The PCBCAT Manual

Printed Circuit Board Convection Analysis Tools

Gerald W. Recktenwald

Portland State University
Mechanical Engineering Department

Printed November 11, 1999

Abstract

The Printed Circuit Board Convection Analysis Tools (PCBCAT) are computer programs for predicting the thermal performance of convectively cooled printed circuit boards. The tools consist of three programs: a preprocessor, a depth-averaged (DA) model, and a three-dimensional energy equation model. The flow field is obtained by solving the depth-averaged momentum and continuity equations for the fluid layer above the electronic components. The two-dimensional DA velocity field data is then incorporated into a solution for the three-dimensional temperature field in the fluid and electronic components. By solving the conjugate heat transfer problem the need to specify a heat transfer coefficient is eliminated. By solving the DA flow equations instead of the three-dimensional Navier-Stokes equations the computing time is reduced substantially.

The physical problem to be analyzed is specified via commands in a plain text file which is parsed by the PCBCAT preprocessor. The commands allow the user to describe features of the domain in a succinct, physically intuitive way. User-extensible libraries of material properties and electronic devices are provided to simplify the problem specification and to eliminate errors in creating the problem description file. No reference to node numbers or grid lines is required. Based on the descriptive commands in the user-created text file, the preprocessor generates the grid and additional control variables used in the analysis. Output from PCBCAT analysis codes is in the form of plain text summaries and field data stored in Hierarchical Data Format (HDF). HDF is a standard that can be read by many commercial and public domain visualization packages.

This manual provides a complete reference to the installation and use of the PCBCAT.

Acknowledgments

This research was supported by a grant from Intel Corporation. Dr. Muralidhar Tirumala of the Intel Architectural Development Laboratory supervised the development of these codes and gave technical advice on modeling electronic components. Peter Butler, also from Intel, worked on the first version of the depth-averaged model. Peter's support and encouragement were instrumental in getting the PCBCAT project started.

Jim Padgett helped develop multigrid solvers that are used in the depth-averaged model. Yidong Ma has helped to create turbulence models and an early version of the PCBCAT preprocessor. Peter Gotseff demystified AVS and wrote the `pick_pcbcat` module used in the display and interactive manipulation of HDF data with AVS.

Downloading the PCBCAT Via the Internet

As of this writing the PCBCAT codes are only available for SUN workstations. A port to Windows/NT is in progress. Contact the author (`gerry@me.pdx.edu`) for updated information, and check the ftp site and the PCBCAT home page for new versions.

The latest version of the PCBCAT can be obtained by anonymous ftp with the steps given below, or by pointing a web browser to `http://www.me.pdx.edu/~gerry/PCBCAT/` and following the links to the "Download" page.

To download the files with anonymous ftp use the following commands

```
ftp ee.pdx.edu
      (login with username ``anonymous'' and give
       your full e-mail address as password)
cd pub/users/gerry/PCBCAT
mget *
quit
```

Additional instructions are in the `ReadMe` file on the ftp server.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Printed Circuit Board Convection Analysis Tools (PCBCAT) are a collection of computer programs for analyzing convective heat transfer from electronic packages mounted on printed circuit boards. The PCBCAT are based on a novel computational fluid dynamics (CFD) model developed at Portland State University. A variant of the control volume finite difference technique is used to simulate the coolant flow, the thermal convection in the fluid above the board, and the heat conduction in the components attached to the board. A two-dimensional solution of the flow field is coupled with a three-dimensional model of heat transport by conduction and convection. This model eliminates the need for specification of heat transfer coefficients as is necessary in conventional conduction-based analysis codes. It also is significantly faster and easier to use than fully three-dimensional CFD models.

The PCBCAT have a minimal, text-based user interface. The user builds a model by creating a plain text file containing commands with associated arguments. This file is read by a preprocessor which interprets the user commands and creates another file that controls the analysis codes. Results of the analysis are summarized in the text files. The two and three dimensional field data created by the anlaysis programs is stored in a standard binary format for input to visualization programs.

The programs were developed to run on Unix workstations. Ports to other platforms are planned.

1.1 Overview of the Manual

This manual provides both an introduction to and a reference for the PCBCAT. Details of the underlying mathematical models are presented in separate publications [6, 7]. Chapter 2 is a tutorial that introduces all of the procedures necessary for running the tools. I recommend that you work through the tutorial before attempting to build complex board models.

Chapter 3, *Building Models with PCBCAT*, provides conceptual orientation to model development. It should be useful in explaining how to express a physical feature of a board with the available PCBCAT commands. Sections in Chapter 3 need not be read sequentially. Chapter 4 describes the various output files created by the PCBCAT. You will need to become familiar with these files in order to get useful results from running the analysis codes. Chapter 5, *Command Glossary*, presents the syntax of each PCBCAT command. After working through the tutorial you should skim Chapters 3, 4, and 5 to develop a general awareness of the layout of the manual and the types of information you might find there.

The sample problems presented in the manual are of two basic types. Those in the *Tutorial* involve PCBCAT models of a relatively simple physical situations. The purpose of these tutorial problems is to cleanly present the procedure for specifying a particular feature, e.g., a heat generating device, a partial inlet, etc. The problems in Chapter 6, *Examples*, are complete models of experiments performed by other researchers. These sample problems are presented so that you can run the tools and develop your own opinion of their accuracy and applicability.

Instructions for installing the tools and organizing the working directories on the hard disk of your workstation are given in the Appendix A. I recommend that if you are not familiar with installing software on Unix workstations that you consult with your system administrator.

1.2 Typographic Conventions

The manual contains descriptive text and Unix commands, though very few system level commands are necessary to run the PCBCAT. You should know how to make and maneuver in directories and run a text editor such as `vi` or `emacs`. In this manual Unix commands that you are asked to enter are shown in monospaced font. For example, to list the files in the current you type

```
ls
```

To execute this command enter the letters "ls" on the command line and press the "return" key. Whenever the output of a Unix commands is shown in the manual it is also displayed in monospaced font. For example, if you had typed the preceding command in the `pcbcat` directory the output would probably look like

```
bin      example    tutorial
```

Monospaced font is also used for the names of PCBCAT commands, and for the names of files created by the PCBCAT.

1.3 A Quick Guide for the Impatient

This manual is intended to be direct and to the point, so there really is no quick guide. For users who think in terms of check lists here are the steps to get up and running with the PCBCAT

1. Install the PCBCAT as described in Appendix A.
2. Work through at least the first tutorial problem in Chapter 2.
3. Identify an example problem from Chapter 6 that is closest to the problem you are trying to solve. Copy the corresponding input file from the `example` directory and modify it until the model it describes fits your situation.
4. Consult the remaining chapters as necessary.

Chapter 2

Tutorial

This chapter is an introduction to the PCBCAT tools and the process of building models of circuit boards for the tools to analyze. The example problems presented here are simplified models of electronic cooling situations. Though simplistic, these models contain the essential features of more complicated board layouts. I recommend that you work through the first three models in succession. The example files in Chapter 6 will be easier to read and edit after you work through the tutorials.

I use the word "model" to refer to the collection PCBCAT commands that specify a physical problem of interest. These commands are contained in a plain text file that I will refer to as a "user input file". The process of building a model involves editing the commands in the user input file.

In the following sections I assume that you have already installed the PCBCAT. Consult Appendix A for installation instructions. Before proceeding with the tutorial make sure you have set the PCBCATDIR environment variable as described in Appendix A.

2.1 Cooling a Single Block

In this section I present the steps necessary to run the PCBCAT for a simple cooling problem involving laminar and turbulent flow past a single heated block in a channel. This tutorial exercise demonstrates

- the basic components of a PCBCAT model
- the structure of the user input file
- how to run the PCBCAT tools
- the files generated by the PCBCAT tools
- how to switch between laminar and turbulent flow models

2.1.1 The Physical Model

The physical situation is depicted by the sketches in Figure 2.1 and Figure 2.2. Fluid enters the domain through the y-z plane at $x = 0$ and exits through the y-z plane at $x = 30$ cm. I use the compass point convention to refer to directions and faces of the domain (see Section 3.2.1). Imagine a compass lying in the x-y plane, and oriented such that north is in the direction of increasing y. Then east is in the direction of increasing x. In the plan view representation in Figure 2.2, the

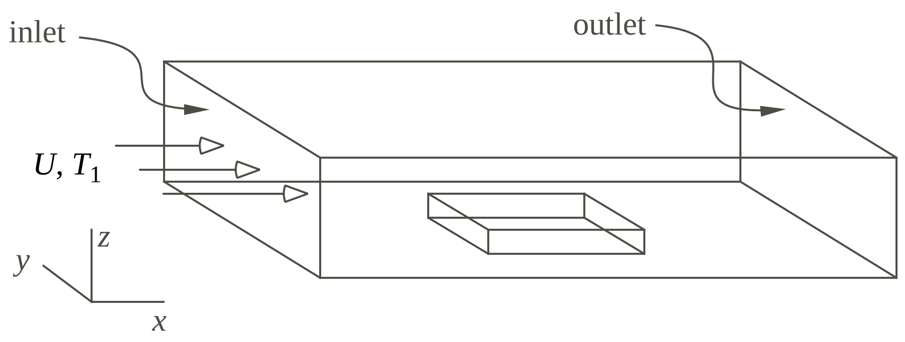

Figure 2.1: Schematic of the physical problem for tutorial exercise number 1.

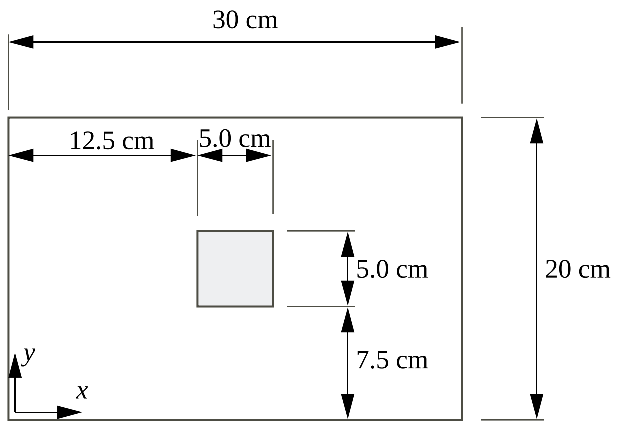

Figure 2.2: Plan view showing the block location for tutorial exercise number 1.

far right boundary is the east face of the domain, and the far left boundary is the west face of the domain. The top and bottom boundaries of the plan view are the north and south faces. This naming convention can be extended to the three-dimensional domain in Figure 2.1. The x-y planes at $z = 0$ and $z = z_{max}$ are called the "up" and "down" faces since up and down are directions like north, south, east and west. Using this direction naming convention the domain can be described as having an inlet on the west face, an outlet on the east face, and a block attached to the "down" face.

