < $sp1_1$ > < $sp1_2$ >< $sp2_1$ > < $sp2_2$ > ... -ofn < $outputfilename$ >"

Purpose :

Create corners on the intersection of the given surfaces.

Example : intersection -fn sphere.fra -is 1 2 -ofn sphere_out.fra

Notes :

1.If there are multiple intersections, it will output only one set of intersection corners.

Applications :

1 To capture the sharp features of the geometry while creating internal surface.

2 This command can be used when you have multiple surfaces intersecting on the sharp feature.

6.1.8 ribbon_nest tool

Usage : "ribbon_nest [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-ng	Nest Group	The group_id which contains the topology to be nested.	None
-sg	Special group	The corners in this group are simply wrapped out without nesting.	None
-rg	Ribbon Group	The group id which contains the ribbon.	None
-nr	Num Refinements	The number of levels of refinement.	0(Max possible)
-r	Ratio	The ratio of extrusion.	1
-og	Outer Group	The outer corners are added to the group.	None
-lg	Length Group	The corners in this group will have fixed length.	None
-nls	Num Length Smooths	The number of length smoothings.	1000
-awl	Add Wrap Layer	Add a wrap layer at the end.	False
-ofn	Output File Name	Output file name with extension '*.fra'.	-

Syntax :

"ribbon_nest -fn < $inputfilename$ > -ng < $nestgroupid$ > -sg < $splgroupid$ > -rg < $ribbongroupid$ > -nr < $numrefinements$ > -r < $ratio$ > -og < $outergroupid$ > -lg < $lengthgroupid$ > -nls < $numlengthsmoothings$ > -awl -ofn < $outputfilename$ >"

Purpose :

If the number of corners on the surface is more, it will consume more amount of time to build the wireframe for the internal surface. In such cases, reverse nest can be used and reduce the number of corners. It creates given number of layer of corners. The number of corners reduces with each layer.

Example : ribbon_nest -fn wing_nest.fra -ng 1 -sg 3 -rg 2 -nr 2 -r 1 -nls 1000 -awl -ofn wing_nest_out.fra

Notes :

1 The outer layer of corners should be given as ribbon group because from which the nesting starts.

2 All the corners should be grouped and given as nesting group.

3 Corners which are at sharp turns or higher feature angle should be given as special group in order to avoid nesting on those corners.

Applications:

1 To reduce the number of corners, while creating the wireframe for the internal surface.

2 To reduce the number of corners in the far field while creating topology for a 2D geometry. This gives a fine grid near the geometry and coarse grid in the far field.

6.1.9 smooth_tube tool

Usage : "smooth_tube"
Syntax :
"smooth_tube $< inputfilename > < outputfilename >$
$< numberoflevelofrefinements >$"
Purpose :
Smoothen the tube file. All the sharp features on the curved region of a tube can be smoothened using this command.
Example : smooth_tube smooth.tube smooth_out.tube 3
Notes :
1 It is valid only for '*.tube' files.

6.1.10 refine tool

Usage : "refine"
Syntax :
"gp_utilities refine $< inputtriafilename > < numberofrefinements >$
$< outputtriafilename >$"
Purpose :
Refine the triangulation of a '*.tria' file.
Example : refine wing_tip.tria 5 wing_tip_refine.tria

6.1.11 mrgn tool

Usage : mrgn fn [-t tol] [-o] $< ret >$
Purpose : merge (equivalent) nodes and cells for TRIA, STL,
QUAD or HEXA data by tolerance.
Input : 'fn' −− unstructure tria, quad or hexa in GridPro format.
Output : 'fn.tmp' −− node merged data in the same format.
Options :
-t tol −−(default 0.001) tolerance relative to local cell scale.
-ts tol −−(default 1e-20) skewness tolerance. That is, all cells with skewness worse than tol will be eliminated.
-tc tol −−(default 0.0) cell size tolerance. That is, all cells with cell size (shortest edge for tria) less than tol will be eliminated and the proper nodes will be merged.
-o −−(default output all) no output.
-azm [fn] −−(default no stamp) stamp message to fn or '_stamp.tmp'.

6.1.12 smg tool

An important use of 'smg' is to round surfaces for inputing to az3000. It uses home-developed variational algorithms for smoothing grid data. It can also be used to give a final touch to a grid generated by az3000 to improve the curvature statistics. In many cases, applying only a few cycles of 'smg' can have dramatic improvement on the grid qualities. The quality measures that 'smg' can improve are curvature, warpage, angle differential, and length differential. Since the algorithm is variationally based in a very general sense, the running of 'smg' may be slow.

Usage : smg fn cycles options $< ret >$
Purpose : smooth grids defined by line seg, tria or quad, or hexa elements.
Output : 'fn.tmp'.
Options :

-r num −− [0 .. 1 (default 1/3)]. set relaxation constant.
-t num −− [> 0 (default 0.000001)]. set lower cutoff for the variational quntity. NOTE: curvature sensitivity.
-c num −− [-1 .. 1 (default 0)]. set cluster strength.(not for hexa)
-wc num −− [0 .. 1 (default 1)]. set curvature weight.
-wa num −− [0 .. 1 (default 0)]. set angle diff weight.
-wl num −− [0 .. 1 (default 0)]. set length diff weight.
-tc num −− [0..180 (default 10)]. set sensitive threshold for curvature.
-ta num −− [0..1 (default 0)]. set sensitive threshold for angle diff.
-tl num −− [0..1 (default 0)]. set sensitive threshold for length diff.
-pc num −− [0..8 (default 1)]. set nonlinearity for curvature.
-pa num −− [0..8 (default 1)]. set nonlinearity for angle diff.
-pl num −− [0..8 (default 1)]. set nonlinearity for length diff.
-d −− (default off). show detailed display.
-b −− (default off). set periodic bc for curve.
-s [num] (for hexa only)
−−(default free boundary). -s or -s 1 → respect boundary.
-s 2 → fix boundary.
NOTE: for fixed bc, curvature on surf is not counted.
-W num −−(default at the end). output period.

NOTE: 1/4M nodes + 1/4M hexs → 85 MB RAM on a IBM RS6000 320h

6.1.13 segn tool

Usage : segn fn [options]$< ret >$
Purpose : 1) segment surf that is branching or non-orientable.
2) segment surf by planes and cylinders.
Input : 'fn' −− unstructure tria or quad in GridPro format.
Output : 'fn.tmp' −− node seged data in the same format.
'_seg.?'−− if -o is on.
Options :
-o [num] −−output surf pieces in separarte files '_seg.?' for every piece that has cell count > num.

default num = 1.

-c x y z r −− if $r > 0 (< 0)$ output all cells without(within) the cylinder of radius |r| about axis from (0,0,0) to (x,y,z)

-p x0 y0 z0 xn yn zn
−−output all cells on the +side of the plane defined by (X-x0)*xn+(Y-y0)*yn+(Z-z0)*zn = 0

-x num −−output all cells with x <= num

+x num −−output all cells with x >= num

-y num −−output all cells with y <= num

+y num −−output all cells with y >= num

-z num −−output all cells with z <= num

+z num −−output all cells with z >= num

-rx num −−output all cells with sign(num)*sqrt(y*y+z*z) >= num

-ry num −−output all cells with sign(num)*sqrt(x*x+z*z) >= num

-rz num −−output all cells with sign(num)*sqrt(y*y+x*x) >= num

-C −−output the complement cells.(has effect only with constraint options.

NOTE: 1) If more than one of -c -p -x +x.. exist, cells satisfying all constraints are outputted.
2) No more than 16 of constraints can exist.

6.1.14 gencv tool

Generating a smooth digitized curve from a set of discreet corners with the control of a cluster parameter and the number of data points for each of the pair of successive corners.

Purpose : using lines and arcs to gen smooth curve.
Usage : gencv fn filterCount < *ret* >
filterCount − − − >= 0
fn − − − corner data (for line segs and arcs
Format:

```
line  1 : I 1 1                    # dim of corner data
line  2 : x1 y1 z1 [xt1 yt1 zt1]   # coords of 1st corner
            ...
line I+1: xI yI zI [xtI ytI ztI]   # coords of Ith corner
line I+2: n1 d1
            ...
line 2I+2: nI dI
```

NOTES:1) If xti exist in a corner line, an arc with giving tangent (xti,yti,zti) and passing (i+1)th corner is draw.
2) ni — No.of pts for the ith line seg.
3) nI — not used.
4) di — cluster strenth for the ith corner. distribution x**di
5) utility smg is called to do the smoothing.

6.1.15 thin tool

Usage : thin fn [-t dd] [-r dd] [-i num] [-s dd]< *ret* >
Purpose : thin or sparse out tria surface representation.
Input : 'fn'−− unstructure tria GridPro format.
Output : 'fn.tmp' −− thinned data in the same format.
Options :
-n d −−(def= 5.0 degree, [0..45] degrees) tolerance on cell normal differential.
-r [d] −−(def=auto,[0..1.0], def d=0.1) cells reduction ratio.
-i num −−(def=auto,[>= 0]) iterations.
-s d −−(def=10 degrees,[0..60] degrees) cell shape (min angle) limiter.
-S −−(def=OFF) turn on cell edge swap.
-b −−(def=NO) preserve border nodes.
Note : if -r is set -i is ignored
Description:
Sequencially each node and its surrounding trias are tested for removal-ability against a set of replacement cells (with the node removed). A node will be removed and the cells replaced if the test satisfies the following criterias:
1) The normals of the old and new cells do not differ more than a given degrees.(the -n flag).
2) The worst angle of all new cells is better than either that of the old cells, or a given number. (the -s flag).
3) A test sweep of all nodes is called an iteration. The removal iteration continues until i). with the the -r flag off, either a preset iteration limiter is reached (the -i flag) or no node can be further removed; or ii). with the -r flag on, either the cell reduction ratio, or the cell normal differential limiter is reached.

6.1.16 xsec tool

Generating intersection curves between a set of 2d multiblock data and a set of planes. The resulting data can be used for viewing with **az**.

'xsec' computes the planer cross-sectional curves of 3d surfaces. Multiple cutting planes can be specified. An important use of 'xsec' is to generate cross-sectional curves for viewing the general shapes of surfaces and for setting initial corner positions for TIL programs.

Purpose : get the planer cross section of digitized surf.
Usage : xsec fn options < *ret* >
Input : native GridPro quad, tria or 2d multiblock data.
Output : fn.tmp. native GridPro 1d multiblock data.
Options : at least one, not more than 128
-x nums −− x-section at x = num1, num2, ...
-y nums −− x-section at y = num1, num2, ...
-z nums −− x-section at z = num1, num2, ...
-p paras −− x-section with plane defined by the paras.
paras = plane_norm plane_point_1 plane_point_2 ...(>= 6 reals).
+x x1 x2 num .. −− num of equally spaced x-sections for x = x1 to x2.
+y y1 y2 num .. −− num of equally spaced x-sections for y = y1 to y2.

+z z1 z2 num .. −− num of equally spaced x-sections for z = z1 to z2.
+p norm point1 point2 num ..
−− num of equally spaced x-sections.
+X x1 dx num .. −− num of x-sections of spacing dx starting from x1.
+Y y1 dy num .. −− num of x-sections of spacing dy starting from y1.
+Z z1 dz num .. −− num of x-sections of spacing dz starting from z1.
+P norm point1 dvec num ..
−− num of x-sections of spacing dvec*norm starting from point1.
-b −− output tria belt used for the x-section.
-a [nx [ny nz]]
−− (default nx=5) automatic xsec with nx,ny, and nz for x,y, and z axes respectively

6.2 Data manipulation tools

6.2.1 transform_topo tool

Usage : "transform_topo [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-g	Group Id	Corners in this group will be subjected to rigid body rotation.	None
-sg	Surface Group	Surfaces in this group will be subjected to rigid body rotation.	None
-s	Surface Id	List of surface ids. These surfaces will be subjected to rigid body rotation.	None
-t1	Tanslation Begin	This translation is applied before rotation.	0 0 0
-sc	Scaling	The scaling wrt origin.	1
-a	Angle	The angle of rotation.	0
-ax	Axis	The axis. The coordinates of centre followed by axis direction.	0 0 0 0 0 1
-t2	Translation End	This translation is applied after rotation.	0 0 0
-m	Mirror	The mirror plane coordinates. The coordinates of a point on the plane followed by its normal.	False
-ofn	Output File Name	Output file name with extension '*.fra'.	-

Syntax :

"transform_topo -fn < $inputfilename$ > -g < gid > -sg < $surfacegroup$ > -s < $listofsurfaceids$ >-t1 < $coordinates$ > -sc < $scalingratio$ > -a < $angle$ > -ax < $centre\&normal$ > -t2 < $coordinates$ > -m < $centre\&normal$ > -ofn < $outputfilename$ > "

Purpose :

Rotate, transform and mirror either a given topology, surfaces or both topology and surfaces.

Example :

Applying Transformation: transform_topo -fn airfoil.fra -g 1 -s 1 -t1 0 0.5 0 -sc 1 -ax 0 0 0 1 0 0 -ofn airfoil.transform_out.fra.

Applying Rotation: transform_topo -fn airfoil.fra -g 1 -s 1 -ax 0 0 0 0 0 1 -a 45 -ofn airfoil.rotate_out.fra.

Applying Mirroring: transform_topo -fn airfoil.fra -g 1 -s 1 -m 0.25 0 0 1 0 0 -ofn airfoil.mirror_out.fra.

Applications:

1.To optimize a design.

6.2.2 trf tool

Transforming the multiblock data. It can be used to output blocks a). with translation and rotation, b). sparsed from the original blocks with given sparse ratios, and c). with gaps between neighboring blocks according to given gap parameters.

Usage : trf fn options < ret >

Purpose : transforming multiblock or -tube or -tria or -quad data.

Options :

-t x y z —— translation on data.

-r x1 x2 x3 y1 y2 y3 z1 z2 z3
—— linearly transform data.
Xnew = x1*Xold + x2*Yold + x3*Zold
Ynew = y1*Xold + y2*Yold + y3*Zold
Znew = z1*Xold + z2*Yold + z3*Zold

-rx theta —— rotate theta degrees around x axis.

-ry theta —— rotate theta degrees around y axis.

-rz theta —— rotate theta degrees around z axis.

-xyz2xrt [scale] —— (x,y,z) to (x,r,theta) convertion.

-xyz2yrt [scale] —— (x,y,z) to (y,r,theta) convertion.

-xyz2zrt [scale] —— (x,y,z) to (z,r,theta) convertion.

-xrt2xyz [scale] —— (x,y,z) to (x,r,theta) convertion.

-yrt2xyz [scale] —— (x,y,z) to (y,r,theta) convertion.

-zrt2xyz [scale] —— (x,y,z) to (z,r,theta) convertion.
scale is the scaling factor on theta.

-D di dj dk —— output data idx steps(sparse output)

-d ratio [back] —— output sparse data by interpolation

-d ri rj rk [back] —— output sparse data by interpolation

-m lold1 lnew1 .. —— output sparse data with length maps (>=2)

-b bid1 i,j,k lnew1 ..
−−output sparse data with given block id,bid (>= 1), an index direction, i or j or k,and the new cell density,lnew (>1 for absolute density and (> 0.0, < 1.0) for relative desity). If conflict occurs, the smallest is in effect.lnew is the grid point count, not the cell count.
-s −− output precision=single. (default=double)
-kji −− output with index switched,(FORTRAN< − >C order) for block grid.
-O fn −− output to fn (and fn.conn).

Descriptions:

'trf' accepts a sequence of mixed and multiple '-t -r -rx -ry -rz' options. They act on the data from left to right.

'trf' has three main functions that can be combined in the command line:

1). linearly transform grid position data.
options: -t -r -rx -ry -rz

2). generate sparsed grid data with given length maps.
options: -D -d -m -b. Do not use them in mix.
If conflict occurs, the smallest is in effect.
Works only for elementary multiblock data.

3). output single precision grid data to save storage.
options: -s

6.2.3 siz tool

Calculating the size of the multiblock data. It can also output blocks with a). switched index order, b). reversing index directions, and/or c). larger than one increment in the index variables. 'siz' computes the physical size of the input data. By physical size, we mean x_min, x_max, y_min, y_max, z_min, and z_max. The input data may be multiblock data or unstructured hex, quad, or tria data. One can also use 'siz' to output user specified blocks with user specified axis switchings.

Usage : siz fn1 [fn2..] [-i i0 i1] [-j j0 j1] [-k k0 k1] [-o bid order]< *ret* >
Purpose : check the physical size of data.
Input : fn1 ..−− grid data in GridPro formats.
Options :
-i i0 i1 −−scan i range from i0 to i1, (i1¡i0 is ok; default 0 to I-1).
-j j0 j1 −−scan j range from j0 to j1, ..
-k k0 k1 −−scan k range from k0 to k1, ..
-o bid1 bid2.. order
−−output data set 'bid1..' in the 1st input file to 'siz.tmp' the data with the order and ranges. 1 <= *bid*1 < *bid*2.. -i,-j,-k flags defined the ranges order = ijk, ikj, jik, jki, kij or kji.
-d −− display per file info.
-D −− display per block info.

6.2.4 replb tool

Replace a block in multiblock data.

Usage : replb fn1 fn2 bid1 bid2...

fn1 −− main block data file

fn2 −− new block data file (¡=1024 blks)

bid1 bid2.. −− in output file 'fn1.tmp', data set bid_i of fn1 is replaced by data set i of fn2.. 1<=bid1<bid2..

Output : fn1.tmp−− new grid file in GridPro format.

6.2.5 shuffle_corners tool

Usage : "shuffle_corners [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with the extension '*.fra'.	-
-ns	Num of Shuffles	The number of shuffles.	5
-sfn	Shuffle File Name	Output file name with extension '*.fra'.	_az.out.fra

Syntax :

"shuffle_corners -fn < $inputfilename$ > -ns < $numofshuffles$ > -sfn < $outputfilename$ >"

Purpose :

Shuffle the corner id 's of the given topology.