2.1.2 The `block1` Input File

A block is the simplest model of an electronic device. Although you could build your own input file from scratch I suggest that you start with the file `block1`, which is found in the `tutorial` directory. You will not need to change anything in this file to run the first exercise. Before running the tools, however, look at the contents of the `block1` file with a text editor such as `vi`, `emacs` or the Sun text editor. For your convenience the `block1` file is also presented in Figure 2.3. The file contains blank lines and comment statements. A PCBCAT comment begins with the `#` character, and can be located anywhere in the input file. The preprocessor ignores all text from the `#` character to the end of the line. Blank lines, like comments, can be added anywhere to enhance readability. Lines that are not comments or blank must be valid PCBCAT commands.

All PCBCAT commands begin with keywords, which must be in all capital letters. Examples of keywords in Figure 2.3 are `FILE_NAME`, `DOMAIN`, `CV_SIZE` and `COOLANT`. Chapter 5 gives a complete description of each PCBCAT command. Following the keyword is a sequence of arguments that supply information necessary for the preprocessor to execute the corresponding command. Arguments are either character strings or numbers. Some commands have no arguments, e.g., `INTERACTIVE` and `VERBOSE`, while other commands have one or more arguments. For example, the `FILE_NAME` command has one string argument, which is the base name of output files created for this model. The `DOMAIN` command has three numerical arguments, x_{len}, y_{len}, and z_{len}, which are the physical lengths of the domain in the x, y and z directions.

The commands in Figure 2.3 will now be described in the order of their appearance. The first command, `FILE_NAME`, specifies a text string, "`block1_`", as the base name from which the PCBCAT will construct output file names[1]. The `DOMAIN` command sets the extent of the overall calculation domain. It has three arguments which are floating point values specifying the x, y and z dimensions of the domain in meters. The `CV_SIZE` command sets the maximum allowable size of the control volumes in each coordinate direction. These tolerances, and the geometric data for objects in the domain are all that is needed to specify the computational grid.

The next group of preprocessor commands set parameters that affect the overall model. The `COOLANT` command tells the preprocessor that the fluid in the domain has the thermophysical properties of air, as defined in the material database. The `ITER_CONTROL` command sets the convergence tolerance and the maximum number of iterations for the DA and 3D models. The "`medium`" convergence tolerance defines several numerical tolerances on residuals of the linear equations that are solved in the analysis programs (see section 3.5.2). The `FLOW_REGIME` command specifies whether or not turbulence model is to be used. When the argument is "`laminar`" the effective viscosity and diffusivity are equal to their molecular values. When the argument is "`turbulent`", the turbulence model described in [6] is used.

The `FLOW_FIELD` command determines whether a uniform flow field or the depth-averaged flow field is to be used in solution to the 3D energy equation. A uniform flow field is not meaningful in all cases. In this simple example the results of solving the energy equation with a uniform flow field could be compared with the results of using the DA flow field. The "`depth_ave`" argument means that the DA flow field will be calculated. Finally, the `FLOW_PROFILE` command is used to select the z-direction variation of the velocities. Choosing "`fully_dev`" for the argument of the `FLOW_PROFILE` command means that the vertical profiles will be computed with the fully-developed profile consistent with the argument of the `FLOW_REGIME` command.

The `INLET` command has several parameters. The first is "`west`", a text string that specifies the face on which the inlet is located. The next four arguments of the `INLET` command give the location of the inlet on the west face. For this simple example the inlet covers the entire face. It's possible, as shown in the next example, that inlets and outlets cover only part of a domain face. The last four arguments of the `INLET` command assign the inlet temperature (10℃), and inlet velocity components, $(v_x, v_y, v_z) = (1.131, 0, 0)$.

[1]The underscore character appears at the end of "block1" because the grid size parameters are appended to the argument of the `FILE_NAME` command to create the file name. Ending the `FILE_NAME` argument with a numeric character might result in a potentially misleading name of the file in the operating system. Suppose that the analysis was performed on a 133 × 23 grid and the filename was specified with "`FILENAME block1`". Then one of the output files created by the PCBCAT codes would be named `block133x23.DAout`, which would appear to contain the results of a calculation on a 133 × 23 grid, not a 133 × 23 grid.

```
#  PCBCAT example problem "block1":  flow past a single heated block.

#  Set the base name used to construct output file names
FILE_NAME   block1_

#  Define a 30 x 20 x 4 (cm) computational domain
DOMAIN  0.30  0.20  0.04        #  dimensions in meters

#  Set the largest allowable control volume sizes
CV_SIZE 0.01  0.01  0.002

#  Global parameters that control the solution
COOLANT       air
ITER_CONTROL  medium    50
FLOW_REGIME   turbulent
FLOW_FIELD    depthAve
FLOW_PROFILE  fully_dev

#  The inlet and outlet cover the entire east and west boundaries
INLET   west  0.0  0.20  0.0  0.04  10.0  1.1310  0.0  0.0
OUTLET  east  0.0  0.20  0.0  0.04

#  North, South, Up and Down boundaries are adiabatic walls
BOUND  north  fluxBC  0.0   wall
BOUND  south  fluxBC  0.0   wall
BOUND  up     fluxBC  0.0   wall
BOUND  down   fluxBC  0.0   wall

#  Locate a 5cm x 5cm x 1cm aluminum block in the center of the duct
BLOCK  heater  aluminum  2.0  1  0.125  0.05  0.075  0.05  0.01

#  Probe locations are scaled to overall domain dimensions
PROBE  relative  0.50  0.5  0.125
PROBE  relative  0.75  0.5  0.5
```

Figure 2.3: PCBCAT input file for cooling of a single block in a channel.

Boundary conditions are prescribed with the `BOUND` command. The first argument specifies the face, the second and third arguments define the thermal boundary conditions, and the fourth argument prescribes the hydrodynamic boundary condition. Since the east and west boundaries are already defined as the inlet and outlet, respectively, only the remaining four domain boundaries need to be treated. All four are adiabatic, as indicated by the "`fluxBC  0.0`" values of the second and third arguments. The "`wall`" value of the fourth argument selects a solid boundary for which no-slip hydrodynamic boundary conditions will be applied.

The heated block is specified with the `BLOCK` command. The arguments specify the block material properties ("`aluminum`"), its total power dissipation, ("`2.0`" Watts), it's heating status ("`1`" indicating that the heater is on), and the location and size of the block. The heating status is redundant with the total power dissipation, in apparent violation of the principle that the user should have to specify the absolute minimum amount of information necessary. In this case the design principle was relaxed so that a user could turn off a heating element without having to reset its nominal power setting.

The last two lines of the input file define two probes via the `PROBE` command. The first argument of the `PROBE` command is either "`relative`" (as in Figure 2.3) or "`absolute`". A "`relative`" probe position means that the preprocessor will interpret the last three arguments as fractions of the x, y and z dimensions of the domain. For example the first probe in Figure 2.3 is located in the geometric center of the x-y plane and one eighth of the z-direction domain length from the bottom plane. The alternative is to locate the probe by the absolute coordinate position, in meters.

2.1.3 Running the PCBCAT

To run the PCBCAT with the block1 model type

```
cp $PCBCATDIR/example/block1 .
runcat block1
```

Do not skip the period at the end of the `cp` command shown above.

The computer will hesitate for a minute or so (depending on the speed of your workstation) while the preprocessor and analysis codes are running. Figure 2.4 shows the typical screen output for running the `block1` model.

2.1.4 Output Files Generated by the PCBCAT tools

The PCBCAT tools create plain text and binary files. For now I will only be concerned with the plain text output. Refer to Chapter 4 for more information on both types of files.

The plain output of the depth-averaged model is in file `block1_33x23.DAout`

2.2 Cooling an Array of Blocks

This exercise involves analyzing laminar and turbulent flow past an array of four heated blocks in a channel. This is a small variation on the preceding exercise. This exercise demonstrates how to

- locate multiple blocks on a board
- selectively turn heat generation of a block on or off
- define inflow and outflow boundaries that do not cover an entire face of the domain.

```
euler: runcat block1 example

    PCBCAT preprocessor successfully parsed block1 input file

    DA model finished in 10.833333 seconds

    Min and max field values are 10.000000 and 80.237785

    3D model finished in 8.550000 seconds

euler: ls block1_*
block1_.dav          block1_33x23.DAout         block1_33x23x10.geom
block1_.prout        block1_33x23x10.3Dout      block1_33x23x10.hdf
euler:
```

Figure 2.4: Screen output from the PCBCAT while running the model described in the "block1" user input file. The execution times will depend on the speed of the computer. Note that the command line prompt (for this computer) is "euler:". Your command line prompt will be different.

The steps are

- `runcat tutorial/block4`
- edit block4 to turn off one of the heaters
- copy block4 to block4r
- restrict the inlet

Chapter 3

Building Models with the PCBCAT

This Chapter provides information on how to create PCBCAT models. In this context a model consists of the list of PCBCAT keywords and arguments that specify features of a physical situation, and that control the algorithms in the PCBCAT codes. Whereas the information in Chapter 5 is organized according to each PCBCAT command, the information in this Chapter is organized by the ways in which commands might interact. Specifically this Chapter describes the structure of the input file, the conventions used in defining model features, the hierarchy in which flow field descriptions are organized, and the general issues concerning control of the analysis programs. This chapter is intended to be a general reference. Its subsections can be read in any order.

3.1 Organization of Model Information

3.1.1 Structure of the Input File

There are very few constraints on the structure of the input file. It must be a plain text file. Each line must be either (1) blank, (2) a comment statement, or (3) a valid keyword followed by valid arguments for that keyword.