Example : shuffle_corners -fn az.fra -ns 30 -sfn shuffle_out.fra

Applications :

1. To resolve the error 'Incomplete Molecule '.

6.3 Extraction and duplication tools

6.3.1 cart_prod tool

Usage : "cart_prod [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'	-
-mg	Master Group Id	The group id consists of the corners to which the topology has to be duplicated.	None
-sg	Slave Group Id	The group id consists of the topology which is to be duplicated.	None
-ofn	Output File Name	Output file name with extension '*.fra'	-

Syntax :

"cart_prod -fn < $inputfilename$ > -mg < $mastergroupid$ > -sg < $slavegroupid$ > -ofn < $outputfilename$ >"

Purpose :

Duplicate a topology to a given location.

Example : cart_prod -fn cartesian.fra -mg 2 -sg 1 -ofn cartesian_out.fra

Notes :

The utility is used to duplicate the topology at desired locations, the duplicated topologies will be individual instances which need to be merged or linked by the user. The process is executed using two different groups 1. Master Group, 2. Slave Group. The master group contains the topology corners which define the locations where it has to be duplicated and the slave group contains the topology to be duplicated.

Important :

1. Once the slave corners are duplicated to the master corner's position, the master corners should be deleted manually by the user.
2. The topology will be duplicated, such that the center of the slave corner group merges with the master corners.

To understand more on the application of the utility please refer the utilities help pdf document.

6.3.2 periodic2topo tool

Usage : "periodic2topo [Options]"

Purpose :

Duplicate the periodic topology using the given periodicity and outputs a full valid topology.

Options	Expansion	Description	Default value
-fn	File name	Input file name with extension '*.fra'	-
-ofn	Output file name	Output file name with extension '*.fra'.	-

Syntax :

"periodic2topo -fn < $inputfilename$ > -ofn < $outputfilename$ >"

Example : periodic2topo -fn periodic.fra -ofn periodic_out.fra

6.3.3 rotate tool

Usage : "rotate [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-g	Group Id	The group id which contains the topology sheet to be rotated.	None
-max	Maximum Angle	The max angle of rotation.	270
-min	Minimum Angle	The min angle of rotation.	90
-ni	Num Instances	Specify the number of instances (or copies) of the topology sheet to be created.Equi-distant instances are created based on the max-min angles. Note that -i option should not be used if it is specified.	None
-i	Instances	The instances. The angles (in degrees) should be specified. Note that -ni option should not be used if it is specified.	None
-sc	Self-Closed	The topology will be looped.	False
-a	Axis	The axis. The coordinates of centre followed by the axis direction.	0 0 0 0 0 1
-p	Pitch	The pitch distance. If the pitch distance is given, then it forms a helix structure.	0
-ofn	Output File Name	Output file name with extension '*.fra'.	-

Syntax :

"rotate -fn < $inputfilename$ > -g < $groupid$ > -max < $maxangle$ > -min < $minangle$ > -ni < $numofinstances$ > -i< $instances$ > -sc -a < $centreand normal$ > -p < $pitch$ > -ofn < $outputfilename$ > "

Purpose :

Create a rotated topology for the given topology using the angle, no. of instance and pitch.

Example :

Rotation without pitch: rotate -fn topo.fra -g 1 -max 300 -min 0 -ni 6 -sc -a 0 0 0 0 1 0 -ofn hex_out.fra

Rotation with pitch: rotate -fn topo.fra -g 1 -max 300 -min 0 -ni 6 -sc -a 0 0 0 0 1 0 -ofn hex_out.fra

6.4 Topology Optimisation tools

6.4.1 reverse_nest tool

Usage : "reverse_nest [Options]"
Purpose :

Reduce the number of corners in the topology as it moves away from the geometry.

Options	Expansion	Description	Default value
-fn	Input File Name	Input file name with the extension '*.fra'.	-
-all	All Group	The group id which contains the topology to be nested.	None
-sg	Special Group Id	The group id which contains the topology to be wrapped out without nesting.	None
-ne	Num Extrusions	The number of extrusion of reverse nesting.If it is zero, topology is extruded to the maximum extent possible.	0
-r	Ratio	The ratio of extrusion. Used for positioning of nestedcorners.	0.1
-es	Extrude Spherically	Assumes that the outer topology is assigned to sphere and extrude in its normal direction.	False
-ofn	Output File Name	Output file name with extension '*.fra'.	-

Syntax :

"reverse_nest -fn $< inputfilename >$ -ag $< allgroup >$ -sg $< specialgroupid >$ -ne $< numextrusions >$ -r $< ratioofextrusion >$ -es -ofn $< outputfilename >$"

Example: reverse_nest -fn reverse_nest.fra -ag 2 -sg 3 -ne 4 -r 0.5 -ofn reverese_nest_out.fra

Notes :

1. It helps in refine the grid near the geometry without affecting the far field. The grid near the geometry is fine and coarse in the far field.

6.5 Grid enhancing tools

6.5.1 autofix tool

Usage : "autofix [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-g	Group Id	Retain the singularities in the group.	None
-cid	Concave Edges Group Id	Append all the concave mildly severe edges to this group.	None
-t	Type	Eliminate singularities of type Very, Mediumly and Mildly.Type & (1 << 0)⇒ eliminate very severe singularities. Type & (1 << 1) ⇒ eliminate mediumly severe singularities.Type & (1 << 2)⇒ eliminate mildly severe singularities.	3
-eb	Ensure Buffer Layer	Ensures buffer layer is created.	-
-ofn	Output File Name	Output file name with extension '*.fra'.	-

Syntax :

"autofix -fn < $inputfilename$ > -t < $typeofsingularitytobeeliminated$ > -g < $group\_id$ > -cid < $concaveedgesgroupid$ > -eb -ofn < $outputfilename$ >"

Purpose :

Solve all the Mildly, Medium and Very severe singularity automatically.

Notes :

1. The tool solves all the 3 singularities by picking appropriate sheets. The condition for the code to run is that the input topology given should have a buffer layer of topology. To understand more on this please refer to the tutorial document on "autofix".

Important :

The code solves mildly severe singularity only at the topology level, it can not build the right surfaces. However the code creates some fictitious surfaces to check whether the topology is a valid one.The surfaces are created with a prefix "_ _new_surf XX ",where the XX denotes the surface number. It is highly recommended to delete the surfaces with the prefixes from the work ing directory and create new surfaces which would align the grid in a smooth pattern. After the deletion of the surfaces with the prefix, the solved topology remains a mildly severe topology with the right sheets to be assigned to the surface.

Example :

Mildly severe: autofix -fn mildly_severe.fra -t 4 -eb -ofn mildly_severe_out.fra
Mediumly severe: autofix -fn medium_severe.fra -t 2 -eb 3-ofn medium_severe_out.fra
Very severe: autofix -fn very_severe.fra -t 1 -eb 3-ofn very_severe_out.fra

6.5.2 enrich tool

Usage : "enrich [Options]"
Purpose :
Refine the grid in a particular area by modifying the topology.

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'	-
-or	Offset Ratio	Offset Ratio.	0.01
-fg	Feature Group	Feature corners of the surfaces	None
-ofn	Output File Name	Output file name with extension '*.fra'	-

Syntax :
"compact_enrichment -fn $< inputfilename >$ -or $< offset\_ratio >$ -fg $< featuregroup >$ -ofn $< outputfilename >$"
Example : enrich -fn car.fra -or 0.01 -fg 1 -ofn car_out.fra
Note : This tool is a powerful tool due to its flexibility, it can create compact enrichment by doing internal wraps on the topology sheets provided.

6.6 Grid tools

6.6.1 cutg tool

Usage : cutg fn options $< ret >$
Purpose : cut out or relabel elements bounded by planes. It cuts out a grid in a convex hull of a simplex or its complement.
Input : GridPro tria, quad, or hexa formats.
Output : 'fn.tmp' with the same format as input.
Options:
-x x0 x1 −− define bounded region by x0 < x < x1
-y y0 y1 −− define bounded region by y0 < y < y1
-z z0 z1 −− define bounded region by z0 < z < z1
-p plane_paras
−− define bounded region by the +side of a set of planes.(simplex).
plane_paras is a set of 6 reals defining norm_vec and a_plane_point of the plane.
-o num −−(default = -o 12) output a portion of elements.
num = 1, 2, 3, 12, 13, 23, or 123... −− the region selector each digit indicates a region to output.
1 = in the simplex, 2 = on the boundary, 3 = out the simplex.
-l pid1 pid2 pid3
−− (default = use input data pid's) label regions with pid's.pid1, pid2 and pid3 are non-negative integers labelling the bounded, the bounary and the complement regions respectively.

Notes : 1). multiple (<1024) planes can be given (simplex).
2). multiple use of cutg can cut out any plane bounded shape.

6.6.2 disjoint_grid tool

Usage : "disjoint_grid [Options]"
Purpose :
Run two distinct valid topologies as a single file and output as a single grid.

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra '.	-
-ns	Num sweeps	Number of sweeps.	1000
-ogn	Output Grid File Name	Output grid file name with extension "*.tmp ". Connectivity file is auto generated.	-

Syntax :
"disjoint_grid -fn < $inputfilename$ > -ns < $numofsweeps$ > -ogn < $outputgridfilename$ >"
Example : disjoint_grid -fn disjoint.fra -ns 500 -ogn blk.tmp
Notes :
1) The individual topologies should be valid topologies. The two files have to be loaded into az and saved as a single file. The resultant disjoint file should be run with this tool in order to obtain a single grid file.

6.6.3 getg tool

Extracting grid data from multiblock grid data using connectivity information. The output can be feed to **az**. It helps to speedup and localize the display of **az**.The typical applications area). extracting surface grid sheets by giving the surface ids,b). extracting a block sheet (or a grid sheet) by giving a block id and a axis id,c). extracting blocks that are whithin 0, 1, 2, or 3 hops from a given block or surface,and d). extracting the wire frame of the blocks.

The 'getg' utility extracts grid from multiblock grid data in the GridPro format. The connectivity file must exist. The output is also a multiblock grid data. However the connectivity is not generated. Also the output may be in lower dimensions (e.g. from a volume grid to a surface grid). One can use 'getg' to isolate a portion of a grid of interest for display or other uses. A sheet of blocks is specified by a block and an axis of that block. Through axis maps between blocks, every block can define one or more axes corresponding to the given axis of the given block. The block sheet is a span of blocks starting from the given block in the two index axes other than the ones corresponding to the given axis of the given block.

Usage : getg fn options < ret >

Purpose : extract grids ('conn.tmp' or fn '.conn' must exist).
Input : fn −− GridPro multiblock format.
Output : fn.tmp −−GridPro multiblock format.
Options :

-b bid(=0..) axis(=0..2) [k(=0.0..1.0 or >= 1)]
−− extract block(grid) sheet with seed block bid and a nor mal axis.With k, a grid sheet is extracted. Let K be the number of grid points along the given axis.The extracted grid is the k*K-th sheet of that axis if k<1.0, or the k-th sheet. if and k>=1 k<=K.

-B bid1 bid2 ... −− extract blocks.

-s range1 range2 .. −− extract surface related grids.

+s range1 range2.. −− extract surface related grids for +side only.

-h [hops] −− with hop blocks(w/ -B or -s opts).

-x range1 range2 .. −− surfs to exclude.

-f −− extract block frame.

-c −− extract complement blocks defined by other options.

-S −− need -s flag on. Extract surface grid with 1st and 2nd distances for TECPLOT display.

Notes : (1) range format = 'num' or 'num1..num2' with num's >= 1.
(2) without options, all surfs are included.
(3) a later option overrides the earlier setting.
(4) if -s is the 1st opt, excluding all is the 0th option.
if -x is the 1st opt, including all is the 0th option.

6.6.4 grid2til tool

Convert an elementary block grid and the .conn file intoa TIL code, so one can smooth the grid with GridPro/Ggrid.Useful for improving grids from other grid generators.

Usage : grid2til fn [options] $< ret >$
Purpose : convert an elementary block grid to a TIL code.
Input : 'fn' −− elementary block data in the GridPro format.
'fn.conn' may or may not exist.
Output : 'fn.tmp.fra' −− the TIL code.
'fn.sf1.tmp'..−− surfaces for the TIL code.
'fn.conn' −− (if not exist) reconstructed .conn file for fn.
Options :

-s num = 1 −− all 1-b connected surfs are combined into a single surf.
(default)2 −− use surfs defined in .conn file or surfs from 'genconn'.
3 −− each face group having surf attachment is on a fixed surf.
4 −− each block face is on a fixed surf.

-t num −− (default=0.01,[0..1]) relative tol. for converting to builtin surf.

-g num −− set default edge grid density for testing.

Limitations :

1). Can't auto handle periodic BC.
2). Works for 3d grid only.
3). A corner can't have more than 24 links.
4). A corner can't have more than 16 surfaces.

See Also: segb, genconn, mrgb.

Description :

'grid2til' is used when one has a structured grid withoutthe corresponding TIL code and still want to use GridPro torelax it. If the given grid is in general multiblock structure 'segb' should be used 1st to segment the data into one with elementary block structure. The TIL code generated from 'grid2til' can be fine tuned using 'az' or just manually edit the TIL code. This mainly involves changing surface partitions and modifying the topology. To run Ggrid the -r flag should be used if the topology has not been changed.

Example:grid2til fn.dat $< ret >$ This will generate the TIL code 'fn.dat.tmp.fra'. Now run, Ggrid fn.dat.tmp.fra -r fn.dat $< ret >$ to get the GridPro relaxed grid.

6.6.5 hex2mb tool

Usage : hex2mb fn $< ret >$
Purpose : reverse engineer a multiblock grid from a unstructured hex grid.
Input : fn −− unstructured hex grid in GridPro format.
NOTE: can't have collapsed cells.
Output : fn.tmp −− multiblock grid in GridPro super block format.
Note : need run segb to generate elementary blocks and the .conn file.
See Also : chfmt, segb, genconn.

6.6.6 hex2emb tool

Usage : "hex2emb [Options]"
Purpose : Convert a hex grid and multi block grid into a multi block grid composed of minimum number of elementary blocks.

Options	Expansion	Description	Default value
-P2d	Property 2D	Do not merge blocks with faces having the property name specified with another block with different property.	-
-P3d	Property 3D	Do not merge blocks with faces having the property name specified with another block with different property.	-
-S	Surface Id	Do not merge blocks with faces assigned to the specified surface to another block whose face has been assigned to a different surface.	-
-L	Label Name	Do not merge blocks whose block/Face labels are different.	-
-I	Internal Surface Id	Do not merge blocks which are separated by internal surface	-
-PA	All Property	Do not merge blocks which have different properties	-
-LA	All Label	Do not merge blocks which have different labels	-
-SA	All Surface	Do not merge blocks which have their faces assigned to different surfaces	-
-IA	All Internal Surface	Do not merge blocks which are separated by any internal surface	-
-mg	Multi Block Grid	Assume that the input file is a MultiBlock Grid	-
-ug	Unstructured Grid	Assume that the input file is a Unstructured Hex Grid	-
-o	Output File Name	Output file name with extension '*.grd'.	-

Syntax :
"hex2emb < $InputGridfilename$ > -P2d < $Propertyname$ > -P3d < $Propertyname$ > -S < $Surfaceid$ > -L < $Labelname$ > -I < $Internalsurfaceid$ > -PA -LA -SA -IA -mg -ug -o < $Outputfilename$ >"
Example : hex2emb blk.tmp -S 2 5 -ug -o output.grd
Notes :

1. The user has to explicitly specify whether the input grid is a multi-block hex or an unstructured hex using the options "-ug" or "-mg".

6.6.7 mkrib

Usage : mkrib fn < *ret* >
Purpose : make a ribbon surf from 3 path lines.
Input : fn −− grid data in GridPro formats.
2 or 3 1d blocks with equal lengths.
Output : rib.tmp

6.6.8 segb

The opposite of **mrgb**. Segment a general multiblock grid into GridPro elementary block grid using a node tolerance spec. The .conn file is also regenerated. In combination with grid2til or GridPro Ggrid, it is useful for working on grids generated by other grid generators.

Usage : segb fn [-t tol] [-b] < *ret* >
Purpose : segment general multi-block data into elementary block data.
.conn .conn_m .conn_n files are reconstructed.

Input : 'fn' −− 1). plot3d multi-block data file, ASCII or binary.
2). GridPro multi-block data file.
Output : 'fn.tmp' −− elementary block data in GridPro format.
'fn.tmp.conn'−− connectivity for 'fn.tmp'.
'fn.tmp.conn_m'−− merge conn for going from 'fn.tmp' to 'fn'.
'fn.conn_n.tmp' −− node conn for 'fn'.
Options :
-b −− (default ASCII data) indicate that the input is binary.
-o mode −− (default output all) output mode
mode=0 −− output everything with segmenting data into elem blocks.
=1 −− do not segment data into elementary blocks.
=2 −− output connectivity for the input data only.
-M −− output .conn_m with details if it is outputted.
-t tol −− (default 0.001) tolerance relative to local cell scale.
-p num −− (default -1) -1=auto,0=single or 1=double. output precision.

6.6.9 split tool

Usage : "split [Options]"

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-s	Surface Id	List of surface ids.	None
-ofn	Output File Name	Output file name with extension '*.fra'.	-

Syntax :
"split -fn < $inputfilename$ > -s < $sid1$ > < $sid2$ > ... -ofn < $outputfilename$ >"

Purpose :

Split the topology into pieces using the given surfaces and outputs a valid topology.

Example : split -fn topo.fra -g 1 -max 300 -min 0 -ni 6 -sc -a 0 0 0 0 1 0 -ofn hex_out.fra

1. This command can be used only on a valid topology.
2. The surfaces which are used for splitting should have corners assigned to it.

Applications :

1. To run gridding process on complex geometries in order to converge faster.

6.6.10 clu tool

Performing clustering on GridPro grids.Reduce the gridding task to

1). Use GridPro only to generate a Euler grid;

2) Then, use **clu** on this Euler grid toproduce more than one viscous grid. Big time saver and high quality.

Usage : clu fn options $< ret >$

Purpose : Algebraically cluster grid points to surfaces for GridPro grid. The number of off wall grid layers is auto calculated to match the spacing specs.

Input : 'fn '−− the GridPro grid file, 'fn.conn' is needed.

Output : 'fn.tmp'−− the GridPro clustered grid.

Options :

-s surf_name spacing [gr [cells]] – spec for surface clustering.

−−surf_name: a surface id ($\geq 1 or \leq -1$) or label defined in the .conn file. A label may represent more than one surfaces. 'all' is a predefined label to mean all 1 sided.surfaces. Note that, if a block face is to be clustered, all the faces in the same face group will be clustered too. ('face group' is explained later).

Examples:

+2 −−cluster to block faces of those face grougs which have faces attached to surface 2.

-2 −−cluster to block faces of those face grougs which have faces attached to surface -2

2 −−cluster to block faces of those extended face groups which have faces attached to surface +2 and -2.

Note, the sid, be it positive or negative, must exist in the .conn file; Otherwise it is ignored. Therefore the sign of the sid is meaningful only for two sided surfaces. For a 2 sided surface, '-s +2 ..' or '-s -2 ..' cluster grids on one side of the surface. '-s 2 ..' clusters grids on two sides of the surface. '-s +2 ..' (or plus '-s -2 .. ') may not be the same as '-s 2 ..' due to the difference of a face group and an extended face group.

−−spacing: (> 0.0) the off wall spacing.

−−gr: (> 1.0,default: global gr) spacing growth ratio.

−−cells: In most cases, one should not need to specify this number. This is the desired cell count in the off wall direction of the wall blocks.'clu' will calculate the needed numbers of cells.The clustering will be performed only if the user specified number here is not too far from the internally calculated number.

-r gr −−(> 1.0,default: 1.2) globally set growth ratio to gr.

Affect only '-s' option followed.

-m n ––(n≥ 1,default 1) for multigrid multiple. The number of cells in the block index direction that is normal to the clustered face will be a multiple of n.

-a n ––(default 2) select an interpolation algorithm.

n=0 → (fast)the linear algorithm with average spacings.

1 → (slow)the smoother algorithm with average spacings.

2 → (slowest)the smoother algorithm with uniform spacings.

3 → (medium)the linear algorithm with uniform spacings.

-k n –– access key.Needed when called from Ggrid.

-S –– output data in single precision. (default double).

-M d –– (default 0.2, >0.0) moment used to calculate average spacing.

-fix n –– (default 1, ≥1) off-wall layers that have the same spacings. Affect only -s specs after it.

-Op surf_name gid cx cy cz nx ny nz scaling––

plane object to specify space varying spacings for the surfaces. These group of options start with the flag '-Op' (plane), '-Ol' (line) , or '-Ov' (vertex) and then t each is followed by a fixed number of number arguments to specify the object and spacing variation.The spacing spec for a givensurface grid point is the product of the regular spacing spec of the underline surface and a scaling factor which is,again, the product of the respective scaling factor of each scaling object group, which, in turn, is the sum of the scaling factors of the nearest two given-grid- point-wise un obscured scaling objects weighted by the distsnce to each of the objects in the same group. Also a line start with # is a comment line.

–– surf_name: a surface id ($\geq 1 or \leq -1$) or label defined. See '-s'.

–– gid: scaling obj group id ($\geq 1, \leq 3$). Weighting of scalings in the same group is additive; while among different groups in multiplicative.

–– cx cy cz nx ny nz: define the plane center (cx,cy,cz) and normal (nx,ny,nz).

–– scaling: scaling factor to the spacing spec. must be > 0.0.

-w num –– scaling obj weighting parameter for varying spacing spec.

num in [0.5,4.0], default= 2.0

num=1 –– linear weighting. Provide linearly varying spacings.

num=2 –– elliptic weighting. Provide smoother varying spacings.

-f fn –– read run options from the file 'fn'. The option's syntax are the same as if they were directly specified in the command line except that they can be in multiple lines.

Example : clu blk.tmp -s 2 1.0e-7 -s 1 1.0e-3 -m 8 -a 2< ret >

Terminologies:

Face –– a face is a block face. The difference with the usual definition here is that when two blocks next to each other, the interface of the two blocks is considered consist of two overlapping faces, one from each block. Therefore a face belongs to one and only one block.

Face group (FG) –– a collection of block faces such that for any two faces, say, F_0 and F_n in this FG, there exists a path of faces F_1..F_(n-1) of the same FG, such that for any 0 =< $i < (n-1)$, F_i and F_(i+1) share an edge and the two blocks that F_i and

F_(i+1) belong to share an interface.A face belongs to one and only one face group.
Extended face group (EFG) −− a union of face groups such that for any two face groups,say, FG_0 and FG_n in this EFG, there exists a path of face groups, FG_1..FG_(n-1) such that for any $0 =< i < (n-1)$, FG_i and FG_(i+1) have overlapping faces. A face belongs to one and only one extended face group.

Notes :

1. The cell growth ratio spec is provisional. The achived growth ratio in the output is some what the product of the growth ratio in the spec andthat in the input grid,plus an adjustment for satisfying other grid quality requirements.

2. A surface spec will be ignored if the off wall cell spacing is less than the spec ('-s ..'), or 'clu' just can't. satisfy some hard requirements derived from the spec. The most likely failure occurs when the off wall spacing is of the same order of magnitude as the spacing spec.

3. The off wal s, pacing for algorithm 0 and 1 is an average sense. The normalized standard deviation of the off wall spacing is unchanged from the input grid to the ouput grid.

4. The off wall spacing for algorithm 2 and 3 is in a per cell sense. The normalized standard deviation of the off wall spacings may be generally reduced from 20-50 percent to 2-5 percent.

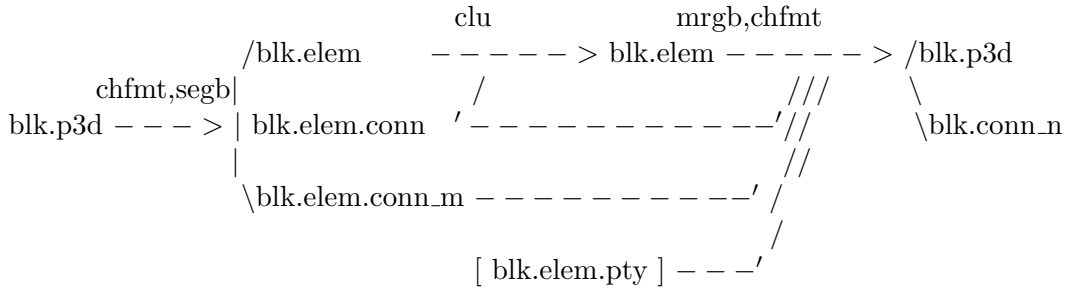
5. For a multigrid grid, the input grid must be a multigrid grid at the desired level. The '-m n' option with a proper n will maintain it.

6. For a typical input grid, the reachable relative cluster level without causing grid folding is beyond $1.0e-13$. The reachable clustering depends also on many other factors,including the quality and the off wall spacing of the input grid.

Cluster General Multiblock Grid:

'clu' used with 'mrgb' and 'segb' can cluster general (incontrast to elementary) multi block grids from other grid generators through a common format such as the PLOT3D data format. 'chfmt' is used to convert data between the PLOT3D and GridPro formats. Then 'segb', 'clu' and 'mrgb' are applied in sequence to generate the clustered grid with the same blocking topology as in the input grid.

The procedure diagram is below:



Notes:

1). 'blk.elem.conn' contains automatically generated surface ids for the facial sheets that have exposed parts. These ids may or may not be the same as what one may think they are.Therefore, to run clu, one needs to know which facial sheet an id is for.One can use the az graphic manager with the MAKE SHEET [surf] button in the viewer panel to visualize this. For a given block topology, this need to be done only once.

2). One may want to write a script to run the procedure.Since one can only set the parameters for clu on a case by case base as noted in 1), a general script is not provided.

3). Properties need to be set by editing 'blk.elem.conn_m' or 'blk.conn_n'. If reproducing the input blocking is not essential, one can use random merging in the dia gram,and the output properties may be inherited from the file 'blk.elem.pty'.Properties in 'blk.elem.pty' can be set using az graphic manager on.blk.elem.

6.6.11 cutb tool

Usage : cutb fn i j k < *ret* >
Purpose : cut a single block grid into 8(=2**3) blocks.
Input : 'fn' −− 3d single block grid in GridPro format.
i j k −− index cut locations (>= 1).
Out range means no cut.
Output : 'cutb.tmp(.conn)' −− 8 block grid in the GridPro format.

6.6.12 mrgb tool

Merge a GridPro grid into general multiblock grid(non-full-face -matching grid, or super blocks). Some CFD solvers prefer a small number of blocks.

Usage : mrgb fn [options] <ret>
Purpose : merge an elementary block grid into a super block grid.
Input : 'fn' −− elementary block data in the GridPro format.
'fn.conn' must exist. 'fn.pty' may exist.
Output : 'fn.tmp' −− merged block grid in the GridPro format.
'fn.tmp.conn_n'
−− nodal connectivity for the merged blocks
'fn.conn_m.tmp'
−− merge schedule for remerging 'fn' the same way.

Options :

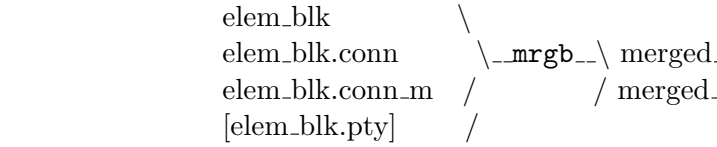
-o	−−	do not output merged grid. This option gives one a chance to try quickly different random merge paths.
-cfx4	−−	output in CFX4 format.
-gasp	−−	output in GASP inp deck + PLOT3D grid.
-gaspXDR	−−	output in GASP inp deck + XDR grid.
-c fn1	−−	merge with a given .conn_m file 'fn1'. This option lets one to do scheduled merge. 'fn1' may be produced from a random merge with 'mrgb'. If further merge on top of the scheduled merge is required, one need to set either the -maxc or -maxb flag.
-maxb num	−−	limit any merged block to be <= num of elementary blocks.
-maxc num	−−	limit any merged block to be <= num of cells. NOTE:num (≥ 0), with 0 means no limit.
-a num	−−	(default=1) 0 or 1, algorithm
0	−−	slowest descending of the number of mergeable face patches.(due to Drs. D. Rigby, E. Steinthorsson, and W. Coirier of NASA Lewis Research Center)
1	−−	fastest descending of the number of face patches.
-i num	−−	(default=1000)>= 0, additional random split and merge iterations.
-s [num]	−−	set the seed for random number generator to select merge path. Without '-s [num]', seed = 1. With '-s 0', seed = current time in seconds. With '-s postive_num',seed = postive_num. The seed with given '-m num', '-a num', '-i num' and without the '-c fn1' option uniquely determines the merge path.
-l	−−	write merge log to 'log.tmp'.
-O	−−	output grid with one overlap layer.

Descriptions :

1). Merge procedures:

Seeded merge:

Scheduled merge:



Note that,

a). The input and output above are not file names.They are used to indicate the nature of the files.

b). [elem_blk.pty] is a optional property file for the elementary block grid data. It may be produced by the property setter of the az graphic manager.

c). merged_blk.conn_n is the node connectivity file for the merged blocks in the GridPro format.

d). elem_blk.conn_m is the merge schedule file in the GridPro format used for future scheduled merges.

2). Format for .pty file: Example,

```
 2 blocks                                                \
   B  1 4 0    1 -1   7 -1   5 -1   0 -1   3 -1   0 -1 | section #1
   B  2 4 0    7 -1   4 0    5 -1   0 -1   3 -1   0 -1/
   1 labels                  \ section #2
   0 symm1                   /
   2D properties             \
   0 DEFAULT                 |
   1 INTERBLK                |
   2 BOUNDARY                |
   3 PERIODIC                | section #3
   4 SYMMETRY               /
   3D properties             \
   0 DEFAULT                 |
   1 BULK                   / section #4
```

Each .pty file has four sections. The first section starts with a line specifying the number of elementary blocks and followed by the block property lines, one for each block.A block property line has the format,

B bid b_pty b_lbid imin_pty imin_lbid imax_pty imax_lbid jmin_pty...

It is a letter 'B' followed by 11 integers. bid is the block id which must be in sequence. b_pty and b_lbid are the block property id and label id. imin_pty and imin_lbid are the facial property id and label id for the i min side face of the current block. The literal meanings of these ids are listed in the sections follow. A value of 0 for label id indicates that it is not assigned. Property ids are >= 0, with 0 means default or not assigned.

The second section lists label id and label string pairs.
The third section lists 2d property id and label string pairs.
The forth section lists 3d property id and label string pairs.
These last 3 sections start with the pair count of the section.

3) . Format for .conn_m and .conn_n files:
They share the same data format. An example of .conn_m file,

```
6super_blocks
SB  1  3  2  2  1  012  4   0
SB  2  3  2  2  11  012  4  -1
...
SB  6  3  2  2  10  012  4   -1
33  face_patches
P  1  1  5  0  0  000  0  0  0  0  1  1  0  0  0  0  0  0  1   -1
P  2  1  10  0 0  000  0  0  0  2  1  0  0  0  0  0  0  0  3   -1
```

...
P 33 6 9 0 0 000 0 0 1 2 1 1 0 0 0 0 0 0 0 -1

It has two sections: one for the merged blocks and one for the facial patches.The first lines of the two sections give the counts of the merged blocks and the patches respectively.They are then each followed by the lines defining the
connectivity and property of each respective merged block or facial patch.
A merged block (superblock) line has the format,

SB sbid I J K ebid eb2sbMap pty lbid

here, SB is a string, stands for super block, other items are integers. sbid is must be in sequence.I J K are the nodal dimensions of the current super block in terms of the elementary blocks. ebid is the base elementary block id, which is siting at (0,0,0) in the IxJxK space. eb2sbMap gives the axis map from the base elementary block's (i j k) directions to the current super block's (i j k) directions.It is in the same format as that in the .conn file.pty and lbid are the property id and label id for the merged block.

If -M flag is on, each SB line are followed by a sequence of format extention lines (with #@, it will be ignored as comment lines by earlier versions of Grid Pro codes) listing all the constitute elementary blocks. The first extention line is the I, J, K dimension of the superblock in terms of elementary blocks. Each line followed lists the block id and its axis map to the superblock for the ele mentary block at the (i,j,k) location. Note that the (i, j, k) index of an extention line is uniquely determinedby its distance from the dimension line with k as the fast running index.

A patch is a rectangular region on a block face.
A patch line has the format,

P pid sb1 sf1 sb2 sf2 fmap ijk1L 1H 2L 2H pty lbid

here, P is a letter, stands for PATCH, other items are integers. pid is the patch id and must be in sequence. A patch has two sides, and each may be attached to 0 or 1 superblock and/or surface. sb1 and sf1 give the superblock id and surface id for one side,and sb2 and sf2 give the corresponding infomation for the other side. A value 0 for sbi or sfi means that no block or surface is attached to that side of the patch. fmap is the axis map from the block on side 1 to the ıblock on side 2. Therefore it is meaningful only when both sides of the patch have blocks attached. Each of ijk1L 1H 2L 2H is a triplet of integers and defines a node in the IxJxK nodal space of a superblock. (ijk1L 1H) are the diagonal corners of a region in the block on side 1 of the patch. Note that the region must be on one of the faces of the block. (2L 2H) are meaningful only when side 2 has a block attached, and give the patch region in the block on side 2.pty and lbid are the property and label ids for the patch.

For the nodal connectivity file (*.conn_n), most of the above applies, except that all the index related numbers are in terms of cells and cell nodes and ebid and eb2sbMap are unused.

4). Mergebility for random merging:(imcomplete) Two blocks are mergeable only if (a).they have the same property type (pid) and label id(lbid), and (b). the single patch that interfaces the two blocks is of pid=0 or 1 (INTERBLK), and (c). the resulting block has consistent pid on all 6 faces.

Two patches are mergeable only if they have the same pid and lbid.

The initial pids and lbids are read in from the .pty file.If the .pty file does not exist, the initial lbids for both blocks and patchs are set to -1;The initial pids for blocks are set to 0;The initial pid for a patch is set to the suface id if it is on a surface that is not labeled '_001_INTERBLK', otherwise it is set to 0.

With the '-P' flag, the resulting block is not required to have consistent pid and lbid on the 6 faces.

Of course, not all mergeables are actually merged.

5). Property inheritance:
For random merge, the property is inherited from the .pty file if it exists.Otherwise, 0 is the value for all properties.

For scheduled merge, the property is inherited from the .conn_m file.Therefore, if the property has been reassigned and the same merge schedule is to be kept(assume can be used), one needs to use the merge seed to rerun the random merge to produce a proper .conn_m file.

6.6.13 mkolp tool

Usage : mkolp fn [options] < *ret* >
Purpose : further process (smooth) overlap layer of grid from 'mrgb -O'.
Input : 'fn' −− block grid in the GridPro format produced by 'mrgb -O'.
'fn.conn' not needed. 'fn.pty' may exist.
Output : 'mkolp.tmp' −− smoothed block grid in the GridPro format.
Options : -i num −− (default=20) set the number of iterations.

6.6.14 mrgg tool

Usage : mrgg fn1 fn2 < *ret* >
Purpose : merge two grid files(MB,hex,quad,tria) into one file
Input : fn1 fn2 – grid files with GridPro(MB,hex,quad,tria)
format.
Output : fn1.tmp (and/or 'fn1.tmp.conn') with the same format.

6.6.15 weld tool

Usage : weld fn1 [fn2] options < *ret* >
Purpose : weld together two grid regions (maybe in one or two files).
Output : 'weld.tmp' (and/or 'weld.tmp.conn').
Input :

fn1 [fn2] −− files in one of the formats:
1). unstructured hexahedral in the GridPro format.
2). multiblock in the GridPro format (need .conn files).