The user input file can contain any number of blank lines and comment statements. PCBCAT comments begin with the `#` character, and can be located anywhere in the input file. The preprocessor ignores all text from the `#` character to the end of the line. Blank lines, like comments, can be added anywhere to enhance readability. Lines that are not comments or blank must be valid PCBCAT commands.

Commands always begin with a keyword which is always in all captial letters. You can think of the keywords as the command name. Keywords can be specified in nearly any order. Some keywords toggle state changes in the preprocessor. These keywords affect the way that arguments of subsequent keywords are interpreted by the preprocessor.

The `DOMAIN` command is mandatory and it can only be preceded in the user input file by commands that do not refer to the dimensions of the domain. Any command that locates an object (e.g. `BLOCK`, `DEVICE`, `INLET`, and `PATCH`) must follow the `DOMAIN` command because the preprocessor checks to make sure each object fits into the domain. If your user input file contains one of these commands before the `DOMAIN` command the preprocessor will print an error message and stop.

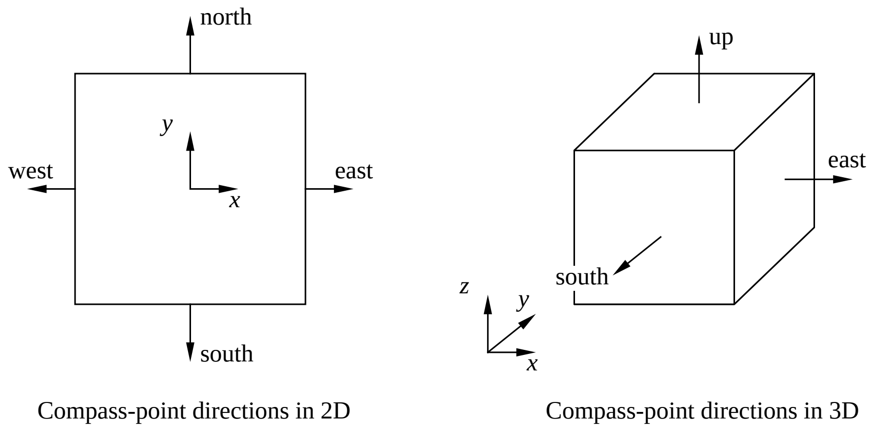
north
y
x
west
east
south
Compass-point directions in 2D
up
east
south
z
y
x
Compass-point directions in 3D

Figure 3.1: Face and direction naming convention used to identify faces of objects in the domain.

Although, with the exception of state-changing commands, the PCBCAT commands in the user input file can be in any order, enforcing a little structure will help reading and debugging the input files. We recommend that you write your input files using the style of user input files given in the *Tutorial* and *Examples* chapters. The beginning of the input file should contain the domain size, coolant type and parameters that control the numerical solution. Following that the boundary conditions, including inflow and outflow boundaries, should be specified. The rest of the user input file should list the `BLOCK`s, `DEVICE`s, etc., that are located on the board. Regardless of whether you follow this style, choosing a style and sticking with it will make your use of the PCBCAT more trouble free. Documentation of the model with comment statements is also highly recommended.

3.2 Specifying Physical Features

3.2.1 Face Naming Conventions

In order to apply boundary conditions or to locate objects in the domain a convention for indicating coordinate directions is needed. The PCBCAT commands refer to directions with the compass point naming convention, which is illustrated in Figure 3.1. First consider the two-dimensional domain depicted in the left half of Figure 3.1. The “east” and “west” faces correspond to the maximum and minimum values of x, respectively. Similarly the “north” and “south” faces correspond to the maximum and minimum values of y, respectively. This convention extends to three dimensions as shown in the right half of Figure 3.1. The “up” and “down” faces correspond to the maximum and minimum values of z, respectively.

3.2.2 Domain Dimensions

The calculation domain is always a three-dimensional, brick-shaped region of space. The size of the domain is given in meters with the `DOMAIN` command. The x, y, and z axes form a right-handed coordinate system. The face-naming conventions (see Section 3.2.1) are defined in terms of the coordinate directions so it is important that the orientation of the calculation domain be consistent with the placement and orientation of boundary conditions, `DEVICE` objects, etc.

Command	Description
`BOUND`	specify the boundary condition on an entire face of the domain
`INLET`	locate the inlet and specify inlet velocity and temperature
`OUTLET`	locate the outlet
`PATCH`	locate a boundary condition on a surface of the domain

Table 3.1: PCBCAT commands used to specify boundary conditions.

B.C. Type	Applies to
`DIRICHLET`	thermal boundary conditions only
`FLUXBC`	thermal boundary conditions only
`SYMMETRY`	hydrodynamic or thermal boundary conditions
`WALL`	hydrodynamic boundary conditions only

Table 3.2: Boundary condition type specifiers for use with commands in Table 3.1.

3.2.3 Boundary Conditions

There are three common mathematical boundary conditions: Dirichlet, Neumann and mixed. These may be independently applied to each of the field variables on each surface of the calculation domain. The temperature field will be used for purposes of demonstration. Dirichlet conditions prescribe a fixed value on the boundary. For example the temperature at the inlet may be held at $15^\circ C$. Neumann conditions involve a specified slope in the direction normal to the boundary. A zero slope in the temperature profile corresponds to an insulated boundary. A nonzero slope corresponds to a prescribed heat flux. Mixed boundary conditions involve a relationship between the slope and the value at the interface. A convective thermal boundary is an example of a mixed boundary condition. The current version of the PCBCAT does not provide a mechanism for specifying mixed boundary conditions.

Boundary conditions are applied with one of the commands in Table 3.1. The `BOUND` command applies a boundary condition to an entire face of the domain. The `INLET`, `OUTLET`, and `PATCH` commands apply boundary conditions on a rectangular subregion on any one of the six bounding surfaces of the domain. The `BOUND` and `PATCH` commands specify the thermal and hydrodynamic boundary conditions on solid walls using the arguments in Table 3.2. The thermal and hydrodynamic boundary conditions are specified with separate parameters. Thermal boundary conditions may be either `DIRICHLET`, `FLUXBC` or `SYMMETRY`. Hydrodynamic boundary conditions may be either `WALL` or `SYMMETRY`. If you wish to specify flow across a boundary segment you must use either the `INLET` or the `OUTLET` commands.

3.2.4 Inlets and Outlets

Flow enters the domain at an inlet, and leaves at an outlet. In the current version of the PCBCAT only one inlet and one outlet may be specified for a given problem.

An outlet boundary is specified with the `OUTLET` command. The arguments of the `OUTLET` command are the position of the four corners of the outlet surface. In the PCBCAT analysis codes zero-gradient boundary conditions are applied to all dependent variables on the outlet surface. The magnitude of the normal component outlet velocity is also constrained by an overall mass balance.

At each outer iteration of the DA model

$$\sum_{inlet} \left(\vec{V} \cdot \hat{n} A\right) = \sum_{outlet} \left(\vec{V} \cdot \hat{n} A\right)$$

where $\hat{n}$ is the outward normal to the surface area, A.

In the 3D model the outlet velocity depends on how the flow field in the domain is computed (cf. section 3.4). The most basic characteristic of the flow field is controlled by the `FLOW_FIELD` command. If the flow field is uniform (`FLOW_FIELD uniform`) the velocity vector at the outlet is uniform and equal to the velocity vector at the inlet. If the flow field is computed with the depth-averaged model (`FLOW_FIELD Depth_ave` the velocity profile over the inlet surface if further determined by the arguments of the `FLOW_PROFILE` and `FLOW_REGIME` commands.

An inlet boundary is specified with the `INLET` command. In addition to the position of the inlet surface, the arguments of the `INLET` command define the velocity and temperature of the fluid at the inlet. Note that all three components of the inlet velocity vector must be prescribed at an inlet surface. Regardless of the arguments of the `FLOW_FIELD`, `FLOW_PROFILE`, and `FLOW_REGIME` commands, the velocity profile over the inlet surface is uniform. In other words the inlet velocity components over the entire inlet surface are equal to the velocity components specified with the `INLET` command.

3.2.5 Objects in the Domain

All physical features in the calculation domain are specified via their name, geometric dimensions, and additional properties unique to each object type. There are four types of objects: block, device, patch and probe. Each object types correspond to PCBCAT commands of the same name.

3.2.6 Blocks

Blocks are brick-shaped objects that may or may not have an internal volumetric heat source. Blocks are used to simulate simplistic electronic devices or solid obstacles that deflect the coolant. Blocks are always attached to the "down" surface of the domain. A block is added to the domain with the `BLOCK` keyword.

3.2.7 Devices

Devices are models of electronic components. Adding an electronic device to the calculation domain requires that the geometric dimensions, material properties, and rated power of the device is already defined in the device database. See Chapter 7 for instructions on creating entries in the device database. The `DEVICE` keyword selects a device from the database and places it in the domain. Devices are always attached to the "down" surface of the domain.

Locating a device in the domain requires choosing it by name, locating it on the x-y plane of the PCB, and specifying its orientation. Entries in the device database are defined in terms of an internal Cartesian coordinate system, which is represented by the (x_d, y_d) axes in Figure 3.2. The orientation of a device is prescribed by the direction in which its internal x-axis points. Thus, a device with an "east" orientation has its internal x-direction aligned with the x-direction of the coordinate system that defines the printed circuit board. A device with a "west" orientation is rotated 180 degrees so that its internal x-direction is aligned with the negative x-direction of the PCB. Devices with "north" and "south" orientations are rotated 90 and 270 degrees, respectively, as shown in Figure 3.2. The orientation of a device is chosen with the `orientation` argument of the `DEVICE` keyword. The `orientation` argument is mandatory, even for east-oriented devices.