Note: for hex welding, make sure distinct ptys exist for interfacial hexs. (When doing chfmt, use -s sid pty option).
Options :

-p id1 id2	−−	For unstructured hexa: 'id1,2' are the 'pty's on the interfacial elements in the welding region of fn1 (fn2).For multiblock data: 'id1,2' are the 'sid's (> 0 as listed in the .conn file)of which the welding will be done.(default= auto search).If two files are inputed,id1 is for fn1 and id2 is for fn2. The parts of grids with id1 and id2 must have the same topology.
-s h1 h2	−−	manually input a pair of seed hexs (blocks)(>=1). For unstructured data, h1(h2) musthave pty=id1(id2). For mb data, h1(h2) must on surf id1(id2).(default= auto search).
-w depth1 [depth2]	−−	(default=8) depths of the welding regions. May be 0.
-t tol	−−	(default=10) weld tolerance in terms of average relative nodal displacement.
-n	−−	output new cells (blocks) only.
-m	−−	do selection dialog if more than one welding possibility exist. (default=auto selection).
-a num	−−	(used only if no -p flag given). num indicate that the first 'num' pairs of topo matching surfaces of smallest possible nodal displacement will be welded.
-1	−−	(used only if fn2 is given) output fn1 grid with interfacial nodes adjusted to match fn2 grid.[Yes, not really a welding; also by default, fn2 grid is fixed.]
-i [pty ..]	−−	for MB grid only. Reassign the property to PTY_INTERBLK(=1)for the INTER-BLOCK faces that currently have the property ids equal to pty ... With out the [pty ..] args,the reassignment is done for all ptys(that is, strip away ptys from all internal faces).
-I [pty]	−−	for MB only. By default, the welding interface will be reassigned to the PTY_INTERBLK(=1) property. With the '-I' flag the welding interface will keep the pre-welding property.With the '-I pty' flag, the welding interface will be reassigned to property pty.pty must be >= 0.

Notes : 1). format of fn is determined implicitly by the contents of fn.
2). pty is a cell or block face property id.
3). a cell or block can be used for multiple weldings.However, a cell or block can't be on both id1 and id2 for any '-p id1 id2' option.

Examples :

1). weld test1.grid test2.grid< *ret* > −− 'test1.grid' and 'test2.grid' are 2 Grid Pro grids.The associated .conn file must exist. weld will weld the 2 grids into one through a pair of GridPro surfaces (one from each grid as defined in the .conn file)that have a matching topology and have the minimum relative welding displacement.The welded grid is in 'weld.tmp' and 'weld.tmp.conn'.

2). weld test1.grid test2.grid -p 2 3 < *ret* > −−Similar to 1). except that the pair of welding surfaces are specified as surface 2 in ' 'test1.grid.conn' and surface 3 in 'test2.grid.conn'.

3). weld test1.grid test2.grid -p 2 3 -w 10 12< *ret* > −−Similar to 2). except that the number of grid layers used for welding is set to 10 for surface 2 of 'test1.grid.conn' and 12 for surface 3 of 'test2.grid.conn' instead of the default 8. The actual number used is bounded by the number of grid layers in the first block layer. One of the two numbers can be 0 to mean that the welding is done by altering only one side of the grids.

Description :

'weld' can smoothly weld two grids into one by altering the positions of grid points near the welding interface. The need for 'weld' is stemed from the situation where the problem at hands is too large and simple symmetry in the geometry is not available.

In such cases, one can divide the problem into smaller grid generation tasks, then later to join the grids together using 'weld'.

For a better result, it is desirable that the welding intefaces of the two grids are on the same surface geometries and this surface is relatively flat. It is also re quired that in the welding regions,the grid structures are properly layered from the welding surfaces into the volume. The welding interface on each side should be connected. However, the numbering of nodes, cells or blocks can be arbitrary on these layers.These should serve as the rules to guide the user to subdivide the problem.

Based on the layered topology, 'weld' will search for all possible ways for welding.

In general,the more complex the topology is,the fewer possible ways there are. If there are more than one way for welding, normally only one of them is acceptable in terms of grid quality. 'weld' can auto select a best fit. Otherwise,the user is prompted to make a selection. This may be a trial and error process.What a user needs to do is to make a selection,output the grid portion in the welding region, and check the quality. If the quality is bad, then make another selection, and repeat the process.Fortunatelly, for most cases, two or three tries will focus on the correct choice.

If the two grids are unstructured hexs, the weld inteface layers must be already labelled by unique property ids in each grid.When converting a multiblock grid to unstructured hex grid using 'chfmt', property ids can be set for surface blocks.

For multiblock grids, surface ids are used to indicate the weld interface, and the grid in the first layer of blocks from the given surface is used to do the welding.

—

See Also : chkhex, qchk, chfmt

6.6.16 extconn tool

Extend the basic connectivity file to one for a grid that has be extended from a basic grid using symmetry.

Often, due to the symmetries presented in the geometries, grid generation task can be reduced to a small fraction of the oringinal problem. The symmetries can be used to produce a grid for the whole problem. In such a process, the grid data is produced by simple translation, rotation and/or reflection, and concatenation. However, the
connectivity information for the final multiblock grid can not be produced by simple
operations. The purpose of 'extconn' is to produce the connectivity file for the whole grid from the one generated for the partial geometry. 'extconn' handles both periodic symmetry and reflection symmetry with the option of the grid looping back to itself.

Usage : extconn fn copies [sid] [Sid] [-l]] $< ret >$
Purpose : extend .conn (and blk) data to multiple copies of grid.
It can be either periodic or reflective extension.
Output : 'fn.tmp'.
Options :
fn - GridPro .conn file or block data file.
copies -(>= 2, 'x', 'y', 'z', or 'p') the number of copies to extend. If copies=x,y,or z, copies=2is used. It is a refelection about the corresponding axis; Other flags are ignored. The surface id for the symmetry plane is auto searched;if copies = p, a rotational symmetry is assumed. Also, other command line flages are ignored. The .conn must actually have a periodic surface. The rotational axis and the pitch is calculated based on this periodic surfaces. The rotational axis must either x, y, or z. For the case of 'x', 'y', 'z', or 'p' the corresponding grid data is also outputted.
sid - (!=0) the primary surface to which the extension is performed.
Sid - (>= 1) the secondary surface to sid.(for cascading)
-l - looping backed merge
Notes : For periodic BC, the extension is a translation-merge in the index space. 'sid' can be negative and 'Sid' is ignored.For fixed BC, the extension is a reflection- merge in the index space. 'copies' must be even for loop back.If 'Sid' exist, 'copies' is even and there is no loop back,the 'Sid' of the last copy is replaced by 'sid'.

6.6.17 genconn tool

Regenerate the connectivity file for a grid that has no .conn file using a node tolerance spec.
Usage : genconn fn [-t tol] $< ret >$
Purpose : regenerate or check .conn file from point data.
Output :'fn.conn'.
Options :

fn −− az grid data file
-t tol−− (default 0.001) relative tolerance.

6.6.18 smooth_block_edges tool

Usage : "smooth_block_edges [Options]"
Purpose : Convert a hex grid and multi block grid into a multi block grid composed of minimum number of elementary blocks.

Options	Expansion	Description	Default value
-fn	File Name	Input file name with extension '*.fra'.	-
-ifn	Input Grid File Name	Input grid file with an extension '*.tmp' or '*.grd'.	-
-sp	Surface Pairs	The intersecting surface ids in pairs.	All
-ibs	Ignore Built-in Surfaces	A flag to ignore built-in surfaces while evaluating intersecting surface pairs for projection purposes.	False
-outfn	Output File Name	Output grid file with an extension '*.tmp' or '*.grd'.	-

Syntax :
"smooth_block_edges -fn < $Inputfilename$ > -ifn < $inputgridfilename$ > -sp< $surfaceidsinpairs$ > -ibs -outfn < $outputgridfilename$ >"
Purpose :
Project the block edges of the grid to the intersection of surfaces.
Example: smooth_block_edges -fn cylinder.fra -ifn blk.tmp -outfn smooth.grd -sp 1 2 1 3

6.6.19 mildclu tool

Usage : "mildclu [Options]"
Purpose :
Mildclu is an alternative to "clu" to control the off-wall spacing from specified surfaces

Options	Expansion	Description	Default value
-s	Surface Id	Used to specify the surface number and the off-wall The sign of "surfnum" indicates the direction of desired clustering for an internal surface. "spacing" is a positive real number which specifies the desired off wall spacing.	None

-ns	No. of cells of Surface	Specify if the desired number of offwall cells in the block is different from the default.The default is the number in the original grid. "num" (a positive integer) is the required number of offwall cells. NOTE: The no.of cells = no. of points - 1.	No of cells in the original grid.
-fix	Fix Number	To specify the number of off-wall layers that has the same spacings. Similar to the fix parameter in clu.	None
-ng	Node Gap	Do not do post process step to fix possible node gaps at mild_block boundaries.If this option is not specified,chfmt will be used to sync node gaps.	None
-ofn	OutputFile Name	Output file name with extension '*.grd '.	-

Syntax :

"mildclu $< InputGridFileName >$ -s $< surfaceId >$ $< spacing >$ -ns $< surfaceid >$ $< num) >$ -fix $< surfaceid >$ $< num >$ -ng $< nodegap >$ -ofn $< outputfilename >$".

Example : mildclu blk.tmp -s 2 0.001 -s +3 0.01 -ns -s 4 24 -ofn clustered.grd

Notes :

1.The sign of "surfnum" indicates the direction of desired clustering for an internal surface.

6.6.20 chden tool

Usage : "chden [Options]"

Purpose :

Change the density of the grid without running the gridding process again.

Options	Expansion	Description	Default value
-r	Ratio	Ratio to which the density of the grid should be increased.	None
-o	Output Grid File Name	Output file name with extension '*.grd'.	-

Syntax :

"chden $< inputgridfilename >$ -r $< ratio >$ -o $< outputgridfilename >$"

Example :

chden blk.tmp -r 2 -o transform_out.grd

Notes :

1. The tool can be used only on GridPro generated grids.

6.6.21 syncb tool

Usage : syncb fn [options] < *ret* >
Purpose : sync or reorient block handness(+-o) or block indices(-p -s).
Input :
fn −− GridPro grid file name (need .conn or .conn_n file).
For a super block grid, the sid and pty may need to be preserved when running 'mrgb' (-S and -P flags).
Output : 'fn.tmp'
'fn.tmp.conn' Or 'fn.tmp'
'fn.tmp.pty' 'fn.tmp.conn_n'
Options :
Without options, +o is assumed.
+o −− sync blocks to right hand ort.(default)
-o −− sync blocks to left hand ort.
-s index sid1 [sid2..] and
-p index pty1 [pty2..] −−
Defines a constraint. That is, what surface or property must be (or not be) on which index direction (imin, imax,jmin,jmax,kmin, or kmax).

index −− a 6-digit field such as 021100 defines the constraint for the surfaces or properties follow. Leading zeros can be omitted.
Each digit from left to right represents the imin,imax, jmin, jmax, kmin, and kmax index direction respectively.

A value 1(or 0) indicates the associated surfaces or properties can (or can't) be on the pespective index direction A value of 2 indicates the undetermined constraint. After all -s or -p flags, undetermined constraints are reset to 1.
Constraint conflicts can occur and are reported at two levels:
1) Command line argument level: to resolve a conflict, the left flag takes the precedence; 2). Topology level: the conflict is simply reported. The indices of the con cerned block are left unchanged.
The index value 000000 is unphysical; 222222 is no action.

sid −− (!=0) surface id as used in the .conn file. Can take at most 128 sids.
The same sid can appear in more than one -s flags.

pty −− (!=0) pdc property id as used in the .conn file.
Can take at most 128 ptys. The same pty can appear in more than one -p flags.

Note: can only have one type of flags (-p and -s) or +o or -o

6.7 Conversion tools

6.7.1 change_format tool

Usage: "change_format [Options] "

Options	Expansion	Description	Default value
-ifn	File Name	Name of the input file with its extension	-
-outfn	Output File Name	Name of the output file with its extension	-

Syntax:

"change_format -ifn < $inputfilename$ > -outfn < $outputfilename$ > "

Purpose:

Change one file format to another format. The following formats are supported by this command.For more conversion formats, please refer chFmt command.

INPUT FORMAT	OUTPUT FORMAT	SYNTAX
GridPro multi block grid	PLOT3d	gp_utilities change_format -ifn <GridPro grid format> -outfn < $*.plot3d$ >
	CFL3d	gp_utilities change_format -ifn <GridPro grid format> -outfn < $*.cfl3d$ >
	NSU3d	gp_utilities change_format -ifn <GridPro grid format> -outfn < $*.nsu3d$ >
	OpenFOAM	gp_utilities change_format -ifn <GridPro grid format> -outfn < $*.foam$ >
IGES	STEP	gp_utilities change_format -ifn < $*.iges$ > -outfn < $*.step$ >
	STL	gp_utilities change_format -ifn < $*.iges$ > -outfn < $*.stl$ >

Example: To convert to Open Foam

change_format -ifn blk.tmp -outfn grid.foam

Note:

1.Extension should be used for both input & output file to determine the format.

6.7.2 chfmt

Converting 2d and 3d, structured and unstructured data format to a selected unstructured data format. The connectivity information is used. The duplicated nodes on block faces are eliminated. The 'chfmt' utility is a format translator and converter for certian types of grid data. The data formats that 'chfmt' can handle are 1). 2d or GridPro 3d multiblock format; 2).GridPro unstructured hex, quad or tria format; 3). NASTRAN CHEXA, CQUAD, CTRIA + GRID format. and 4). Fidap Neutral File. Notice that, not all format translation or conversion among these formats are meaningful or possible. The input data format is determined by 'chfmt'.

Usage : chFmt fn [-f output_format] [-n] < ret >

Purpose : (under construction) convert and translate data formats.

Output : ‘fn.tmp’.
See Also: iges2gp, and hex2mb.
Options :

fn	−− file in one of the formats:
-n	−− with normal field (default=without).
-c number	−− cutoff below which 0 is taken for NASTRAN and STARCD. (default 1.0e-30)
-g number	−− (default 100) set relative threshold for showing or sync node gap.
-s sid [pid]	−− assign pids to hex layers from surf sid starting with pid sid>=1, pid>=0, default pid = 2, can have 8 -s options.If more than one -s is used, make sure pids are separate enough to avoid accident pid collisions, since hex layers are assigned increasing pids from the surface.(only affect volume grid for now).
-b pid	−− background pid (>= 0, default 0).
-o	−− reverse orientation for all cells.
-f gridpro, nas, nas1, fidap, tria, p3d, pat, stl, ab_switch sync tubex tubey tubez starcd,fluent, mixed, ncc,cgns,dtf	−− output format.
nas	−− NASTRAN, ignoring label id.
nas1	−− for MB→NASTRAN, with pty = pty + 10000*label_id.
sync	−− sync nodal positions for interblock faces.for elementary block data only.
ab_switch	−− switch between ASCII and BINARY formats.
-D	−− do not display the copyright notice.
-d	−− output block I J K to ‘dim.tmp’ for GridPro grid.
-S	−− output surface quad only for MB→NAS_H or PATRAN.
-Ofn	−− prefix for output file name.

Notes: format of fn is determined implicitly by the contents of fn.
Possible conversions are:
FORMAT:

IN	OUT
GridPro 3d multiblock	→ NASTRAN CHEXA + CQUAD4 + GRID.
	→ FIDAP hex and quad neutral file.
	→ FIDAP hex and quad neutral file.
	→ GridPro unstructured hexahedral.
	→ GridPro HEXA + QUAD.
	→ plot3d multiblock.
	→ PATRAN 3d + 2d.
	→ StarCD .vrt .cel .bnd .inp files.
	→ Fluent .msh file.
	→ NCC(Patran)
	→ GridPro 3d MB (bin to ascii, use -f ab)
	→ CGNS (use -f cgns)
	→ DTF (use -f dtf)
	→ ACUSIM (use -f acu)
(use ‘mrgb’)	→ GridPro super block
(use ‘mrgb’)	→ CFX4

(use 'mrgb')	→ GASP
GridPro 3d super block	→ plot3d
GridPro 3d unstru hex	→ NASTRAN CHEXA + GRID
	→ PATRAN
GridPro MIXED (h+q)	→ NASTRAN CHEXA + CQUAD4 + GRID
NAS CHEXA+CQUAD4+GRID	→ GridPro HEXA.
	→ GridPro MIXED.
FIDAP hex and quad	→ NASTRAN.
PATRAN (h+q+t)	→ GridPro MIXED.
	→ GridPro.
	→ GridPro HEXA or QUAD + TRIA.
STARCD .inp file	→ GridPro HEXA
GridPro 2d multi patch	→ NASTRAN CQUAD4+ GRID
	→ PATRAN
	→ GridPro quad w/o norm vector field
	→ GridPro tria (use '-f t')
	NOTE: patch data should have only I J index instead of I J 1 for the first patch.
NASTRAN CQUAD*	→ GridPro quad w/o norm vector field
NASTRAN CTRIA*	→ GridPro tria w/o norm vector field
GridPro quad	→ NASTRAN CQUAD4+ GRID
	→ PATRAN
	→ GridPro quad with norm vector field
	→ GridPro tria (use '-f t')
GridPro tria	→ NASTRAN CTRIA3+ GRID
	→ PATRAN
	→ STL tria
	→ GridPro tria with norm vector field
plot3d ASCII or Binary	
	→ GridPro 3d multiblock (bin or ascii)
GridPro hex (use 'hex2mb')	→ GridPro 3d multiblock
IGES (use 'igeslst')	→ GridPro 2d multi patch
STL tria (use 'mrgn')	→ GridPro tria
GridPro 1d linear	→ tube (along x, y or z axis)
Fluent .msh file (hex)	→ GridPro HEXA
CGNS MB	→ GridPro MB

defualt output format = GridPro hex, quad or tria(or NASTRAN)

6.7.3 surf2tube tool

Usage : surf2tube fn options < *ret* >
Purpose : generate tube surf from a curve, a quad or a tria surf.
Input : fn −− mIX1X1,IX1X1,QUAD or TRIA format.
Output : fn.tmp −− GridPro tube format.
See Also : thin, xsec
Options :

If fn is a curve, must have exact one of below:
num −− radius. fn is center line of the tube
-x −− tube about x-axis. fn is cross sectional curve.
-y −− tube about y-axis. fn is cross sectional curve.
-z −− tube about z-axis. fn is cross sectional curve.
-a ctr dir −−
-a dir −− ctr and dir each is a triplet defining the rotation axis.
If fn is quad or tria, try no flag 1st.
However, you may have one of below:
-x −− tube about x-axis.
-y −− tube about y-axis.
-z −− tube about z-axis.
-a ctr dir −−
-a dir −− ctr and dir each is a triplet defining the rotation axis.
-d −− assume multi-valued. use xsec for calc.