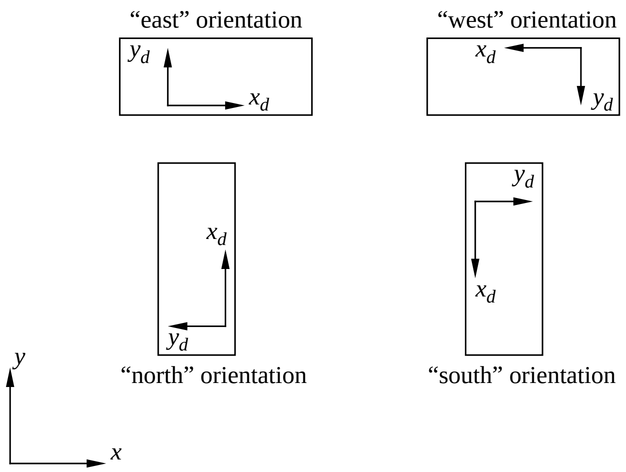

Figure 3.2: Allowable orientations of devices in the calculation domain.

3.2.8 Patches

Patch objects provide a flexible approach to applying boundary conditions. Patches are attached to one of the six bounding surfaces of the calculation domain. There is no way to specify an internal, heat-generating patch.

3.2.9 Probes

A probe is a passive object used in selectively reporting the velocity and temperature at a point. A probe does not affect the solution, and it may be located anywhere in the domain. There is no internal limit on the number of probes specified by the user.

After the `3D` code completes the solution to the temperature field, the velocity and temperature at each probe location is computed by interpolating between the velocity and temperature values at the nearest grid points. The location of a probe is specified with the `PROBE` command.

3.3 Coolant and Material Properties

Thermophysical properties of all materials, fluid and solid, are specified in a user-maintained material database. The user does not enter material property data directly in the user input file. Instead the user specifies a material name, which corresponds to an entry in the material database. For example, for an air-cooled PCB the input file would contain the command

```
COOLANT air
```

The material database must contain an entry specifying the thermophysical properties of air. The default database contains the entry

```
air     1.2047    1004.0    25.63e-3    1.817e-5
```

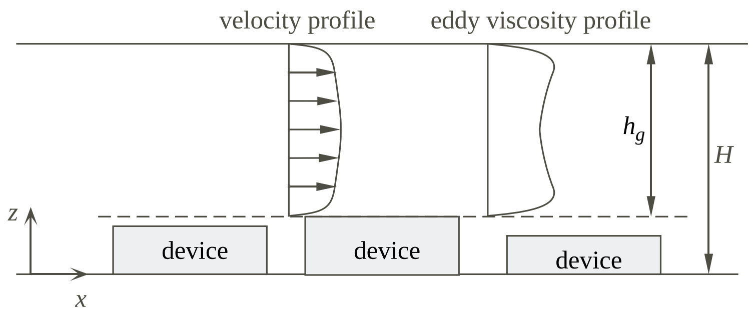

Figure 3.3: Two dimensional (elevation view) representation of the velocity profile used in the PCBCAT model. The shaded boxes labeled “device” are electronic components attached to the bottom surface of the domain.

where the first entry is the name of the material, “air”, and the numerical entries are, respectively the density, specific heat, thermal conductivity, and viscosity of the air. If the preprocessor cannot match the material name in the input file with a material in the database it prints an error message and aborts the analysis. Comparison of material names is *not* case sensitive.

For additional information on the material database refer to Chapter 8.

3.4 Specifying Flow Fields

The user chooses the type of velocity profile with a combination of three keywords:

- `FLOW_FIELD`
- `FLOW_PROFILE`
- `FLOW_REGIME`

The `FLOW_FIELD` keyword determines how the flow varies in the x-y plane. Choices are “`no_flow`”, for no flow anywhere in the domain, “`uniform`”, for a uniform (constant) velocity vector everywhere in the domain, and “`depth_ave`”, for a solution to the depth-averaged velocity field in the x-y plane.

The `FLOW_PROFILE` determines how the velocity field varies in the z direction, i.e., the direction normal to the plane of the PCB. This command only makes sense if the flow field is computed by the depth-averaged model.

For any circuit board, define the fluid gap, h_g as the distance from the top of the tallest electronic component to the lid of the fluid layer. This fluid gap is depicted in Figure 3.3. We then arbitrarily assert that all the flow is confined to this gap. This is at least qualitatively consistent with experimental and numerical results obtained by other researchers, see, e.g. [2, 4]. The assumption of a uniform gap for the flow is used only in the 3D model. The DA model includes the changes in bottom surface topology. The velocity field satisfying the depth-averaged continuity and momentum equations reflects the blockage effects of electronic devices and other obstacles.

The `FLOW_REGIME` determines whether a laminar or turbulent flow model is used. This in turn determines the effective viscosity and the shape of the profile.

3.4.1 Fully-Developed Laminar Velocity Profiles

3.4.2 Fully-Developed Turbulent Velocity Profiles

3.5 Controlling the Solution

The PCBCAT consist of numerical models of momentum, mass and energy conservation. These conservation principles are approximated by the control volume finite difference method [5]. The user can control the accuracy of the numerical approximation by adjusting the grid size (resolution) and the tolerance on the iterative solution procedure. The grid and convergence tolerance interact in affecting the overall behavior of the solution. In this section these issues are discussed and a strategy for obtaining efficient and reliable solutions is presented.

In general the time to reach a solution will increase as the grid resolution increases and as the convergence tolerance is tightened.

The amount of memory (RAM) needed to solve a given problem increases as the grid resolution increases. Adding objects to the domain causes a much smaller increase in memory requirements than does increasing the grid resolution. There is no limit on size of the grid or the number of objects in a given problem so long as the computer running the model has sufficient memory. If you are unable to complete an analyis because your computer does not have adequate memory you will have to decrease the grid resolution by *increasing* the arguments of the `CV_SIZE` command.

3.5.1 Grid Resolution

The number of control volumes in the domain is determined by the overall extent of the domain, the number of objects and their placement in the domain, and the parameters set in the `CV_SIZE` command. For a given problem only the `CV_SIZE` parameters can be arbitrarily adjusted, but it is important to recognize the other factors that affect the grid used in the analysis.

PCBCAT commands are used to locate objects in the calculation domain without direct reference to the grid system used in the analysis.

After the preprocessor reads the commands from the input file it assembles three lists from edges of all objects in the domain. One list is constructed for each coordinate system. Each list is sorted and redundant edges — those within the tolerances specified in the `DISTOL` command — are eliminated. The result is the three edge lists that are required to locate all objects to within the user-specified tolerance. These edges form a relatively coarse three-dimensional grid called the object grid. Figure 3.4 shows the projection of an object grid on the “down” face of the domain. The parameters in the `CV_SIZE` command are used to subdivide the object grid further into what is called the analysis grid. The analysis grid is consistent with the object grid in that it has control volume faces that are aligned with the edges of the objects in the domain.

This grid definition procedure has important consequences for PCBCAT model development. If you specify sufficiently large tolerances with the `CV_SIZE` command the analysis grid and the object grid will be identical. This is an advantage because the finite volume method used in the analysis will satisfy the mass, momentum and energy conservation (to within user tolerance) even on the coarsest grids. Thus, during the early phase of model development the goal in setting the grid size is to make it coarse enough that the execution time will be short.

As you refine the grid the number of control volumes will not necessarily increase in proportion to the reduction in the `CV_SIZE` parameters. This is especially true for complex models consisting of many physical objects because the object grid is unlikely to have uniform spacing. The representative object grid in Figure 3.4 shows that whenever objects are nearly, but not exactly aligned, the object

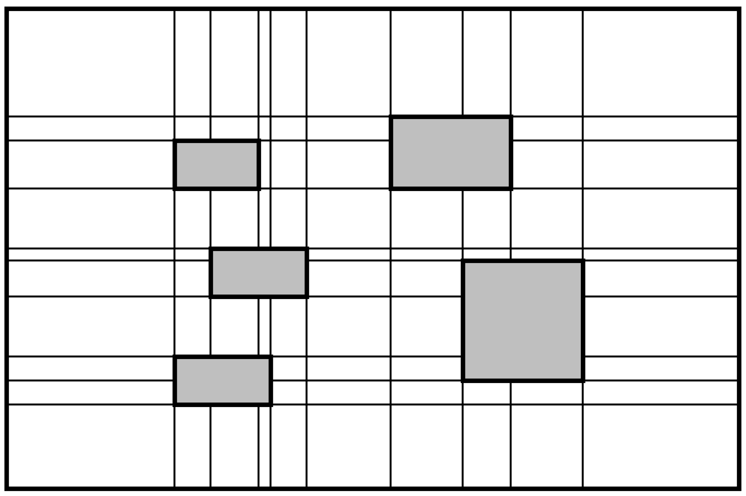

Figure 3.4: Object edges form an object grid. This figure shows a two-dimensional grid defined by five arbitrary objects (shaded).

grid will have closely spaced grid lines. In this situation, halving the `CV_SIZE` parameters will not necessarily double the number of analysis grid lines.

For a given problem, the domain size is set by the dimensions of the printed circuit board you are trying to model. The PCBCAT analysis grid size depends on these dimensions *and* the arguments of the `CV_SIZE` command. For two problems of different physical size, the same settings in the `CV_SIZE` command will result in different numbers of control volumes. In other words the optimal grid tolerances will not be universal for problems with significantly different physical scales.

3.5.2 Convergence

The PCBCAT models involve solution of large sets of linearized algebraic equations. These equations are derived from the control-volume finite-difference approximation to the depth-averaged flow equations and the three-dimensional energy conservation equation [6].

In the depth-averaged model the SIMPLER algorithm [5] is used to enforce mass conservation while the momentum and continuity equations are solved sequentially. The linear equations are solved iteratively: from an initial guess the solution is updated until the equations are solved to within a user-specified tolerance. In the PCBCAT these tolerances are chosen by a specifying "loose", "medium", or "tight" as the first argument of the `ITER_CONTROL` command. To understand the meaning of these terms it is first necessary to introduce some terminology.

Convergence of the u, v and p equations.

The depth-averaged flow model solves equations for the depth-averaged velocity components, $\bar{u}$ and $\bar{v}$, and the depth-averaged pressure, $\bar{p}$. These field variables are defined on a staggered, finite-volume grid. Each control volume roughly[1] corresponds to a discrete value of each of $\bar{u}$, $\bar{v}$, and $\bar{p}$.

[1]The pressure boundary conditions and the staggered control volumes lead to slight disparities in the number of unknown velocities on the boundaries.

The SIMPLER algorithm involves sequentially solving the equations for each of the discrete fields. The set of equations for any one of these discrete fields can be written

$$Ax = b \tag{3.1}$$

where A is a matrix of coefficients, x is a vector of unknowns, and b is a vector of source terms. The x vector is obtained by writing the set of nodal values for one of the fields, say $\bar{u}$, in order from one corner of the domain to the opposite corner. The finite-volume method allows the partial differential equation for the conservation of the x variable to be approximated by the set of linearized equations 3.1. One step in the numerical approximation, therefore, involves converting the continuous problem into a discrete problem amenable to solution by digital computer. This step is done for you by the PCBCAT analysis codes. Another step involves solving this linear equation set. This second step is controlled by the parameters of the `ITER_CONTROL` command.

An iterative solution involves constructing a sequence of approximations, $x^{(k)} : k = 1, 2, \ldots,$ which approaches the solution to equation 3.1 as k increases. Here k is the iteration counter, and $x^{(k)}$ is understood to be the approximation to the x vector at iteration k, not x raised to the k^{th} power. Even for very large values of k we do not expect equation 3.1 to be solved exactly. The closeness of the k^{th} iterate to the true solution can be measured by the size of the residual vector

$$r^{(k)} = b - Ax^{(k)} \tag{3.2}$$

Only if $x^{(k)}$ is exactly equal to the true solution of equation 3.1 then $r^{(k)}$ will be identically zero. In practice $r^{(k)}$ will decrease at the iterations procede. The tolerance specified with the `ITER_CONTROL` command determines when the residual is small enough.

To measure the size of the residual we use the L_2 norm

$$\|r^{(k)}\|_2 = \sum_{i=1}^{n} r_i^{(k)}$$

where n is the number of internal nodes in the calculation domain. If A is a well-conditioned matrix, the approximate solution, $x^{(k)}$ will be close to the true solution when the normalized residual

$$\eta^k = \frac{\|r^{(k)}\|_2}{\|b^{(k)}\|_2} \tag{3.3}$$

is small [3].

The iterations in the depth-averaged model are repeated until $\eta_u^k < \epsilon$, $\eta_v^k < \epsilon$, and the mass correction tolerance (see following section) is met. The first argument of the `ITER_CONTROL` command corresponds to the ϵ values listed in Table 3.3. The same value of ϵ is applied to the η_u^k and η_v^k. A separate convergence tolerance is not applied to the discrete p equation because the the mass correction tolerance is a more sensitive indicator of convergence than the normalized residual of the p equation. In any case the η_p^k values are printed as part of the convergence history in the "`DAout`" file (see Section 4.3).

Convergence of mass corrections.

The depth-averaged momentum and mass conservation equations are coupled, non-linear partial differential equations. The SIMPLER algorithm [5] is a robust procedure for resolving the coupling in this set of equations. In SIMPLER, the discrete equations for $\bar{u}$, $\bar{v}$, and $\bar{p}$ are solved sequentially. First the equation for $\bar{p}$ is solved while the values of $\bar{u}$ and $\bar{v}$ are temporarily held constant. Then the

	ϵ	Energy Balance (Watt)
Loose	5×10^{-3}	0.5
Medium	5×10^{-5}	0.05
Tight	5×10^{-7}	0.025

Table 3.3: Convergence tolerances obtained with the `ITER_CONTROL` command.

$\bar{u}$ equation is solved with $\bar{v}$ and $\bar{p}$ held constant. Next the $\bar{v}$ equation is solved with constant $\bar{u}$ and $\bar{p}$. Before the $\bar{p} - \bar{u} - \bar{v}$ sequence is repeated the velocity field is corrected so that mass conservation holds for flow into and out of every control volume in the domain. Consult the book by Patankar [5] for more details.

The corrections to the velocity field are obtained by solving an auxiliary equation for a quantity called the pressure correction. Before the pressure correction equation is solved, there are errors in local mass conservation which correspond to local mass sources and sinks. The largest mass source/sink is an indication of how poorly the continuity equation is satisfied. Note that there are no real mass sources or sinks in the velocity field. These source/sink terms are an artifact of the iterative solution, and the size of the sources/sinks are an indication of how far the iterations are from convergence. Normalizing the maximum mass source/sink by the largest cell-based mass flux gives a reliable and relatively problem-independent indication of convergence for the continuity equation.

The absolute value of the maximum, normalized mass source/sink is given the symbol S_{max}. Because S_{max} is normalized the same magnitude of the tolerance parameter, ϵ, from Table 3.3 is also used to determine convergence of the mass corrections. In other words, the local continuity equation is converged when $S_{max} < \epsilon$.

Convergence of the T equations.

The three-dimensional energy equation is a linear partial differential equation. The `3D` code solves the discrete approximation to this PDE obtained with the control-volume finite-difference method. The equations for the nodal temperatures can be put into the form of equation 3.1, and convergence of the iterative solution to this set of equations follows the same principles discussed in the preceding sections. The `3D` code iterates until $\eta_T^k < \epsilon$ where the value of ϵ depends on the argument of the `ITER_CONTROL` command as indicated by Table 3.3.

Convergence of the energy balance.

The control-volume finite-difference method conserves energy. When the three-dimensional energy equation has converged a global energy balance should also hold. This energy balance is calculated as another indication of the convergence of the iterative calculations. The energy balance tolerances used by the PCBCAT are listed in Table 3.3. Note that the tolerance on the energy balance is *not* normalized.

3.5.3 Recommended Strategy for Model Development

As you gain familiarity with the PCBCAT you will probably develop your own technique for developing models. Here is recommended a strategy for model development that will help you build models efficiently and with greater confidence in the final results.

In general do not expect to obtain reliable results on a new problem from just one run of the PCBCAT. A good strategy is to start with a coarse grid and refine it systematically. The PCBCAT is designed to make grid refinement as easy as possible.

Since the execution time increases nonlinearly with the grid size, it makes sense to debug your model with the coarsest grid that is feasible. Regardless of the grid resolution parameters specified in the `CV_SIZE` command, the PCBCAT will use a grid that is sufficient to locate all physical objects *at the exact locations you request.*

Before you make any grid refinements it makes sense to verify that your model not only converges, but that it is consistent with your physical intuition. Here are some questions to ask during this early phase of model development.

- Have the mass corrections (S_{max}) decreased significantly during solution of the depth-averaged flow equations?
- Do the velocities at the first probe location oscillate? If so, do the S_{max} values oscillate? Oscillations of more than one or two percent indicate potential model problems.
- Is an energy balance obtained at the end of the calculation?
- Are the flow velocities consistent with the flow regime (laminar or turbulent) specified with the `FLOW_REGIME` command?
- Are the device temperatures reasonable, given what you know about the board design and the range of flow rates appropriate for this design?

Once these questions are answered to your satisfaction you should begin grid refinement. At this stage all the physical features of the model are fixed, only grid tolerance and solution convergence parameters will be altered in the grid refinement stage.

The goal of grid refinement is to solve the problem on a sequence of finer grids until the computational grid itself no longer has an effect on the results. When this occurs the solution is said to be grid independent. Figure 3.5 is a sample worksheet that can help organize a grid refinement study. You may want to develop your own worksheet using Figure 3.5 as a model.

Ideally you will only need to change the grid tolerance parameters with the `CV_SIZE` command. In practice, you may also need to to loosen the convergence tolerance (cf. `ITER_CONTROL` command) in order to maintain acceptable solution times for the finest grids. The grid refinement worksheet has columns for the three grid tolerances, `xcv`, `ycv`, and `zcv` along with a column for the convergence tolerance. These four values are specified before the simulation is run.

After each run, fill in the remaining columns of the grid worksheet. The grid size parameters, L1, m1, and n1 are determined by the geometric complexity of the problem along with the tolerances specified with the `CV_SIZE` command. As discussed in section 3.5.1, decreasing any one of the grid tolerances by a factor of two will not necessarily result in a corresponding doubling of the number of control volumes in that direction.

Record the iterations necessary to obtain convergence for both the DA and 3D models. Be careful to note whether the calculations were stopped because the convergence tolerances were met, or whether the maximum number of iterations specified with the `ITER_CONTROL` parameter were exceeded. If the upper limit of iterations was reached before convergence you should increase this limit or loosen the convergence tolerance. If the solutions do not converge after many, say 500 to 1000 iterations, then there is probably an error in your model.

The remaining columns of the worksheet are for recording the energy balance, and junction temperatures for devices in the domain. Extra columns are provided for any other variables you

Grid Tolerance			Convergence Tolerance	Grid Size			Iterations		Energy Balance (W)	$T_{j,1}$	$T_{j,2}$			
xcv	ycv	zcv		L1	m1	n1	DA	3D						

Figure 3.5: Sample worksheet used to determine when a grid-independent solution is obtained..

wish to monitor. The energy balance should always converge to a small fraction of the total energy input for the problem. As you refine the grid the number of iterations needed to reach an acceptable energy balance may increase.

The calculations can be said to have reached grid independence when the junction temperatures (along with other monitored variables you have selected) do not change when the grid is significantly refined. You should at least attempt to increase the number of control volumes (L1, m1, and n1) by a factor of 2 or 4 during the grid refinement. This goal may not be practical if your computer has limited memory or is too slow.

Chapter 4

Output from the PCBCAT

This chapter gives information on the output files created by the PCBCAT. All of the numerical results are saved in plain text files or binary files. Plain text files are meant to be read by the user. They contain summary information useful for further engineering analysis, such as calculation of heat transfer coefficients. Binary files contain field data meant either to be read by the 3D energy equation model or by visualization software.

Throughout this chapter excerpts from PCBCAT output file will be examined. These file come from the example problems in the tutorial Chapter. You may want to refer to those example problems for additional information.

4.1 File Formats and File Names

Each run of the PCBCAT creates several files. For these files, the type of file as well as the physical problem it relates to can be deduced from the file extension according to the convention summarized in Table 4.1.

The name of a typical PCBCAT output file looks like

$$\underbrace{\texttt{basename}}_{\texttt{FILE\_NAME}}\underbrace{\texttt{45x35x25}}_{\text{grid}}\underbrace{\texttt{.3Dout}}_{\text{ext.}}$$

When the PCBCAT creates a new file it uses the string supplied by the `FILE_NAME` command as a base. The complete file name is constructed by adding characters to indicate the grid size and the kind of data contained in the file.