Descriptions:

If the input is a tria or quad surface and no flags are given, 'surf2tube' detects first whether it can be converted to a tube and if yes, 'surf2tube' proceed to convert it.

A convertable surface is one with 1) the data is for the entire 360 degrees of the tube; 2) the center line must be a straight line and 3) From center line radius-ly out, it is single valued (torus is not). The nodes of the data should be largely forming rings about the center line. However, the orientation and the off-origin displacement of the center line can be arbitrary.

There are a few parameters control the conversion procedure, at this moment, none of them are user changeable.

6.7.4 tube2tria tool

Usage : "tube2tria "

Syntax:

"tube2tria $< tubefilename > < triafilename >$ "

Purpose:

Convert a '*.tube ' file to '*.tria ' file. It works only for the tube which is created using 'make tube'command

Example: tube2tria nozzle.tube nozzle_out.tria

Applications:

1.To get a better grid for the tube surfaces.

6.8 Quality check tools

6.8.1 qchk tool

Quality check of grid data: a). the warpage statistics; b). the upper bound fold count, and c). the lower bound fold count. The upper bound fold is determined by the consistency of volume

signs for the tetrahedras generated by cutting a 3d cell with minimum number of resulting tetrahedras. The lower bound fold count is determined by the consistency of the 8 corner volume signs for each cell.

Grid quality is a complicated issue. The complication can range from how a particular quality should be measured to the importance of any given quality. 'qchk' checks only for 4 types of cell based local quality measures:

1).folding: is defined as an inconsistency of Jacobians for a cell. Since it is not practical to sample the Jacobian for every point in a cell, 'qchk' evaluate two simpler folding measures which provide a lower and an upper bound for the folding status of the cell assuming the cell volume is defined by the tri-linear interpolation.

The lower bound: = 0, if the Jacobians based on each of the 8 corners of a cell are consistent.
= 1, otherwise.

The upper bound: = 0, if Jacobians for all minimum trianglelization of the cell are consistent.
= 1, otherwise.

The lower bound may miss true foldings. The upper bound may report false foldings, especially for high aspect ratio cells.

2). skewness: measures the deviation of a cell from the Cartesian one. Again, there is no simple and unique way to measure it. 'qchk' can evaluate four quantities and list their distributions:

a). best, average and worst normalized volume: the normalized volume (NV) is a real value between 0 and 1 defined for a set of three normalized base vectors A, B, C as NV = —A*(BXC)—. For a cell, there are 8 of such NV's, one for each corner. 'qchk' uses best, average and worst normalized volumes to define this skewness measure.

b). worst edge edge angle:

c). worst edge face angle:

d). worst face face angle:

3). warpage: 'qchk' uses the worst warp angle of 12 possible measures for a cell's faces to define the warpage angle of the cell. The warpage angle of a face is defined as the angle of the normal vectors of the two planes generated by cutting through the diagonal corners of the face.

4). smoothness: both turning angle and stretching ratio along grid lines are measured.

In my opinion, a good grid should first have no foldings, then have good warpage measures, then have good skewness measures, and then have good smoothness. The quality measures which is not checked, but may have relevance, include, but not limited to, any of the globle measures one might define and locally defined aspect ratio.

Usage : qchk fn options $< ret >$
Purpose : quality check for grid data.
Input : fn −− multiblock grid data (conn file not needed).
hex or quad data in the GridPro format.
Options : (default: +f, with only lower bound fold check).
-o −− output blocks with bad cells in 'qchk.tmp'.
+-f −− turn off or on lower bound fold check.

+-f u −− turn off or on upper bound fold check.
+-w [num]
−− turn off or on warp check with warp count
threshold = [num] (>10 deg,default=75).
+-s num1 num2 ...
−− turn off or on additional skewness measures:
num < 1 −− set threshold for cell's normalized volume measure.
= 2 −− cell's worst edge-egde angle.
range= 0..180 degrees, good = 90.
3 −− cell's worst edge-face's norm angle.
range= 0..90 degrees, good = 0.
4 −− cell's worst face-face angle.
range= 0..180 degrees, good = 90.
NOTE: measure 1: is always on or off with fold check.
threshold range= [good] 0..1 [bad], default=1.0
measure 2..4: can be turned on or off independently.
+-A [num [num1]]
−− turn off or on cell aspect ratio checks.
threshold = [num1] (> 1.0,default = 100.0).
bin_size = [num1] (>= 0.1,default = 1.0.
+-a −− turn off or on all checks.
+-S [d1 d2] −− turn off or on smoothness checks.
two checks:
1).turning angle of grid lines.
2).stretching ratio of grid lines.
d1 and d2 are thresholds for output bad cells for each check:
d1 −− (default=180 degrees, >10)
d2 −− (default=infinite, >1)
-p −− turn on pictorial view of distributions.
+-surf sid [layers [d1 d2]]
−− turn on checks for off-wall grid smoothness.
For MB grids only. Other checks will be off.
sid −− (>= 1,<= −1) surface to check for.
layers −− (default=4, >= 4) off-wall grid layers to check.

Two checks:
1). 1st order turning angle of off-wall grid lines.
2). stretching ratio of off-wall grid lines.
d1 and d2 are thresholds for output bad cells for each check:
A '-' for d1 and d2 means no change.
d1 −− (default=90 degrees, range= > 0.0)
d2 −− (default=infinite, range= > 1)

6.9 Printing tools

6.9.1 hide tool

Generating hidden line removed data with machine precision in the PostScript format for printing. The default output file is 'eps.tmp'. The input should be a file containing a set of 2d multiblock data in the GridPro data format with the last line giving the parameters of the viewing window:

x_min y_min x_max y_max

The sight is along the z-axis. This combination of data can be generated from **az**. The procedure is: a). use **az** to view a multiblock 2d grid, b). choose a view angle and a view scale, and c). click the [grid]/[save sheet] button.One can also use the 'trf' utility to rotate and translate the data,then, manually append the viewing parameter line.

Usage : hide fn_in [fn_out] options
Purpose: You are running a two(one) stage fast HLR.
Options :

-t [[left_title] [[FILL [center_title]] FILL [right_title]]]
−− place a title in the title bar
a). Without -t flag, print the PDC default title.
b). With -t and without argument, print no title.
c). Otherwise, left(center or right)_title will be printed flush left (center or right). FILL is an as-is key word separates the left_title from the center_title and right_title. Left (center or right)_title is composed of a sequence of space separated (maybe double quoted) strings and the key words TIME and DOT. When printing,TIME is replaced by the stamp of current time,DOT is replaced by a center dot.
NOTE: title must fit in one line
Example:
-t TIME "my_name" FILL "my_tele" FILL "my_email_addr"
will print on the title:
[Mar 09 18.22 1999 my_name my_tele my_email_addr]
-fs −−font size. (say, 35 to 100, default=35)
-c −−print in black and white
-w num −−set grid line width to 'num'.
-W num −−set frame line width to 'num'.
NOTE: for both -w and -W, the default is use values in fn_in.
-x −− reverse the x axis
-y −− reverse the y axis
-z −− reverse the z axis
-f −− with block frames
-F −− block frames alone
-h −− no hidden line removal
-p num −− print on num by num sheets of paper
-T num −− (default=0.000001) node merge tolerance

Contents of fn_in:

First line is for the viewing window: #@ x_min y_min x_max y_max (The sight is along the z-axis). Then, 2d multiblock data in the GridPro format follows.

The I J K line for each block may have 3 or 4 extra real numbers.the first 3 are between 0 and 1 to define the RGB color values for the block.The 4th is the line width. If color or line width are provided this way, I, J and K must be all present even if it has a value of one.

Output : hidden line removed data with machine precision in the PostScript format for printing. The default file name is 'eps.tmp'.

6.10 Other utilities

6.10.1 ascbc tool

Usage : ascbc fn $< ret >$
Options : fn $--$ file name of a multiblock grid data in the GridPro format.
'fn.conn' must exist.
Output : fn.tmp
Purpose : generate surface face list in terms of index range
for multiblock data.

6.10.2 chconn tool

The connectivity format of az3000 grids has changed since its first release. 'chconn'will convert connectivities in the old formats to the newer format.

Usage : chconn fn [format_selector]$< ret >$
Purpose : change conn format from old to new.
Input : fn $--$ conn file in the old format.
Output : fn.tmp $--$ conn file in the new format.
Options :
format_selector $--$ (default=3). 1..3. 3 is newer.

6.10.3 chkhex tool

This tool checks the topological integrity of grid data. It does not check any real space related properties of The grid. The topological properties checked include:

1). cell orintation statistics.
2). node singularity statistics.
3). cell property id statistics.
4). grid disjointness.

Usage : chkhex fn $< ret >$
Purpose : check topological integrity of hex data.

Input :
fn −− file in one of the formats:
1). GridPro unstructured hexahedral.
2). Nastran CHEXA + GRID (not implemented).
Notes: 1). format of fn is determined implicitly by the contents of fn.
2). resource usage: 1M nodes + 1M hex → 100 MB RAM

6.10.4 geth tool

It extracts grids from the GridPro hex grid data. The selection can be based either on cell's property id or on whether it is on a surface. For the later case, the output are quads instead of hexs. The +o and -o are used to synchronize the cell orientations. The +-o, -s and -p options are mutually exclusive, and if more than one option is specified, their precedent order is +-o, -s and -p.

Usage : geth fn [-p pid1 pid2] [-s] [+o] [-o] < *ret* >
Purpose : extract grids with pid in [pid1,pid2], boundary quads.
sync orientation, or synchronize cell's orientation.
Input : fn is in the GridPro hex format
Output : 'fn.tmp'
Options :
-p pid1 pid2 −− specify property id range
-s −− specify surface quads are to be extracted.
+o, -o −− sychronize cell orientations.

6.10.5 getvol tool

Usage : getvol fn < *ret* >
Purpose : calculate volume and surface area of GridPro grid. Cell
folding is also calculated.
Input :
fn −− GridPro grid file. 'fn.conn' or 'conn.tmp' for 'fn' is needed for surface area calculation.
Description:
For a 3-d grid, the total volume is the sum of all cells. To calculate the cell volume, a cell is symmetrically cut into 24 tetrahedrons using the mass centers of the 6 faces of the cell. Any inconsistency of the signs of the 24 tetrahedral volumes is consid ered as a cell folding.To have the surface area calculated, the correct connectivity for the grid data must exist. The surface areas will be listed for each individual surface and for the total. The area of a surface is defined as the sum of the quad areas that compose the surface. The area of a quad is defined as the sum of the 4 trianglar areas resulted from cutting the quad with the lines connecting the mass center of the quad to its nodes. For a 2-d grid, the area is defined in the same manner as the surface area discussed above.
The boundary arc length is not calculated.

For a 1-d grid, the arc length is calculated.

6.10.6 iges2gp tool

This tool converts certain types of IGES entities to digitized IXJ patchs. Data related to the unconverted entities in the input file is skipped. For each converted entity, the parametrization variable U (and V) are assumed to be within the range of 0 and 1.

The first step in the conversion is to generate digitized parameter arrays U[1..I] (and V[1..J]). Then, the digitized IXJ patch is defined by the space positions SURFACE(U[i],V[j]). In turn, U[i] (similarly for V[j]) is generated from i=1 to i = I with U[1] = 0.0. For i ¿= 2, U[i] satisfies the following restrictions:

a). U[i] <= 1.0.
b). min_du < dU[i] (= U[i] - U[i-1]) < max_du
c). within a) and b)., U[i] minimizes
ABS(SURF_NORM(U[i],0.5)*SURF_NORM(U[i-1],0.5) - turn). Here, c). is a curvature condition which is measured along U at V=0.5. I and J depend on user specified parameters min_du, max_du, min_dv, max_dv and turn.

Usage : iges2gp fn options $< ret >$
Purpose : convert iges data to GridPro data with given resolutions.
Output : 'fn.tmp'
Input :
fn −− IGES file with the following entities:
114 −− parametric cubic spline surface.
128 −− rational B spline surface.
126 −− rational B spline curve.

Options :
-u min_du [max_du] −− (default 0.01 0.2). i direction min and max u resolution. Set min_du to '*' to use default.
-v min_dv [max_dv] −− (default 0.01 0.2). j direction min and max v resolution. Set min_dv to '*' to use default.
-t turn −− (default 0.01). max turning angle (¡ 1.0) .
-e num1 num2.. −− num = 114, 126, or 128. not output the IGES entities.
-n −− output control points.
-h −− not to display header of input file

6.10.7 Ggrid tool

Usage : Ggrid fn.fra options
Purpose : generate 3d multiblock grid.
Options :

-e[x [scl]] [fn]
−− generate expanded .fra file (with input corner positions if -R or -r is used).where scl is a scaling factor for theta.

and x can be x,y,z,X,Y,or Z or c
x −− corner (x,y,z) to (x,r,theta) convertion.
y −− corner (x,y,z) to (y,r,theta) convertion.
z −− corner (x,y,z) to (z,r,theta) convertion.
X −− corner (x,r,theta) to (x,y,z) convertion.
Y −− corner (y,r,theta) to (x,y,z) convertion.
Z −− corner (z,r,theta) to (x,y,z) convertion.
c −− strip out -p and -g flags (for compatibility to early version . of GridPro)

-s −− generate user surf lib without run.
-d −− debug the topology.
-r [fn] −− resume a previous computation.
-R [fn] −− setup init grid by interpolating input data.
-S −− show a sample .h surf file.
-c −− display version configuration.
-D −− suspend all screen display.
-D num −− 0-suspend all display; 1-default; 2=suspend copyright.
-i −− dump initial setup only.
-a num −− set mode for exclusion of faces and blocks.
num=0 −− auto off and manual on. (fully manual)
=1 −− auto on and manual on. (semi-auto)
=2 −− auto on and manual off. (fully auto, i.e. ignore 'x f')
default: num=1
-b bid1.. −− used only with -R. Bid1.. in [1,bCnt].
Init block bid1.. with corner positions.
-W −− wait for user return at bad surfaces.
-t t1..t3 -initial translation
-t r1..r9 t1..t3
−− 3X4 initial transformation matrix.
-T −− code debug switch.
-A num −− set tria or quad surf special handling mode.
0 = (default) do nothing.
1 = add a layer of border cells.
2 = output surfaces with added border cells.
-C −− do not do clustering (do Euler grid only).

Notes: 1). Use 'helpaz < *ret* >' to see available GridPro utilities.

2). Ggrid requires a license to run. The license manager daemon 'lcmgr' must be running on a preselected machine. To start the deamon, please follow the instructions in the GridPro/README.install file.

3). To reorder a grid to match 'grid2til' generated TIL, run Ggrid fn.fra -R blk.grd -e < *ret* >Here, 'blk.grd' is the original grid;'fn.fra' is generated by 'blk.grd' through 'grid2til'. 'fn.fra' may be further edited (in 'az') for the surface and property assignments.

The reordered grid that reflects such editing is outputted to 'blk0.tmp' and 'blk0.tmp.conn'. This is particularly useful for adding periodic BCs.

6.10.8 rdmb tool

Usage : rdmb fn [seed] $< ret >$
Options : fn −− file name of a multiblock grid data in the GridPro format.
seed −− positive integer for random seed.
Purpose : randomize bid and axis maps for multiblock data.
Output : 'fn.tmp' and 'fn.tmp.conn'.
Notes : require 'fn.conn' (not 'conn.tmp').

Chapter 7

Property And Boundary Condition Assignments of Grid

Property or boundary condition assignments of grid are simply the assignments to different portions of a grid with special labels which the CFD or FEM solvers understand. Such assignments for a 3D grid can be on a 2D level, such as to indicate a portion of the surface grid being a wall or an inlet, or on a 3D level, such as to indicate a volumetric region being uid or solid. For simplicity, we will uniformly call it property assignments, since the term boundary condition mostly only means 2D assignments.

Before further discussion, it is important to know that, though the property assignments can be done either at the topology building stage or after grid being generated, however, any assignments do not in any way affect the distribution of grid points, that is, the grid generation process.

Further more, with GridPro the property assignments are done only to the elementary block level (3D) and the block face level (2D). If a pointwise distribution of property is required, the user has to take the initiative to develop the required code or rely on the solver to accomplish it.

7.1 GridPro Property Basics

7.1.1 Property id

In general, the property assignments are solver dependent and are a part of the output grid for any specific solver. However, internally the assignments are ONLY represented in the GridPro neutral format, in which a property is just an integer number of greater or equal to zero, which is also called a property id.

When a particular solver format is required the property ids are mapped to the solver properties through a property map file provided either with the GridPro software if the format is currently supported, or by the user if the user coded the conversion for his or her own solver.

The 2D property ids 0 through 3 and the 3D property ids 0 through 1 have generic meanings. They are,

```
2D properties 0 DEFAULT
              1 INTERBLK
              2 BOUNDARY
              3 PERIODIC
```

```
3D properties 0 DEFAULT
              1 BULK
```

Certain GridPro utilities such as mrgb will use these generic meanings to process the grid. Therefore, if the user develops his(her) own solver conversion, make sure the GridPro property ids of generic meanings map to the solver specific properties of corresponding meanings.

Other values of property ids do not have generic meanings in the GridPro neutral format.