The grid size is encoded in a substring of the form "`LxM`" for depth-averaged output files and "`LxMxN`" for three-dimensional output files. The `L`, `M`, and `N` parameters are the size of the grid in the x, y, and z directions, respectively. In the preceding example, `L` = 45, `M` = 35 and `N` = 25. The content of the file is indicated by the file extension according to the convention summarized in Table 4.1.

Consider the effect of including the following lines in the PCBCAT input file

```
FILE_NAME  motherBoard
DOMAIN  0.28  0.18  0.0254
CV_SIZE 0.005  0.005  0.001
```

The text output from the DA model will be stored in the file called `motherBoard56x36.DAout` and the text output of the 3D model will be stored in `motherBoard56x36x26.3Dout`.

Extension	File type and contents
`.3Dout`	text file summary of results from the 3D energy equation model
`.DAout`	text file summary of results from the depth-averaged flow model
`.cntl`	temporary text file read by 3D and DA models
`.dav`	PCBCAT binary file containing the depth-averaged velocity field
`.geom`	text file of geometry data for input to AVS postprocessing
`.hdf`	Hierarchical Data Format (HDF) binary file of field data

Table 4.1: Meaning of file name extensions for files created during a run of the PCBCAT.

The `FILE_NAME` command is optional. If there is no `FILE_NAME` command in the user input file the base name "`pcbcat`" is used for all output files.

While the base name and grid refer to a specific run of the PCBCAT, the file extension indicates the type of data contained in the file. The `.cntl` extension is reserved for the temporary file, `pcbcat.cntl`, that is created by the preprocessor. This file is a translation of the user input file into a form suitable for input to the `DA` and `3D` analysis codes. Under normal operation the `runcat`, `runda`, and `rune` scripts delete the `pcbcat.cntl` file after the analysis codes are finished.

The `DAout` file is a plain text summary of the results from the depth-averaged model. Refer to Section 4.3 for a detailed description of a typical `DAout` file.

The `dav` extension is used for the binary file containing the depth-averaged velocity field. This file is the mechanism by which the depth-averaged velocity field is transferred to the three-dimensional energy equation. You will not be able to read the contents of this file, but you will want to save it until the analysis with the `3D` model is complete. Of course, each time the depth-average model is run a new `dav` file is created. The `3D` code looks for a `dav` file with the appropriate base name and grid dimensions. If one is not found the `3D` code prints an error message and stops.

The `3Dout` file is a plain text summary from the solution to the three-dimensional energy equation. The contents of this file are described in Section 4.4. The `3D` code also creates files with `geom` and `HDF` extensions. The `geom` file is read by the `pick_pcbcat` AVS module as described in Section 4.5.1. If you do not have AVS installed then the `geom` file is of no use. The `HDF` file contains the computational grid and temperature field stored in Hierarchical Data Format (HDF). A variety of visualization postprocessing packages can read HDF files. Refer to Section 4.5 for more information.

4.2 The PCBCAT Preprocessor

The preprocessor converts the commands in the user input file to a format that is easily read by the analysis codes. The preprocessor output is always stored in the file `pcbcat.cntl` (cf. Figure A.2), which is a plain text file. During normal operation (including error handling) the `runcat` script cleans up after itself by deleting the `pcbcat.cntl` file.

If the user input file contains the `VERBOSE` command the preprocessor creates an additional file named `fname.prout`, where "`fname`" is the string specified in the `FILE_NAME` command. The `fname.prout` is used for debugging and it will be of little use to the average PCBCAT user.

4.3 Depth-Averaged Model

The DA model creates the `fnameLxM.dav` and `fnameLxM.DAout` files. The "`L`" and "`M`" in the file name indicate the size of the grid used in the analysis. The `fname.dav` file contains the depth-

```
Depth-Averaged Computation  in  Cartesian  coordinates
------------------------------------------------------

u-v-p coupling enforced with the SIMPLER algorithm

L1,m1                 =       33        23
xLen,yLen,zLen        =   0.3000    0.2000    0.0400
rhof,cpf,conductf,muf =     1.20   1004.00    0.0256    1.82e-05

Initial interior velocity field set to:
    U = 0  V = 0  W = 0
```

Figure 4.1: Example of the list of control parameters in the `block1_33x23.DAout` file.

averaged velocity field stored in binary format for reading by the three-dimensional energy equation model. Reading and writing binary files is more efficient than reading and writing plain text files. You will not be able to read the `fnameLxM.dav` file with a plain text editor.

The `fnameLxM.DAout` file is a text-only summary of the analysis. The first line of `fnameLxM.DAout` lists the name of the file and the time it was created. The rest of `fnameMxN.DAout` is divided into major sections, each of which is indicated by a heading and underlining. The first section, an example of which is shown in Figure 4.1, lists a summary of the parameters controlling execution. The number of nodes in the x and y directions is given by the values of `L1` and `m1`. The values of `L1` and `m1` should be equal to the "L" and "M" in the file name. The physical length the domain in each coordinate direction is indicated by `xLen`, `yLen`, and `zLen`. These dimensions are in meters and should correspond exactly to the arguments of the `DOMAIN` command in the user input file. The fluid properties are the density, `rhof`, (kg/m^3), specific heat at constant pressure, `cpf`, $(J/kg/K)$, the thermal conductivity, `conductf` $(W/M/K)$, and the dynamic viscosity, `muf`, $(Pa \cdot s)$. The property values should equal the values in the material database for the fluid selected by the `COOLANT` command.

The last piece of information in this section indicates the velocity field used as an initial guess for the solution. The initial velocity field is always zero.

The next section of the `fnameLxM.DAout` file summarizes the convergence of the flow field calculations. An excerpt of this section from a sample PCBCAT run is shown in Figure 4.2. Immediately following the heading is an indicator of the monitored probe position. This probe is either the first probe specified by the user in the user-input file, or, if no probes are specified, a point in the geometric center of the domain.

The convergence data is presented in columns of numbers. The first column, labelled `iter` is the iteration number for the outer iterations of the SIMPLER algorithm. Each "`iter`" involves a solution to the $\bar{u}$, $\bar{v}$, and $\bar{p}$ equations, along with solution to the pressure correction equations and subsequent correction of the velocity field so that local continuity is enforced.

The second column of the iteration history is the value of the S_{max} parameter discussed in section 3.5.2. For the results presented in Figure 4.2 the value of S_{max} is reduced by four orders of magnitude in 47 iterations. This indicates convergence of the local continuity equation as well as the pressure correction calculations.

```
Convergence History
-------------------

       Velocities monitored at probe position:
       (xprobe,yprobe) = (0.225000,0.100000)


  iter   smax       Umon       Vmon    normres[U] normres[V] normres[P]

   1   7.66e-01   1.51e+00  -1.91e-02   7.07e-01   2.24e+00   3.02e-02
   2   4.37e-01   1.48e+00  -1.79e-02   7.07e-01   2.24e+00   5.00e-02
   3   1.17e-01   1.48e+00  -1.57e-02   7.07e-01   2.24e+00   1.79e-01

                                 ⋮         ⋮         ⋮

  46   1.15e-04   1.20e+00  -1.35e-02   2.52e-05   4.66e-04   7.26e-07
  47   9.44e-05   1.20e+00  -1.35e-02   2.52e-05   4.66e-04   6.07e-07
```

Figure 4.2: Example of the convergence history in a `.DAout` file

The `Umon` and `Vmon` colunms are the x and y velocity components at the monitored probe location. For a converged solution these values should become constant. The last three columns are the normalized residuals of the $\bar{u}$, $\bar{v}$, and $\bar{p}$ equations. Refer to section 3.5.2 for a definition of the normalized residual. The solution has converged when the normalized residual values have been reduced by several orders of magnitude. The desired degree of residual reduction is specified by the argument of the `ITER_CONTROL` command. Refer to section 3.5.2 and Chapter 5 for additional information.

The last section of the `fnameLxM.DAout` file presents the overal results of the calculations. This includes the total mass flow and Reynolds number of the flow through the domain, as well as information for the user-defined objects in the domain. The Reynolds number is based on the hydraulic diameter of the calculation domain, which for complicated problems may not be that meaningful. The Reynolds number based on hydraulic diameter is

$$Re = \frac{\rho \bar{U} D_h}{\mu}$$

where $\bar{U}$ is the average velocity, D_h is the hydraulic diameter, and ρ and μ are the fluid properties. The value of $\bar{U}$ is easily computed from the flow through the inlet. The hydraulic diameter is

$$D_h = \frac{4A}{P}$$

where A is the cross-sectional area of the duct and P is the wetted perimeter. For a complicated circuit board with a partially obstructed inlet or outlet, the hydraulic diameter may not be the

Label	BC Type
`D`	Dirichlet
`F`	Specified Flux (Neumann)
`S`	Symmetry
`I`	Inlet
`O`	Outlet
`W`	Wall

Table 4.2: One character labels used to indicate the type of boundary condition.

appropriate length scale. The PCBCAT analysis codes deduce the A and P values from heuristic rules about the location of the inlets and outlets. The value of Re is provided as a convenience to the user.

The rest of the `.DAout` file contains information for each of the user-defined objects in the calculation domain. The objects are either probes, surfaces, blocks or devices.

The probe summary lists the position, velocities and temperatures at the location of the probe. The (x, y, z) velocity components are (u, v, w), respectively. For the DA model the w velocity component and the temperature are zero because these values are not computed by the model.

The surface summary is organized by domain face. Only user-defined surfaces are presented. (The PCBCAT defines internal surfaces as necessary to cover those parts of a domain face that are not *explicitly* specified by the user.) For each face, the first column of the summary gives the surface number, starting with zero and ending with $n_s - 1$, where n_s is the number of user-defined surfaces on that face. The second column is the name given to the surface as the first argument of the surface-defining command, cf. `INLET`, `OUTLET`, or `PATCH`. The third column is a single character indicating the type of boundary condition applied at the surface. The relationship between the boundary condition label and the type is summarized in Table 4.2 The fourth column of the surface summary is the average surface temperature. If the surface has a Dirichlet boundary condition the temperature will be that specified by the user. Otherwise it will be the average temperature resulting from the solution to the energy equation. (In the DA model the wall temperature is always zero, because the energy equation is not solved.)

The fifth and sixth columns are the average heat flux and the total heat transfer rate for the surface. If the surface has a Neumann (`FluxBC`) boundary condition, the heat flux and total heat transfer rate will be the values specified by the user. Otherwise these values are calculated from the solution to the energy equation. Note that the overall heat transfer rates for all surfaces on a face will add up to the corresponding value (for that face) appearing in the global energy balance.

The remaining columns give the location and extent of the surface as defined by its origin and length in each coordinate direction. Consider the outlet surface located on the east face in Figure 4.3. This surface has no x-direction extent so its origin (`xOrg`) and length (`xLen`) in the x-direction are both zero. The outlet covers the entire east face which is 20 cm wide and 4 cm high. This corresponds to y and z origins both equal to zero, a y length of 20, and a z length of 4.

The block summary follows a format very similar to that of the surface summary. The material properties of the the blockare listed.

NO DEVICE SUMMARY YET

4.4 Energy Equation Model

The 3D energy equation model creates three files with extensions `3Dout`, `HDF` and `geom`. The `3Dout` file is a text-only summary of the analysis. The `HDF` and `geom` files contain the field and object-based data designed to be imported into visualization programs.

The temperature field calculated by the 3D energy equation model is stored in `fname.HDF` in Hierarchical Data Format (HDF). This format was developed at the University of Illinois and is in the public domain. Many visualization packages can import HDF files. We have written interactive modules for the AVS visualization system that allow the user to import the HDF output from PCBCAT along with the problem description data saved in `fname.geom` (see section 4.5.1 and reference [8]). With this system users can render a three-dimensional model of the circuit board, and using mouse input, interactively query the objects in the domain.

4.5 Viewing the HDF Field Data

The three-dimensional field data is stored in HDF files. With this data you can look at the flow and temperature fields over the devices on the circuit board. There are several commercial and public domain visualization systems that can read the HDF files written by the PCBCAT models. HDF-reading software from three sources, AVS, NCSA and Spyglass, are briefly described below.

4.5.1 AVS

AVS is a software package specifically designed for visualization of scientific data [1]. AVS software is organized into modules which perform specific data manipulation or rendering tasks. By connecting modules together the AVS user achieves the desired representation of his/her data.

We have written an AVS module that superimposes the three-dimensional geometry of a PCBCAT model onto a rendering of the temperature field data [8]. The `pick_pcbcat` AVS module reads the `fnameLxMxN.geom` and `fnameLxMxN.HDF` files. The `geom` file supplies the geometric description of the PCBCAT objects in the domain. The `HDF` file contains the computational grid and the field data used by AVS to render the field. Refer to that paper for more information on using AVS with the PCBCAT.

4.5.2 NCSA Tools

HDF was invented at the National Center for Supercomputing Applications (NCSA) at the University of Illinois. This group has also written public domain visualization programs for a variety of computer platforms. The programs can be downloaded by anonymous ftp from `ftp.uiuc.edu`. The NCSA tools read the HDF files created by the PCBCAT.

4.5.3 Spyglass Tools

Spyglass, Inc. was formed by some of the original team of programmers from NCSA. Spyglass sells commercial version of the NCSA tools for the most popular computer platforms. All of these programs are capable of reading the HDF files created by the PCBCAT.

```
   Flow Field Summary
   ------------------

   Mass flow through the domain       =   1.8167e-03 kg/s
   Volumetric flow through the domain =   1.5080e-03 m^3/s
   Re based on hydraulic diameter     =        833.2



   Summary Results for User-Defined Objects
   ----------------------------------------


   Values of the field variables at each probe
   -------------------------------------------

  i    x (cm)   y (cm)   z (cm)   u (cm/s)  v (cm/s)  w (cm/s)    T (C)
  0    22.50    10.00     2.00     19.35     -0.23      0.00      0.00
  1    15.00    10.00     2.00     21.66      0.05      0.00      0.00
  2     7.50    10.00     2.00     18.81      0.23      0.00      0.00


   User-defined surfaces on the  east face
   ---------------------------------------

 n   label          BC     T        q       Qeast    xOrg  xLen  yOrg  yLen  zOrg  zLen
                          (C)    (W/cm^2)    (W)     (cm)  (cm)  (cm)  (cm)  (cm)  (cm)
 0  Outlet           O    0.00     0.000     0.00     0.0   0.0   0.0  20.0   0.0   4.0

   User-defined surfaces on the  west face
   ---------------------------------------

 n   label          BC     T        q       Qwest    xOrg  xLen  yOrg  yLen  zOrg  zLen
                          (C)    (W/cm^2)    (W)     (cm)  (cm)  (cm)  (cm)  (cm)  (cm)
 0  inlet            I   15.00     0.000     0.00     0.0   0.0   0.0  20.0   0.0   4.0


   Summary data for   1 blocks in the domain
   -----------------------------------------

  Label            Qtot   Tave    rho      cp     conduct  xOrg  xLen  yOrg  yLen  zOrg  zLen
                    (W)    (C)   kg/m^3  J/kg/K    W/m/C   (cm)  (cm)  (cm)  (cm)  (cm)  (cm)
heater              2.0   0.00   2770.0   875.0  1.77e+02 12.50  5.00  5.50  5.00  0.00  1.00


   DA model finished in 8.000000 seconds
```

Figure 4.3: Example of the overall summary and results for user-defined objects in the `.DAout` file

Chapter 5

Command Glossary

This chapter lists the PCBCAT preprocessor commands used to specify printed circuit board models. The commands can be loosely classified as one of the following types

- Solution and Execution Control
- Domain Specification
- Flow Field
- Object Location

The Solution and Execution Control commands are listed in Table 5.1. These commands either control the numerical solution procedure or else specify some aspect of how the PCBCAT codes will run on your particular computer. The Domain Specification commands are listed in Table 5.2. These commands define the physical lengths and length tolerances in the calculation domain. The Flow Field commands are listed in Table 5.3. These commands define the fluid properties and specify how the depth-averaged flow field is to be interpreted in the three-dimensional energy equation model. The Object Specification commands are listed in Table 5.4. These commands are used to add physical objects to the calculation domain.

The remainder of this chapter consists of detailed descriptions of each of the commands. The commands appear in alphabetical order and are described in a standard format. First a statement of the command's purpose is given. Following that is a synopsis of how the command and its arguments are to appear in the input file to the preprocessor. The arguments are described, first by specifying the type of value — string, integer or floating point value — and then with a verbal description of the values appropriate for each argument. Some commands do not have any arguments. Additional information, if appropriate, is provided in a "Notes" section at the end of the command description.

Command	Type	Description
`FILE_NAME`	optional	specify the base name for all output files
`INTERACTIVE`	optional	flag to control prompting during execution
`ITER_CONTROL`	optional	sets the numerical tolerance on the solution
`MATERIAL_DBASE`	optional	sets path to user's material database file
`VERBOSE`	optional	flag to turn on extra output

Table 5.1: Solution and execution control commands.

Command	Type	Description
`CV_SIZE`	optional	set maximum sizes for control volumes
`DISTOL`	optional	set tolerances on minimum meaningful distance
`DOMAIN`	required	sets the physical size of the domain

Table 5.2: Domain specification commands.

Command	Type	Description
`COOLANT`	required	define the fluid properties
`FLOW_FIELD`	optional	specify whether or not to compute the flow field
`FLOW_PROFILE`	optional	specify z-direction variation of velocity profiles
`FLOW_FIELD`	optional	specify whether the flow is laminar or turbulent

Table 5.3: Object specification commands.

Command	Type	Description
`BLOCK`	optional	locate a solid, three-dimensional block in the domain
`BOUND`	optional	specify the boundary condition on an entire face of the domain
`DEVICE`	optional	locate an electronic device in the domain
`INLET`	optional	locate the inlet and specify inlet velocity and temperature
`OUTLET`	optional	locate the outlet
`PATCH`	optional	locate a boundary condition on a surface of the domain
`PROBE`	optional	locate a velocity and temperature probe in the domain

Table 5.4: Domain specification commands.

# Comment Statement

Purpose: Provide a mechanism for documenting user input files. Text appearing in a comment statement is ignored by the preprocessor. Comment statements must begin with the `#` character in the first column. Comment statements and blank lines may appear anywhere in the user input file.

Synopsis: `# any text`

Arguments: none

Notes: Comment statements are not required, but strongly recommended.

BLOCK

Purpose: Define a uniform block of solid material in the domain. The block can be heated with a uniform heat source. Blocks can also be used to specify thermally passive barriers or electronic components with negligible heat dissipation. See also the `DEVICE` keyword for a way to specify a more realistic description of an IC package. Blocks defined with the `BLOCK` keyword are always located on the bottom of the domain. If a PC board is defined, then the blocks are located on top of the PC board, which, by definition, is located on the bottom of the domain. See the `BOARD` keyword for a way to define a PC board.

Synopsis: `BLOCK label material Qin heated xorg xlen yorg ylen height`

Arguments:

`label`	(string) a user-defined label for the block. The label can be up to 16 characters long and must not contain any spaces. Examples of valid labels are “Heater_1”, “PassiveDevice”.
`material`	(string) the name of a material in the material database. The material name is used to look up density, specific heat and thermal conductivity of the block. Edit the material database (default file is material.data) to change property values or define new materials
`Qin`	(float) total rate of heat dissipation (W) in the block. This is not the volumetric heat generation rate. The value of `Qin` is ignored if `heated=0`.
`heated`	(integer) flag used to indicate whether the block is heated or not. This allows the user to “turn off” a heater block while keeping the nominal value of the heat dissipation rate stored as a constant in the data file.
`xorg`	(float) x-direction origin of the block
`xlen`	(float) x-direction length of the block
`yorg`	(float) y-direction origin of the block
`ylen`	(float) y-direction length of the block
`height`	(float) z-direction length of the block. The z-direction origin of the block is either the bottom surface of the domain, or the top surface of the PC board if the BOARD keyword is used.