7.1.2 Property file

The GridPro property ids assigned to different parts of a grid may be defined in the .pty file explicitly. The .pty file for a grid, say, blk.grd(.conn), must have the file name blk.grd.pty. The following is an example of a .pty file:

```
2 blocks                                          #section #1
B 1 4 0 1 -1 7 -1 5 -1 0 -1 3 -1 0 -1
B 2 4 0 7 -1 4  0 5 -1 0 -1 3 -1 0 -1
1 labels                                          #section #2
0 symm1
5 2D properties                                   #section #3
0 DEFAULT
1 INTERBLK
2 BOUNDARY
3 PERIODIC
4 SYMMETRY
2 3D properties                                   #section #4
0 DEFAULT
1 BULK
```

A .pty file has four sections. The first section starts with a line specifying the number of elementary blocks in the grid and followed by the block property lines, one for each block. A block property line has the format,

B bid b_pty b_lbid imin_ pty imin_lbid imax_pty imax_lbid jmin_pty...

It is a letter 'B' followed by 11 integers. bid is the block id which must be in sequence. *b_pty* and *b_lbid* are the block property id and label id. *imin_pty* and *imin_lbid* are the facial property id and label id for the face on the i min side of the current block. The literal meanings of these ids are listed in the sections follow. A value of -1 for the label id means default or not assigned.

The second section lists label id and label string pairs. The third section lists 2d property id and label string pairs. The forth section lists 3d property id and label string pairs. These last 3 sections start with the pair count of the section. These literal strings are just for user reference purposes.

The label ids can be used to further distinguish subregions of the same property, such as inlet 1, 2, 3...

Property deduction rules

Internally in GridPro, the property assignments for a grid always exist even if the assignments has not been done explicitly and/or the .pty file does not exist.

Within the GridPro system, including Ggrid and various utilities, the property may be inherited, transferred, regenerated, and mapped. The deduction of the property for an outputted grid follows a set of predefined rules. It is important to understand fully these rules in order to assign properties correctly. It is also important to have any user developed utilities that involve GridPro properties follow this set of rules to insure the uniform handling of the properties as the other GridPro utilities.

The deduction of properties is based on 1) the property assignments of GridPro surfaces defined in the topology and appear in the .conn file; 2) the property assignments defined in the .pty file; and 3) a set of default property rules that convert the property 0 (DEFAULT) to a non-default property.

File Precedence Rules:

Rule 1: If the .pty file exists, the properties are as defined in the .pty file. This means that the properties defined in the .pty file always take the highest precedence. In this case, a change to the property assignments of GridPro surfaces, or equivalently, a change of the surface properties defined in the .conn file will not take effect unless the .pty file is deleted or regenerated.

Rule 2: If the .pty file DOES NOT exist, the properties assignments are deduced from special surface labels in the .conn file. The special surface label has the form, say, "*4_005_ USER7*". The number, 5, is the property id assigned to the surface number 4. The "*USER7*" here is only a literal reference string that does not have effects on the assignment. The assignments proceed as follows:

All block faces that is on a surface having a property id assigned with special surface label will be assigned to the same property id. All other block faces are assigned the property 0 *(DEFAULT)*.

Rule 3: After Rule 1 or Rule 3, any assignments of property 0 (DEFAULT) is further process by the Default Property Rules.

Default Property Rules: converting property 0 to non-0 properties based on the connectivity information in the .conn file.
3D Rules:
Rule 1: convert to property 1 *(BULK)*.
2D Rules:
Rule 1: if the face is a inter-block face, convert to property 1 *(INTERBLK)*.

Rule 2: if the face is a periodic face, convert to property 3 *(PERIODIC)*.

Rule 3: if the face is a 1-sided block face, convert to property 2 *(BOUNDARY)*.

Note again,the default property rules gives certain property ids special roles.

7.2 Incorporating Your Solver Format Into AZ-Graphic Manager

To understand this section, you need to reading about AZ-Graphic manager first (see the GridPro GUI manual).

AZ-Graphic Manager has a command panel for assigning properties to blocks and block faces. The grid data and the associated information can be outputted in any solver format that

has already been supported in AZ-Graphic Manager.

AZ-Graphic Manager implements the functionalities for outputting grid in any given solver format as a plugin script and executable. That is, a user can add to the AZ-Graphic Manager the capability of outputting the grid in a given solver format without the requirement of knowing, changing or recompiling the source code of AZ-Graphic Manager.

1) Adding entries into two existing files:
   −−GridPro/az_mngr/gridfmt.menu
   −−GridPro/az_mngr/ptymap.menu

2) Creating a property map file that maps GridPro property ids to solver specific literal names. These mapped property names are loaded into AZ-Graphic Manager when the given solver format is selected from the menu button [**pty=***] on the top menu bar.
3) Creating an executable that takes in the GridPro grid (including grid data, connectivity and property assignments) and the corresponding property map, and output the grid in the given solver format. The execution of this executable is a part of tasks in the script mentioned in 4).
4) Creating one or two UNIX or PC (.bat) script files that are called when the grid is saved in the given solver format within the AZ Graphic Manager. The executable mentioned in 3) and other GridPro utilities such as trf, mrgb and chfmt can be used in the scripts.

The grid files in the GridPro format are solver independent in the sense that no solver specific names appear in them. The linkage between the grid in the GridPro format and a particular solver is through the property map file where the numbers (property ids) are mapped to the solver specific literal names.

For the purpose of explaination, we will call the new solver *MYSLV*.

7.2.1 Add An Entry To GridPro/az_mngr/gridfmt.menu

The file **GridPro/az_mngr/gridfmt.menu** controls the solver list in the format line of the file dialog box when a grid is to be saved. Each solver format takes one line in the file. The line for a new solver can be inserted in any position in the file. Each solver format line has four items separated by the symbol '&'. For example,

```
MYSLV &MYSLV & outE myslv.script & outM myslv.script
```

The first item *MYSLV* is the solver name that appears in the format list when a grid is saved. It should not be longer than 10 characters. The second item is not used for now. The third item is the script or executable file to run when the grid is saved in the form of elementary block data. The forth item is the script or executable file to run when the grid is saved in the form of super block data. For the PCs, the file names in item 3 and 4 must be script files and should end with the '.bat' extension. One of the third and the forth item can be the reserved name 'unused'.

Internally, when a grid is saved, say, with the elementary block data, AZ-Graphic Manager always first saves the corresponding .pty file, then executes the following system call,

```
system("outE myslv.script grid file name output file name");
```

where, *outE_myslv.script*– name of the file to run as specified in the solver line above. *grid_file_name* – the file name of the current grid. *output_file_ name* – the output file name typed in during the file dialog. The current 'gridfmt.menu' file is listed below,

```
#---- FILE: gridfmt.menu ----
#menu lbl & head & elem b script to run & super b script to run
CFX4 &CFX4 & outE cfx4.script & outM cfx4.script
GASP &GASP & outE gasp.script & outM gasp.script
plot3d &plot3d & outE p3d.script & outM p3d.script
FIDAP &FIDAP & outU fidap.script & unused
Nastran&Nastran& outU nast.script & unused
Patran &Patran & outU patran.script& unused
Fluent &Fluent & unused & unused
pdc &pdc & outE pdc.script & outM pdc.script
pdc uns&pdc uns& outU pdc.script & unused
#---- END OF FILE ----
```

7.2.2 Writing The Output Script

As mentioned in the previous subsection with the example for *MYSLV*, when a grid is saved, the AZ-Graphic Manager makes a system call to execute the script outE_ myslv.script (or outM_myslv.script). It is the user's responsibility to write the script, though one may copy an existing script of another solver in the directory **GridPro/az_mngr/** and edit it from there. This script should be placed under **GridPro/az_mngr/**. It should read in the GridPro elementary block grid (grid data, .conn file and .pty file), and the property map file ptymap.myslv which will be discussed later. And, it should output the grid in the MYSLV format. One can use a combination of GridPro utilities including trf, mrgb and chfmt, in the script to accomplish some of the tasks involved.

However, in general, one needs to write at least one C or FORTRAN program to finish the final convertion. Typically, this program will read GridPro files, say: blk.grd, blk.grd.conn and optionaly blk.grd.pty for elementary blocks and blk.grd blk.conn n for super blocks.

An example script for GASP is listed below,

```
#!/bin/csh -fe
#---- FILE: outM gasp.script ----
mrgb $1 -s 1 -gasp
mv $1.tmp.inp $2.inp
chfmt $1.tmp -f p3d
mv $1.tmp.tmp $2.p3d
exit
#---- END OF FILE ----
```

```
The corresponding PC .bat file is
rem ---- FILE: outM gasp.script.bat ----
mrgb %1% -s 1 -gasp
mv %1%.tmp.inp %2%.inp
chfmt %1%.tmp -f p3d
mv %1%.tmp.tmp %2%.p3d
del %1%.tmp
rem ---- END OF FILE ----
```

7.2.3 Add A Entry To GridPro/az_mngr/ptymap.menu

The property maps map the internally used GridPro property ids to solver specific literal names. Each solver has its own property map file which appears also in the corresponding solver entry in the file **GridPro/az_mngr/ptymap.menu**.

In **GridPro/az_ mngr/ptymap.menu**, each solver takes one line. And each solver line has three items separated by the symbol &. Again, using **MYSLV** as an example, we need to add one entry as follows,

```
MYSLV &pty=MYSLV & ptymap.myslv
```

The first item is the new solver name appearing in the menu item list for both the button [**pty=***]on the top menu bar and the property line in the surface parameter dialog box. The second item is the name label appearing on the two menu buttons once the property is selected from the menu item list. The first two items should not be longer than 10 characters each.

The third item is a file name containing the property maps for the given solver which will be discussed in the next section. This file is solver speci

c and user supplied.

The current ptymap.menu file is listed below,

```
#---- FILE: ptymap.menu ----
default&pty=PDC & ptymap.default
CFX4 &pty=CFX4 & ptymap.cfx4
GASP &pty=GASP & ptymap.gasp
FIDAP &pty=FIDAP& ptymap.fidap
#---- END OF FILE ----
```

7.2.4 Writing The ptymap.* File

Again, it is the user's responsibility to create the file *ptymap.myslv*. The literal property names in the file will be what one sees when using AZ-Graphic Manager to assign properties to grids with the involved solver format.

It is the file *ptymap.myslv* that makes the connection from the GridPro property id numbers to the literal names that are specific to the solver.

To create ptymap.myslv, one can either copy and edit the file ptymap.template or the ptymap.* file for another solver.

Let us use *CFX4* as an example. The file is listed below,

```
#---- FILE: ptymap.cfx4 ---
10 # 2d-ptys.
#pdc-id& pdc-name & mapped name &label #comments
1 & INTERBLK & BLKBDY &*BLKBDY #grid generic basic
2 & BOUNDARY & WALL &*WALL #grid generic basic
3 & PERIODIC & USER2D &*USR2D:prd#grid generic basic
4 & SYMMETRY & SYMMET
5 & user5 & unused
6 & user6 & CNDBDY
7 & user7 & PRESS
8 & user8 & INLET
9 & user9 & OUTLET
```

```
10 & user10
#---------------
7 # 3d-ptys.
#pdc-id& pdc-name &mapped-name &label #comments
1 & BULK &default &*FLUID #grid generic
2 & user2 &SOLID &
3 & user3 &SOLCON &
4 & user4 &POROUS &
5 & user5 &USER3D &
6 & user6
7 & user7
#---- END OF FILE ----
```

There are two sections of maps, one for each of the 2-d maps and the 3-d maps. They have the same syntax: The first line is the number of properties listed. Then followed by the property map list. Each line here defines the map for one property.

There are four items for each map line. They are,

1) pdc-id: must be >= 1 and <= the length of the list, and can appear in any order.

2) pdc-name: generic property names. can be renamed. However, (2d)id=1..3 (3d)id=1 have special roles when the property of surfaces, faces and blocks are initialized. The default property rules previously discussed apply here.

3) mapped-names: solver specific names. Routines for a particular solver format may use these names. A blank *mapped_name* means unused. Due to the default property rules, the mapped-names for (2d)id=1..3 should have the same corresponding meaning as the pdc-names.

4) label: (<= 10 chars) is what is shown on the button, can have spaces.

5) & : delimiter. A blank item should use & & instead of & &

For a PDC pre-defined pty map file, a user

1) can change the order of the lines;

2) can change the labels;

3) can change pdc-name for id >= 4(2d) and id >= 2(3d);

4) need to understand the initialization rules.

For a user defined pty map file, a user needs to map *INTERBLK* correctly since mrgb will merge blocks through it.

Chapter 8

Graphic Manager

This is a brief discussion of az Graphic Manager. Details are left to The GridPro GUI manual.

The az-Graphic Manager (invoked with az) is an integrated GUI serving several purposes: 1) To build, inspect and debug topology for GridPro , 2) To perform some CAD functions for surface building and restructuring, such as, segment surfaces specified with unstructured triangles and quads data, 3) To view multi-block grids; And 4) To assign grids properties, such as boundary conditions for interfacing with CFD or FEM solvers. That is, az is a topology builder (grid preprocessor), a mini CAD (grid preprocessor), a multi-block grid viewer (grid postprocessor), and a property setter (grid post processor) for GridPro.

The objects displayed and manipulated with az may be 1) Topologies that are interactively created or from existing TIL codes; 2) Surfaces stored in data files; And 3) Multi-block grids.

As a topology builder, az outputs TIL codes which are used as both a means to record the designed topology for later redisplay, and the input for the main grid generation process (Ggrid). Certain handy features such as unlimited undo-redo for corner and link creation and movement, graphic handles, non-linear mouse sensitivity, dynamic drawing interruption and logical selection and grouping operations are a basic part of az functionalities. One can also launch Ggrid within the az environment.

As a limited version of digital CAD system, current implementation allows one to segment -tria and -quad surfaces to suit the needs of one's topology design.

The grid viewer part of az is specifically tailored to view the structured multi-block grids. It does not in any way change the grid. It has flexible grid sheet visual making and trimming capabilities. It is also coupled with the stand-alone, advanced hidden surface removal utility of GridPro to generate high resolution prints in the Postscript format.

In the future implementation, Ggrid scheduling and az utility operations will be part of az functionalities.

8.1 Hardware requirements

The hardware requirements and resource usages to run az are:

1) IBM/rs6000 under AIX3.2.5 or later, SGI under IRIX 5.3, or DEC/alpha under OSF1 V3.2 or later, HP 700/800 series and Windows 95 or Windows NT.

2) 24 bit Z-buffer and 8 bit double buffered color planes.

3) > 16 MB RAM + 2x25 MB swap space.

4) 20 MByte hard disk (60MB for PC).

8.2 Graphic Layout

Az is invoked by typing:

```
~~~~~~~~~az [Options] <ret>
```

Type

```
~~~~~~~~~az -h <ret>
```

for options.

Certain color choices can be changed system-wide through modifying the file `colormap` in the directory `$GRIDPRO/az_mngr/`.

An az application window consists of three primary sub-windows. The main viewing area occupies most of the application window. On the top is a menu bar. And, on the right is a command panel with buttons grouped by their functionalities.

The command panel can be switched between a topology builder, a surface repair panel, a grid viewer panel and a property setter panel. They are shown in Figure 8.1 for the topology builder window, Figure 8.2 for the surface repair panel, Figure 8.3 for the grid viewer window, and Figure 8.4 for the property setter window. The windows differ only in the lower part of the command panel.

8.2.1 The viewing area

The viewing area accepts viewing operations and topology/grid operations through mouse button press/release and mouse movements. A particular combination of key pressings on the keyboard, button settings in the command panel and/or current states of graphic handle sets the operation mode.

A graphic handle is a graphic object on the screen that a mouse button pressing on it changes the operation mode.

In general, the left mouse button is used for translating an object or the view, and for picking and moving an object; The middle mouse button is used for rotating a object or the view; The right mouse button is used for zooming the view and doing regional selections of objects,

The mouse cursor changes when it is on different objects. It may indicate the current mouse operation mode.

A single beep indicates an invalid operation; While multiple beeps warn that a pick operation picked more than one object.

8.2.2 The menu bar

Each word in the menu bar is a button. A mouse button press on a word in the menu bar displays a pull-down menu. A selection is made by releasing the mouse button on the desired item in the pull-down menu.

[exit] menu:

for quitting az and printing the displayed grid. If [quit] is selected, a confirmation is prompted for the final exit.

[surf] menu:

for loading, reloading, and deleting of surfaces.

[topo] menu:

for reading, saving or debugging topologies and for the creation and deletion of topological components. When [TIL save to _az.fra] or [Ggrid start] button is selected, the current topology is written to a file named ‘ _az.fra’ . Older versions of ‘_az.fra’ will be renamed to ‘_az.fra.~?~’ , where ? can be a number from 1 to 4.

[grid] menu:

for loading, reloading, saving, and deleting of multi-block grids.

[Dim=3] menu:

for indicating whether the current case is 2 dimensional or 3 dimensional. The current selection is shown on the button. When the topology is saved, the DIMENSION parameter of the TIL code uses the current setting of this button. Also, when [Dim=2] is on, rotation operations of the viewing objects can only be about the z axis of the world coordinate system.

[Read=N] menu:

for setting the viewing mode when a topology is read in. The current selection is shown on the button. When [Read=D] is on, the topology input will be shown step by step. When [Read=D] is on, the current grid (if it exists) is in an auto reload mode.

[Panel=T] menu:

for setting the lower half of the command panel. The selection [Panel=T] is for topology building. The selection [Panel=S] is for repairing surfaces of type -tria and -quad. [Panel=G] is for grid viewing. The selection [Panel=P] is for setting the grid properties. By default, az has [Panel=T] as the current panel. [Panel=G] can also be invoked with az command line option ‘-v’. The current selection is shown on the button.

[pty=PDC] menu:

Select a solver format. Only affects the post processing of the generated grid.

[help] button:

On-line dynamic help.

8.2.3 The upper half of the command panel

The buttons in the command panel are grouped by the functionalities. From the top to the bottom, the groups are:

‘ROTATE’ subwindow:

Used for setting the mode of rotation operations for the view. This concerns the selection of a rotation center and a rotation coordinate system. Note that the rotation operations themselves are performed through mouse operations inside the viewing window. Under the [world] system, all the rotations are about the current axis. The current axis is the thinner axis in the display. A click on an axis sets it to the current axis.

[snap] – for snapping the view into engineering views.
under sys – for selecting rotation system.
[pk] – for picking the rotation center from a screen object.
under ctr – for selecting rotation center.

'STYLE' subwindow:

Used for selecting the drawing style of surfaces and grid sheets. The moving style and the stopping style can be selected independently to best match the hardware capacity. The [shade0] selection is for the shading style for the portion of surfaces near the cut plane. This provides both a good speed and a localized view of surfaces. The [HLR] is a crude, but fast, implementation of the hidden line removal algorithm.

Tapping the left mouse button forces a redraw with the moving style; while tapping the right mouse button redraws the objects with the stopping style.

'SHOW' subwindow:

The buttons here control what are displayed on the screen on an overall level. Some detailed controls can be done with buttons in other subwindows.

[axis] – coordinate axes.
[surf] – all surfaces.
[cut] – cut plane.
[top] – topology: corners,links and other objects.
[vec] – topology: (reserved for future use).
[x&a] – topology: added and excluded objects.
[e&h] – topology: error and highlighted objects.
[pos] – coordinate boxes of objects.
[MB] – grid viewer: all grid objects.
[blk] – grid viewer: block frames.
[she] – grid viewer: existing grid sheets.
[ort] – grid viewer: block orientation indicators.
[bid] – grid viewer: block ids.

'CUT-P' subwindow:

The Cut Plane is a plane of focus. It has many functions. It is used for cutting the view, placing the corners, manipulating a sub-topology and so on. The buttons in this subwindow control how the cut plane should look and function.

[pos] – position the cut plane with numbers or by picking a corner or link point
(A mouse pick may be needed.)
[ctr] – center the cut plane at an object with a proper visual scale. Menu center selections:
`[ view ]` – at the viewing window center.
`[ group ]` – at the current topology group center.
`[grp fit 1]` – the above plus the normal axis aligned with the group major axis 1.
`[grp fit 2]` – the above plus the normal axis aligned with the group major axis 2.

`[grp fit 3]` – the above plus the normal axis aligned with the group major axis 3.

[norm] – normalizing the normal axis of the cut plane.

[hand] – show the graphic handles of the cut plane.
(So the cut plane can be handled with viewing operations)

[clip] – clip the surfaces/grids with the cut plane.

[side] – set the side to be clipped (need [clip] on).

[fill] – fill the cut plane with translucency.

'UNDO-REDO' subwindow:

For the undo and redo of corner and link creation/deletion, and the view changes.

[unzoom] – unzoom or zoom out the view.

topo[<][>] – undo and redo corner and link operations.

[view] – determine whether the topology undo-redos should be with corresponding view changes.

[grp] – determine whether the topology undo-redos should be in group or by step.

view[<][>] – undo and redo view points.

[rec] – record the current view for later undo or redo.

8.2.4 The lower half of the command panel: Topology builder

'TOPO' subwindow:

For building topology, in particular for setting group action modes.

[S.][+][-][x][p-bc]– for selecting current surface, and surface assignments for corners and for setting periodic boundary conditions.

[G][1]...[9] – 9 topology groups are available. One can be active.
[G] changes the display style of the active group.

[<][>] – access backup topology groups. 22 groups may be backed up.

[+][-][*][x] – for constructing and modifying the active topology group.

[cp0][1][2] – copy the active group toward the cut plane with [cp0]= simple translation, [1]= [cp0] + drop back links, and [2]= [1] + projection to the cut plane.

[i:a] – for setting corner insertion mode. [i:a] is for all,
[i:g] is for group, and [i:1] is for one link.

[den] – for setting edge grid density.

[mv] – attach the active group to the cut plane for transformation.

[wrp2][1] – for constructing a wrap of the active group. The active group must be a 2-d quad object (unstructured) for [wrp2], or 1-d string like object for [1].

[P] – pancake the group to the cut plane.

'CURRENT' subwindow:

For selecting the current surface and component. Note that the current surface can also be picked with [S.] button. Individual surface and component can be switched on and off for

display.

[surf][<][>] – change the current surface and switch on/off the display of it.
[comp][<][>]– change the current component and switch on/off the display of it. (This part is not fully implemented) .

'Pick mode by key' subwindow:

A list of keys for modes is given. When a listed key is pressed, the mouse operations in the viewing area are in a special mode. In the following table a click means a left mouse button click.

c – A click places a corner on the cut plane.
e – Clicking two corners sets a link(edge) between the 2 corners.
i – A click on a link inserts a single/group corners with proper links.
s – A click on a surface makes it current. A click on a corner toggles the assignment of the corner to the current surface.
r – A click on a corner/edge or added/excluded object remove that object.
a – A click on an edge assigns the current surface to the edge. Clicking diagonal corners of a face assigns the current surface to it.
x – A click on an edge unassigns the current surface from the edge. Clicking diagonal corners of a face unassigns the current surface from it.
f – Clicking diagonal corners of a face excludes it to be a topological object.
b – Clicking diagonal corners of a block excludes it to be a topological object.
p – A click on a corner places a projected corner on the cut plane with a drop back edge.
q – Query current position of the cursor on the cut plane, or the corner.

8.2.5 The lower half of the command panel: surface repair tools

[path+][-][<][x]: segmentation path generation buttons. When [path+] is on, surface nodes can be picked. Each pick adds to the segmentation path a segment of new path that connects the current picked node to the previous picked node. The path segment is determined by an algorithm that weighs both the feature strength of the path and the path direction. With [-] on, an arbitrary segment of the path can be removed by picking surface nodes at the two ends of the segment. If more than one path exists between the two nodes, the shorter path (in terms of node-bond count) without branches is removed. [¡] retracks the last path segment and [x] clears the entire path.

[Segment]: surface segmentation button. This button segments the surface with the path generated using the button discussed above. A segmentation is done only when the path can partition the surface into two or more disjoint pieces and no redundant path segment exists.

[cur piece][<][>]: surface piece selection buttons. After a segmentation, a surface is segmented into multiple pieces. One of the pieces is the current piece indicated by the sea blue color. [cur piece] button display and undisplay the current piece. [<][>] scrolls the current piece among all pieces.

[save]: surface piece save-to-file button. It saves the current piece to the file '_surf.?' , where ? is the piece id number.

[feature+][x]: display and undisplay the features. A surface feature is a cell edge for which the angle between the cells sharing the edge larger than a given number. Each press on [feature+]

will add highlight on the next most significant features in sequence. [x] will clean all feature highlights.

8.2.6 The lower half of the command panel: Grid viewer

'CUR' subwindow:

for selecting the current grid set and the current grid sheet,stepping and scrolling the current grid sheet and deleting the current grid sheet.

'TRIM' subwindow:

for trimming blocks in the current grid set (B) and faces in the current grid sheet (SF).

'MAKE SHEET' subwindow:

for making grid sheets for display. There are three methods: 1). surface sheet; 2). shell sheet; And 3). sheet that is perpendicular to a block edge.

Bottom subwindow:

[C:she] button: for selecting the grid colouring method. The choices are 1). color by sheet; 2). color by block; And 3). color by IJK index direction.

[space]: for viewing the grid spacing by clicking on a node bond (cell edge).

8.2.7 The lower half of the command panel: Property Setter

There are two subwindows, one for setting 3d properties and the other is for 2d properties. Since they function very similarly, the 2d subwindow will be explained in detail.

GRP:[F.][+][-][*][all][ ] : these are operators to construct the viewing portion of the surface and interblock patches.

[wall][<][>][list]: these are buttons to select the current property. Patches with the current property are shown in the sea blue color.

[.][+][-] : [.][+][-] are property assignment buttons. [.] toggles a face between the current property and the previous property of the face. If the previous property is not defined, the toggle is to the 'undef 2d' property. [+] assigns the faces in a regional selection with the current property. [-] assigns the faces of current property in a regional selection to the previous property of each individual face involved.

8.3 Building topology with az

8.3.1 Inputting surface

Click on the [surface] button in the menu bar, then make correct selections thereafter. If the surface is specified by a data file, the data must be in the GridPro format and of GridPro types. The format and type assignments are done automatically.

The acceptable types are: 1-d or 2-d single or multiple structured patches, tube surfaces, triangular or quad unstructured surfaces. Note that the utility `chfmt` and `iges2pln` can convert surface data in many other formats to the GridPro format.

For the implicit plane type, a surface has an infinity span in space. However, on display is only a visual representation of the plane (that is, a part of the plane is displayed for visual clarity). This visual representation can be manipulated in form by setting visual handles on it and operating on these visual handles. The visual handles are set on by clicking the cut plane off, clicking the [hand] button of the cut plane on and selecting the to-be-manipulated plane as the current surface.

Surfaces can also be inputted with topology input.

8.3.2 Inputting topology

Click on the [topology] button in the menu bar and select [TIL read], then make correct selections thereafter.

The menu bar selection of read mode controls how the display is done. For [norm-read], the topology is displayed at the end of the input; For [demo-read], the topology is displayed for every TIL statement read in. This gives some demo effects.

8.3.3 Changing the viewpoint

Translation:

Move the mouse with the left mouse button pressed.

Rotation:

Move the mouse with the middle mouse button pressed. The <ROTATE> subwindow sets the rotation modes. Under the world system, clicking on an axis in the viewing window makes that axis the rotation axis.

Zoom and unzoom:

To zoom, forming a zooming box in the viewing window by pressing the right mouse button on one corner of the intended box, then moving the mouse with the button pressed to the diagonal corner of the box and release the button.

To unzoom, click on the [unzoom] button in the 'UNDO-REDO' subwindow.

Undo-redoing views:

The viewpoint can be undone or redone by the view [<][>] buttons. The views in record are those when a corner is created or moved (automatically recorded), or those when the [rec] button is pressed (manually recorded).

8.3.4 Changing the viewing method

Surface style:

Surfaces can be displayed in different styles such as shading, line and point. The moving and stopping styles of surfaces can be set independently.

All surfaces can be undisplayed by the [surf] button in the 'SHOW' subwindow. The current surface can be undisplayed by the [surf] button in the 'CURRENT' subwindow. The cut plane related modes also affect what part of a surface is seen.

Note that visually the current surface is uniquely assigned the sea blue color. An undisplayed surface will be displayed when it is scan-passed by the [<] [>] buttons in the 'CURRENT' subwindow.

Topology viewing:

One can select to show the whole topology, a component of a topology or a subgroup of a topology. The topology can be undisplayed by the [top] [x&a] and [e&h] buttons in the 'SHOW' subwindow. Group display mode can be selected by the [G>] button in the 'TOPOLOGY' subwindow. Note that, before displaying a topology group, it must be formed. The [+][-][*][x] buttons next to the [G>] button combining with region selection operations are used to form a group.

Setting and using the cut plane:

When the visual handles are on (press the [hand] button), the cut plane can be resized, rotated, and translated. With the left mouse button, clicking on the handle axis selects the rotation axis; Pressing and dragging on any of the cut plane corner handles resizes the cut plane's visual representation; And pressing and dragging on the normal handle axis translates the cut plane along the normal axis, and pressing and dragging on the other two handle axis translates the cut plane in the cut plane. The middle mouse button, when pressed and dragged on any handle axis, performs cut plane rotation.

The cut plane can be centered to the view center by clicking on the [ctr] button in the 'CUT PLANE' subwindow. It can also be numerically set by clicking on the [mv] button.

Many other operations rely on the location and modes of the cut plane.

Changing the current object:

To change the current surface, one can:
1) Use the surf[<][>] buttons in the 'CURRENT' subwindow;
2) Hold down the s key and click on a displayed surface;
3) Set the [S.] button on, then click on a displayed surface;
4) Input a surface.

8.3.5 Placing corners and links

Placing a single corner or link:

For simple corner creation, corners are always created on the the cut plane; Therefore, one first needs to properly place the cut plane before creating any corners.

To create a corner on the cut plane, hold down the c key, then click the left mouse button in the viewing window. The corner is shown by a small orange square.

To create a link (or edge), hold down the e key, then click, in sequence, the left mouse button on the two existing corners to be linked. A link is shown by a yellow line connecting the two corners.

One can undo or redo a corner or a link by topological undo-redo buttons. Alternatively, one can remove a corner or a link by holding down the r key while clicking on the corner or the link to be removed. Note that undo or redo follows the action sequence, while removing actions can be random. However, removing actions and undo actions themselves are undo-able actions.

Moving a single corner:

There are two moving operations: 1) Parallel to the cut plane, or 2) Normal to the cut plane.

To move a corner parallel to the cut plane, press the left mouse button on a corner, then drag the corner with the button still pressed, and release the button when the corner is at the desired location. Similarly, corner can be moved normal to the cut plane by using the right mouse button.

Corner moves are undo-able actions.

Projecting a corner:

Holding down the p key while clicking on an existing corner will create a new corner on the cut plane with a drop back link to the clicked corner. The location of the new corner is the projection point of the clicked corner on the cut plane.

Inserting corners

Holding down the i key while clicking on an existing link inserts a corner to every link in the parallel group of the clicked link when [i:grp] is on. Otherwise, if [i:one] is on, only one insert corner is generated and is only for the clicked link.

Forming a corner group:

The corner group is, if non-empty, displayed by clicking on the [G>] button. The [x] button is used to empty the group. The [+][-][*] buttons set the group construction modes. If one of the buttons is set, the right mouse button, which is normally used to set the zoom box, is now used to set a region box. For [+], the corners in the region box is added to the group; For [-], the corners in the region box are subtracted from the group; and for [*], the intersection of the corners in the region box and the corners in the group are left in the group.

Corner group actions:

Corner groups can be transformed or copied.

To transform a group, one needs to: 0) Construct the corner group; 1) Click the [mv] button in the 'TOPOLOGY' subwindow on to display the group and to attach the group to the cut plane. Now any transformation operation on the cut plane is also done on the group; 2) Click the [hand] button in the 'CUT PLANE' subwindow on to show the visual handles of the cut plane. 3) Manipulate the cut plane with the handles.

There are two group copy modes and, for both of them, the location of the copy is a translation by the average displacement vector of the group to the center of the cut plane. The difference of the two copy modes is that [cp0] does not carry drop back links to the corners in the original group.

To copy without drop back links: 1) Construct and display the corner group; 2) Move the cut plane to a desired location; 3) Press the [cp0] button.

8.3.6 Surface assignments for corners

For a single corner:

There are two methods to perform the same function: the first is to hold down the s key and the second is to click the [S.] button in the 'TOPOLOGY' subwindow to the on state. In this

mode, a click on a corner toggles the corner on or off the current surface; A click on a surface makes it current. Also in this mode, a corner assigned to the current surface has a white center as a marker.

For a group of corners:

The [+][-][x] buttons are for the assignment of the current surface to a group of corners. The [x] button removes all assignments to the current surface. The [+] and [-] buttons need to be combined with the region selection by using the right mouse button. For [+], the corners in the selection region are assigned to the current surface. For [-], the corners in the selection region are unassigned to the current surface.

8.3.7 Excluding object

To exclude a face from becoming a GridPro face, hold down the f key, and then, click any two diagonal corners of the face. A red line connecting the two corners is the visual identification of the excluded face.

Similarly, with the b key, a block can be excluded.

To remove an exclusion, hold down the r key, and then click on the red line representing the exclusion.

8.3.8 Grid density assignments for links

Clicking on the [gden] button in the 'TOPOLOGY' subwindow, a grid density dialog box will be popped up. Now, if a link is clicked, then all affected links will be highlighted and the current setting for this edge group will be shown in the dialog box. One can set the grid density to a different number by typing the number in the dialog box, and then pressing the [apply] button in the box to accept it. Without closing the box, one can continue the procedure for the next link. When all is done, press the [close] button in the box to close it.

8.3.9 Debugging topology and generating grid

When the topology construction is completed, one can start to debug the topology and launch the grid generation process. This is done by selecting the [Ggrid start] button in the [topology] menu. If a topological error exists, an error message is shown in the top portion of the viewing window and the corners and links in the error are highlighted; Otherwise, a confirmation dialog box is shown. If the [ok] button in the box is pressed, the Ggrid command will be launched for the current topology with a default schedule.

Whenever the [Ggrid start] button in the [topology] menu is selected, the current topology is also written into the file '_az.fra' in the launching direction of az, and a default schedule file '_az.sch' is also generated if it does not already exist. One can rename and edit these files to fine tune the run.

8.4 Surface repair with az

A surface of type -tria or -quad can be segmented with az under the surf-panel selection. The surface to be segmented must be made current before segmentation. The segmentation consists of 4 steps: 1) Generate a segmentation path on the surface. The path can be closed (but can not branch in the current implementation); 2) Segment the surface with the path. az will first

check whether a valid segmentation exists with the given path; 3) Select a current surface piece; And 4) Save the current surface piece to a file.

8.5 Viewing grid with az

The data for grid viewing purposes are organized as grid sets and grid sheets. A grid set normally is a data set of multi-block grid generated by GridPro. Multiple grid sets can be loaded into and deleted from the az system with the [grid] menu in the top menu bar.

One of the grid sets in the system is current. By default, it is usually the last one loaded into the system. The current grid set can be changed using the [<][>] buttons in the [grid] button row of the 'CUR' subwindow. The grid panel (lower half of the command panel) only operates on the current grid set except for the buttons for changing the current grid set.

The main grid objects for display are blocks and grid sheets. The display is grid sheet oriented. Initially, all the blocks in the current grid set are displayed and no grid sheet exists for display. Grid sheets must be first generated and optionally followed by trimming, before they can be displayed.

The blocks can also be trimmed with the row of buttons [B.][+][-][*][all] in the 'CUR' subwindow. All subsequent sheet makings will be bounded by the set of active blocks in the current grid set.

The current grid sheet can be trimmed with the row of buttons [SF.][+][-][*][all] in the 'CUR' subwindow.

NOTE: 1) Be careful with the buttons in the 'MAKE SHEET' subwindow. You could unintentionally hit a button multiple times to generate many identical grid sheets, therefore, to slow down the display. If this happens, use the [del] button in the 'CUR' subwindow to delete the extra copies of grid sheets.

2) Know the difference between deleting and undisplay.

3) Use [reload current], if you want only to replace the current grid set with new data and keep the same scene.

4) [print] in the [general] menu in the top menu bar operates only on the grid blocks and grid sheets in display. It first writes the current scene in a file named `sheet.tmp`, then run the (c)sh script `$GRIDPRO/az_mngr/azprint.script` to process it. By default, `$GRIDPRO/az_mngr/azprint.script` will run the '`hide`' utility on `sheet.tmp`, which produces a postscript file named `eps.tmp`. Then `eps.tmp` is sent for printing. The user should customize this script for a proper printer linkage.

5) Properly select the axis center to make the axes visible.

8.6 Setting the grid properties

First, we would like to note that property assignments do not in any way affect the grid generation process.

Internal to GridPro, a property is a number associated with either grid cells or cell faces. The smallest unit that `az`-Graphic Manager can assign a property to is either an elementary block or a face of an elementary block. The property assignments are normally recorded in a file with .pty extension in the GridPro format generated when one saves the grid in the `az`-Graphic Manager environment.

Two mapping tables are kept in the `az`-Graphic Manager. The first maps generic property names, such as 'wall' and 'outlet', to internal property ids. The second maps internal property

ids to specific property names for a particular CFD or FEM solver.

If more than one set of grids is in the system, the property assignments are only for the current grid set.

8.6.1 Default property assignments

For many cases, if the surface properties are properly assigned in the topology construction stage, reading in the grid, and saving it to a particular solver format are all one needs to do for property assignments.

The internal process of default property assignments are as follows: Consider a grid file 'blk.dat' with its connectivity file 'blk.dat.conn' generated from GridPro.

If 'blk.dat.pty' does not exist, simply by reading in 'blk.dat', and saving it, the grid properties will be assigned with default rules and recorded in 'blk.dat.pty'.

The default rules are:

1) All the grid faces on a surface are assigned the property same as the surface property.

Surface Property: A surface has its property defined either by special surface label or by default. If 'blk.dat.conn' contains surface labels that starts with '_', they are interpreted as surface properties. For a surface without labelled property in 'blk.dat.conn', the property is set by the default rule: A one-sided surface has the 'wall' property; And a two-sided surface has the 'inter-block' property.

The special surface labels are defined for surfaces, when the TIL code used to generate the grid is produced from the `az`-graphic Manager. During the topology building phase, the surface property labels can be assigned; Otherwise, a surface is given a default property, either 'wall', 'inter-block' or 'periodic', according to the surface type.

2) All the other faces: If it has one block to it, the 'wall' property is assigned; If it has two blocks to it, the 'inter-block' property is assigned.

8.6.2 Assigning properties

Steps for assigning 2d properties:

0) Start az.

1) Load in a grid by using the [grid] menu in the menu bar. The correct connectivity file must exist.

2) Change the panel to Property Setter by using the [panel=*] menu in the menu bar.

3) Select the current property by using the [<][>] or [list] button in the 'SET 2d property' subwindow. The faces with the current property are show in sea blue color.

4) Change the property assignments with the [.][+][-] buttons in the last row of the 'SET 2d property' subwindow. With one of [.],[+],or [-] is on, each selectable face is shown with a square pickable indicator at the center and with a face skeleton showing the connectivity to the neighbouring faces. When [.] is pressed, a left mouse button click on a face indicator toggles the face between the current property and the previous property of the face. When [+] or [-] is pressed, a regional selection box need to be dragged with the right mouse button. With [+], all the faces with the face indicator in the selection box will be assigned the current property, while with [-], all the faces with the face indicator in the selection box and are of current property will be assigned the previous property of each face.

5) Outputting the grid by selecting the [save as] item of the [grid] menu in the menu bar. In such a case, a file dialog box will be opened. One should select a format, and a file name for saving. Independent of output format, a .pty file which stores the property ids in the GridPro format will be generated.

3d properties can be assigned similarly.

8.6.3 Reuse of property assignments

For a grid (say, in 'blk.new') that has the same topology as, but different grid density and distribution from an older grid (say, in 'blk.old'), the 'blk.old.pty' file for the older grid can be reused.

One can simply copy 'blk.old.pty' to 'blk.new.pty', then load 'blk.new' into az, and save it to a desired format.

Figure 8.1: Display of topology building panel of az

Figure 8.2: Display of surface repair panel of az

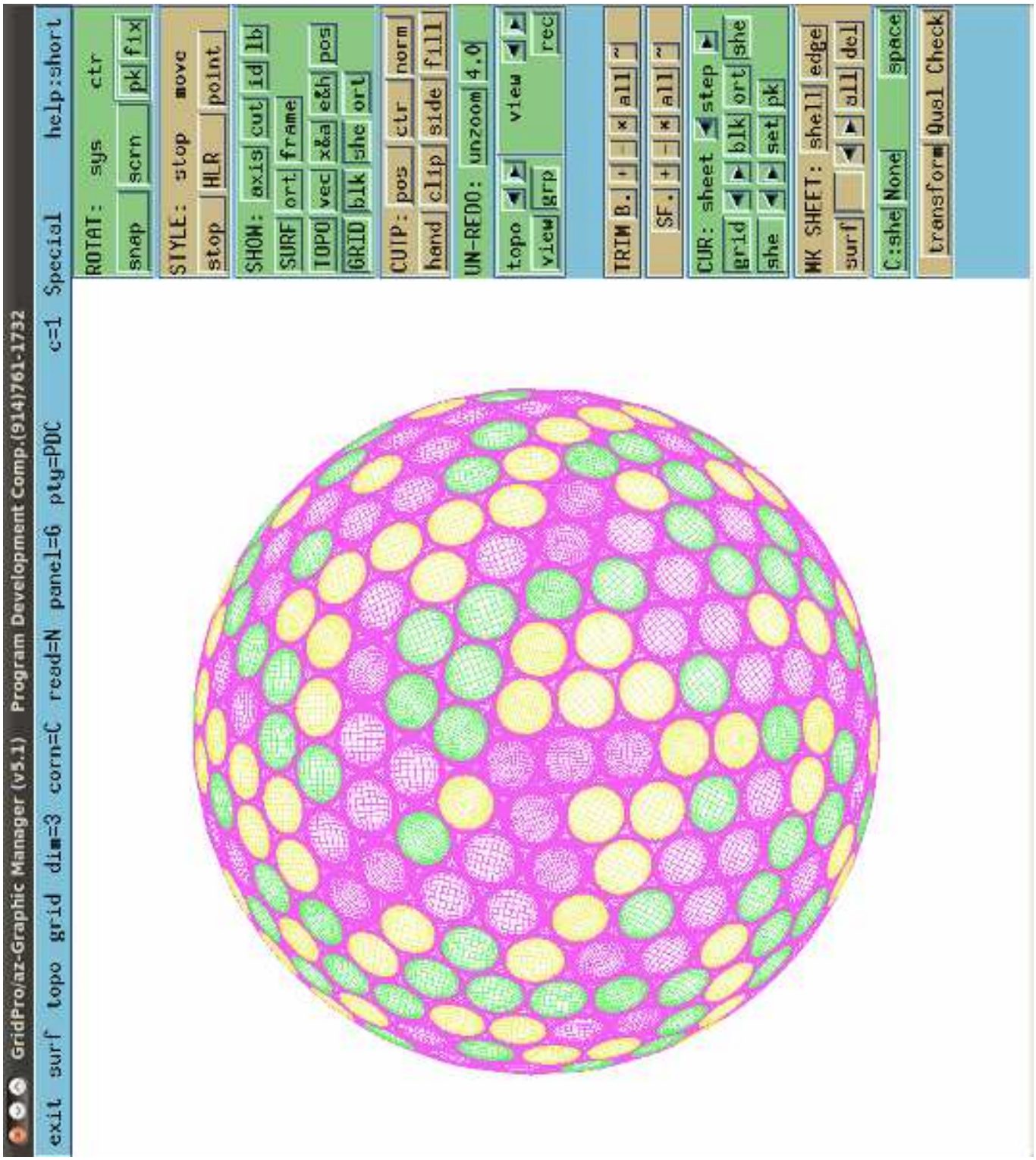


Figure 8.3: Display of multi-block grid viewer panel of az

Figure 8.4: Display of grid property setter panel of az

Part II

Appendices

Appendix A

Quick Reference to Schedule Syntax

A schedule file consists of two sections: A mandatory schedule section and an optional output section. A line is continued to the next by ending with a '\' character.

Conventions: When a *num* is used to refer to a corner or surface, it represents an internal corner or surface id, which can be obtained through running GridPro in the debug mode or with a proper `PRINT..` statement in the topology file. A *label* is a name string defined in the TIL program for a collection of objects of the same type. An object is either a corner, an edge, a face, a block or a surface.

A.1 Schedule section

The schedule section is composed of a sequence of `step`s with the syntax as follows:

`step` *num*: *actions*

where *num* is a step label and *actions* is a sequence of actions that will be executed from left to right. `Step` is a convenient way to group actions to be executed. The steps are executed one by one. If a step is in the gap of the steps listed in the schedule file it is implied that the actions for this step are the same as those of the next nearest step explicitly specified in the file. The syntax of most actions has two basic structures:

1) *–flag* {*obj_list*} {*parameters*} and

2) *–flag obj parameter obj parameter ...* .

The possible actions for a step are:

`-g` ***label num...***
— Change the grid density on edges collected in *label* to *num*. If *num*is 0, the edges with the *label* are redesignated as default edges and assigned the default grid density.

`-g` ***label d*x ...**
— Change the grid density for edges collected in *label* to *d*times the current density rounding off to the integer part.

For the `-g` action, *label* can be predefined words 'ALL' or 'all' to mean all edges, and 'DEF' or 'def' to mean the edges with default grid density (default edges).

`-a` ***label num...***
— Accelerate the convergence of blocks collected in *label* with a loop count *num*.

`-a` ***num1 num2 ...***

— Accelerate the convergence of block *num*1 with a loop count *num*2.

`-S` ***{num}***

— {Set the length of sweep laps to *num*, and} run GridPro for one lap of sweeps.

`-C` ***num***

—change the internal clustering resetting interval to *num* sweeps. That is, the internal cluster parameters will be re–evaluated every *num* sweeps. *num* must be >= 1. The default value is 5.

`-C` ***surf_list d1 d2***

— Setting clustering parameters for surfaces. *surf_list* is a list of surface items separated by one or more spaces. A surface item is either a surface label or a surface range bounded by the internal surface ids. *d*1 and *d*2 are two clustering parameters for the listed surfaces.

Affected Surfaces: Not all surfaces in the list are affected. The rule is that, if at least one of the listed surfaces has its spacing parameter set in the TIL code, then only surfaces with their spacing parameters set in the TIL code will be affected, otherwise all the surfaces in the list will be affected.

About *d*1: If the affected surfaces have the spacings specified, *d*1 is a scaling factor to the spacings, that is, the target spacing for a surface will be `specified_spacing`**d*1. Otherwise, *d*1 is the targeted spacing ratio for the affected surfaces.

About *d*2: *d*2 gives the grid range used for clustering. At most, one layer of block from the surface can be used for clustering. Let K be the number of grid layers in the block. If $d2 < 1$, the number of grid layers affected is K**d*2. Otherwise, the number of grid layers affected is `min`{*d*2,K}.

Turning off Clustering: If either of *d*1 or *d*2 is <= 0, the clustering is turned off for the affected surfaces.

`-c` ***num1*** [***num2***]

—change the algebraic clustering multiple and algorithm.

About *num*1: make the new cell count to to be a multiple of *num*1. *num*1 must be >= 1. The default is 1.

About *num*2: select the algorithm *num*2. *num*1 must be 0, 1, 2, or 3. The default is 2.

0 – linear algorithm with average spacings.

1 – curve fit algorithm with average spacings.

2 – curve fit algorithm with equalized spacings.

3 – linear algorithm with equalized spacings.

`-c` ***surf_list d1 d2***

— Setting the algebraic clustering parameters for surfaces. *surf_list* is a list of surface items separated by one or more spaces. A surface item is either a surface label or a surface range bounded by two numbers that represent two internal surface ids. *d*1 and *d*2 are two clustering parameters for the listed surfaces.

Affected Surfaces: Only surfaces with their spacing parameters set in the TIL code will be affected.

About *d*1: *d*1 is a scaling factor to the spacings specified in the TIL code, that is, the target spacing for a surface will be `specified_spacing`* *d*1.

About *d*2: *d*2 gives the cell growth ratio for clustering. *d*2 must be > 1.0 and < 2.5. The default is 1.5.

`-w` ***{num}***
— If *num* > 0, set the output interval to *num* sweeps. Without *num* , the output interval is unchanged, but output is done once immediately. What to output is determined in the output section of the same schedule file. If `-w ..` is an action before any sweep is run, action "`-w`" will output the initial setup, and action "`-w -1`" will output the initial setup with the surface grid points projected on surfaces.

`-r` ***{num}***
— Readjust surfaces with radius = *num*. Without *num*, the default value for radius is 1.0; The affected surfaces are those marked with `-r` flag in the TIL code.

`-r` ***surf_list num***
— Readjust surfaces with radius = *num*. The surfaces in *surf_list* are readjusted. *surf_list* is a list of surface items separated by one or more spaces. A surface item is either a surface label or a surface range bounded by the internal surface ids (e.g. 2..5 SURF1).

`-scpl` ***surf_list d***
— Set the surface–volume coupling constants for surfaces in *surf_list* to *d*. *d* should be greater than 0 (default = 1.0).

`-s`
— Switch to the script control mode in which actions are read in from the schedule file.

`-m`
— Switch to on-line control mode in which actions are typed in from the keyboard.

`-v` ***num d***
— Set the volume relaxation count per sweep to *num* and set relaxation constant to *d*.

`-sys` "script with args"
— Run a Unix or PC script file (for post processing grid).

`-D`
— Display current run-parameter settings.

`-R` ***parameter value***
— Change the value of run-parameter *parameter* to *value*. For a list of settable *parameter*s, use the `-D` action. Some of the most used parameters are:

`CTRL.SINGULAR` ***d*** – *d* is a number between 1.0 and 2.0. for most cases 1.25 is an adequate choice.

`CTRL.CURVA.STRENGTH` ***d*** – for curvature control. *d* should be in the range [0,2]. This parameter provides an *average* importance of curvature contribution relative to the other grid quality measures in the grid generation.

`CTRL.CURVA.LIMIT` ***d*** – for curvature control. *d* should be in the range [0,2]. This parameter is a supplement to the parameter above. The `STRENGTH` parameter is specified in an average sense. At the local level, the `STRENGTH` is limited by the `LIMIT` parameter to eliminate possible contribution spikes.

`CTRL.CURVA.CUTOFF.LOWER` ***d*** – for curvature control. *d* should be in the range [0,90] and less than `CTRL.CURVA.CUTOFF.UPPER`. *d* provides a relative curvature threshold below which the normalized relative curvature is regarded as 0. Therefore, a larger value makes the distribution less sensitive to the curvature.

`CTRL.CURVA.CUTOFF.UPPER` ***d*** – for curvature control. *d* should be in the range [0,90] and greater than `CTRL.CURVA.CUTOFF.LOWER`. *d* provides a relative curvature threshold above which the normalized relative curvature is regarded as 90 degrees. Therefore, a larger value makes the distribution less sensitive to curvature.

`CTRL.SPACE.STRENGTH` ***d*** – for spacing ratio control. *d* should be in the range [0,2]. This parameter provides an *average* importance of grid smoothness contribution relative to the other grid quality measures in the grid generation.

`CTRL.SPACE.LIMIT` ***d*** – for spacing ratio control. *d* should be in the range [0,2]. This parameter is a supplement to the parameter above. The `STRENGTH` parameter is specified in an average sense. At the local level the `STRENGTH` is limited by the `LIMIT` parameter to eliminate possible contribution spikes.

`CTRL.SPACE.CUTOFF.LOWER` ***d*** – for spacing ratio control. *d* should be in the range [0,1] and greater than `CTRL.SPACE.CUTOFF.UPPER`. This gives the inverse of sensitivity level to the spacing ratio. *d* provides a relative threshold below which the spacing ratio is regarded as 1.0.

`CTRL.SPACE.CUTOFF.UPPER` ***d*** – for spacing ratio control. *d* should be in the range [0,1] and less than `CTRL.SPACE.CUTOFF.LOWER`. This gives the inverse of sensitivity level to the spacing ratio. *d* provides a relative threshold below which the spacing ratio is regarded as 1.0.

NOTE: Schedule file can be modified during a run to steer the run.

A.2 Output section

The output section determines what to output when a '`-w`' action is encountered in the schedule section. The output section is composed by lines beginning with the key word '`write`'. The syntax is:

`write` *which* {*what*} {*where*}

which = Specify which block(s) to write out. There are four choices:
- `-b` *bid* – the bid^{th} block.
- `-c` *cid1 cid2* – the block defined by corners *cid1* and *cid2*.
- `-a` – all the blocks

what = Specify what kind of data to write out. It is a combination of the following flags:
- `-D` *dim* – dimension parameter:
  0 is for binary dump for restart.
  2 is for 2-d grids for 2-d runs.

3 is for 3-d grids.

`-d` – output the dual grids.

`-m` *cid1 cid2*

Or `-m` *dir* – Output a maximum chain of blocks starting with the one defined in *which* and chaining in the direction defined by Face(*cid1*, *cid2*), or by the direction *dir*(=0..5)

The default is 3-d grids.

where = Specify where to output the block(s). There are two choices:

`-f` *fn* – output will over-write file *fn*.

`-F` *fn* – output will be appended to file *fn*.

The default file is '`dump.tmp`' for dumping and '`blk.tmp`' for others.