BOUND

Purpose: Prescribe a boundary condition on an entire face in the domain. See `PATCH` for specification of a boundary condition over a limited surface on the boundary

Synopsis:

```
    BOUND  face  thermBC  Tin  flowBC
```

or

```
    BOUND  face  thermBC  Qin  flowBC
```

Arguments:

`face`	(string) specifies one of the primary six faces to which the boundary condition patch is to be applied. Valid values of face are “west”, “east”, “south” “north”, “down”, and “up”.
`thermBC`	(string) that defines the type of thermal boundary condition being imposed. It must be one of the following: “Dirichlet”, “FluxBC”, or “Symmetry”
`thermBC`	(string) that defines the type of thermal boundary condition being imposed. It must be one of the following: “Dirichlet”, “FluxBC”, or “Symmetry”
`Tin`	(float) temperature of the boundary if `thermBC = DIRICHLET`
`Qin`	(float) total heat transfer from the boundary if `thermBC = FLUXBC`. The value of `Qin` is ignored if `thermBC = SYMMETRY` because a symmetry boundary has zero heat flux by definition.
`flowBC`	(string) that defines the type of flow boundary condition being imposed. It must be one of the following: “wall”, or “symmetry”.

Notes: This is an optional key word. The default boundary conditions for all surfaces are no-flow, adiabatic (`FLUXBC` with `Qin = 0.0`) boundaries.

COOLANT

Purpose: To prescribe the coolant flowing through the domain. In a convection problem the interpretation of the coolant is straightforward. The `COOLANT` keyword is also used in pure conduction analysis to specify the material comprising the bulk of domain. Consider a heat conduction problem in which a single solid material is subjected to various thermal boundary conditions. In this problem the "coolant" is the solid material, i.e., it is the material that occupies the bulk of the domain. Input Specification

Synopsis: `COOLANT material`

Arguments:

`material` (string) the name of a material in the material database.

CV_SIZE

Purpose: Define limits on the control volume sizes. This is a very important keyword because it directly affects the accuracy and execution time of the model. The sizes in the arguments are the maximum allowable sizes in the respective coordinate directions. The mesh is made finer when any of the sizes are reduced. Finer meshes will, in general, be more accurate. Finer meshes will also take much longer to achieve a solution.

Synopsis: `CV_SIZE xcv_size ycv_size zcv_size`

Arguments:

`xcv_size` (float) maximum allowable size of the control volume in the x-direction

`ycv_size` (float) maximum allowable size of the control volume in the y-direction

`zcv_size` (float) maximum allowable size of the control volume in the z-direction

Notes: This specification can not be omitted. `xcv_size`, `ycv_size` and `zcv_size` must be large than zero.

DEVICE

Purpose: Add a predefined electronic device model the domain. The geometric features, material properties and nominal power dissipation of the device are defined in the device database.

Synopsis: `DEVICE name orientation xorg yorg powerLevel`

Arguments:

`name`	(string) the name of the device to be placed in the domain. The name *must* match the name of one of the devices in the device database.
`orientation`	(string) orientation direction of the device. The `orientation` argument sets the alignment of the internal directions of a device with the coordinate directions in the calculation domain. Valid `alignment` values are “east”, “west”, “north”, and “south”. See the reference manual for additional information.
`xorg`	(float) x-direction origin of the block
`yorg`	(float) y-direction origin of the block
`powerLevel`	(float) the power level of the device, expressed as a percent of the nominal power defined in the device database. `powerLevel` $= 0.5$ for 50 percent power, 1.0 for 100 percent power, and 1.25 for 125 percent power, etc.

DEVICE_DBASE

Purpose: Specify the path to a user-defined device database. This command allows an alternative device database to be used in defining the physical characteristics of electronic devices. The device database is only read by the preprocessor. The default database is `device.data` and is located in the `pcbcat/dbase` directory. Refer to Chapter A for additional information.

Synopsis: `DEVICE_DBASE  pathname`

Arguments:

`pathname` (string) the path name to the database file.

Suppose that you wish to use your own device database, which is contained in a file named "`myDevices.dat`". (Refer to Chapter 7 for a discussion on how to create a custom device database.) You must not only tell the PCBCAT the name of this file, but also its location in your directory structure. This information is contained in the full path to the file. Suppose that the `myDevices.dat` file is located in the directory `/usr/home/joe/pcb`. The PCBCAT command for setting the path to this device database would be

```
DEVICE_DBASE  /usr/home/joe/pcb/myDevices.dat
```

Notes: This keyword is optional. If no `DEVICE_DBASE` keyword is used, device definitions will be read from the default database "device.data". If you request devices that are not in the device database the PCBCAT preprocessor will print an error message and stop. You will then need to either edit the default device database or provide your own device database. The PCBCAT uses only one device database, either the default or the database specified with the `DEVICE_DBASE` command.

DISTOL

Purpose: Set the size of distance tolerances used in aligning objects within the domain. Distance tolerances are needed for two reasons. First of all the calculations in the model are susceptible to round-off. The preprocessor determines the location of objects in the domain by aligning the edges of the objects to the "object grid". The alignment is always performed to within the current values of the distance tolerances. If an exact match were required the comparison might fail due to the finite precision of the floating point numbers used in the calculations.

The second reason for specifying distance tolerances is in the interest of computational efficiency. If the edges of two objects in the domain are nearly aligned, say to within a specified distance, it is reasonable to assume that they are exactly aligned. If the edges of two objects are very close, but are greater than the distance tolerance, then an extra plane of control volumes will be inserted into the computational domain. Though these extra control volumes may not effect the accuracy of the calculation, they will certainly add to the solution time. Thus, in those situations where devices are almost exactly aligned, the user may reduce the solution time by increasing the distance tolerance. Note that there are distance tolerances in each coordinate direction, and these tolerances are specified independently.

Synopsis:

```
DISTOL  distolx  distoly  distolz
```

Arguments:

`distolx` (float) distance tolerance in the x-direction

`distoly` (float) distance tolerance in the y-direction

`distolz` (float) distance tolerance in the z-direction

Notes: This keyword is optional. The default distance tolerances are distolx = distoly = distolz = 5×10^{-7} m.

DOMAIN

Purpose: Define the physical size of the computational domain. All objects (blocks, devices, inlets, patches, etc.) must fit within the domain.

Synopsis: `DOMAIN  xlen  ylen  zlen`

Arguments:

`xlen` (float) x-direction length of the domain

`ylen` (float) y-direction length of the domain

`zlen` (float) z-direction length of the domain

FILE_NAME

Purpose: Provide a unique base name to be used in constructing output file names. Suppose that you are analyzing a mother board. You could specify “FILE_NAME motherBoard”. The output from the depth-averaged model would then be in a file called motherBoard.DAout, and the output from the energy equation model would be in a file called “motherBoard.3Dout”

Synopsis: `FILE_NAME filename`

Arguments:

`filename` (string) the base name of the output file.

Notes: This keyword can be omitted. If no `FILE_NAME` keyword is used, a default name “pcbcat” will be used as output file name.

FLOW_FIELD

Purpose: Specify how the x-y variation of the velocity field. The velocity field must be defined before the energy equation will be solved. In most cases users will want to compute the flow field with the depth-averaged model. In some cases, however, either no flow or a uniform flow is desired. For example a uniform flow might apply to a regular array of components mounted in a card cage.

Synopsis: `FLOW_FIELD  type`

Arguments:

`type` (string) one of the following "No_flow", "Uniform" or "Depth_ave". If `type` = "No_flow", the velocity field is zero everywhere and a pure conduction analysis is performed. If `type` = "Uniform" the velocity field is taken to be a constant and uniform value throughout the flow gap. In other words if If `type` = "Uniform" there is no x-y variation in the the velocity field. The z-direction variation is specified with the `FLOW_PROFILE` keyword. If `type` = "Depth_ave" the velocity field is computed by the depth-averaged model. The default flow type is "Depth_ave".

FLOW_PROFILE

Purpose: Specify the velocity variation across the gap. This keyword applies only if `FLOW_FIELD` = "uniform" or `FLOW_FIELD` = "depth_ave"

Synopsis: `FLOW_PROFILE  type`

Arguments:

`type` (string) one of the following "Uniform" or "Fully_dev". If `type` = Uniform, the velocity profile is a constant, i.e., plug flow. If `type` = "Fully_dev" the shape of the velocity profile is defined by the profile for fully-developed laminar or turbulent flow between parallel plates. The user selects the flow regime (laminar or turbulent) with the `FLOW_REGIME` keyword. The default flow type is "Fully_dev".

FLOW_REGIME

Purpose: Selects whether laminar or turbulent velocity profiles and effective viscosity distributions are used. This keyword affects the shape of the velocity profile specified by the `FLOW_PROFILE` keyword.

Synopsis: `FLOW_REGIME  type`

Arguments:

`type` (string) either “Laminar” or “Turbulent”. The default flow type is “Laminar”.

INLET

Purpose: Define the location of the inlet and fluid properties there.

Synopsis: If face = “east’ or face = “west”:

```
INLET face yorg ylen zorg zlen Tin Uin Vin Win
```

or if face = “east’ or face = “west”:

```
INLET face xorg xlen zorg zlen Tin Uin Vin Win
```

Arguments:

`face`	(string) specifies the face on which the inlet is located. Valid values of face are “west”, “east”, “south”, and “north”. The value of face determines the meaning of the next four input values, which specify the origin and length of the plane that defines the inlet surface.
`xorg`	(float) x-direction origin of the inlet
`xlen`	(float) x-direction length of the inlet
`yorg`	(float) y-direction origin of the inlet
`ylen`	(float) y-direction length of the inlet
`zorg`	(float) z-direction origin of the inlet
`zlen`	(float) z-direction length of the inlet
`Tin`	(float) temperature (℃) of the coolant at the inlet
`Uin`	(float) x-direction mean velocity of the inlet
`Vin`	(float) y-direction mean velocity of the inlet
`Win`	(float) z-direction mean velocity of the inlet

Notes: Inlets cannot be located on the up or down boundary. The preprocessor uses the absolute value of the normal velocity component to specify the inlet velocity. In other words the flow is guaranteed to be into the domain. The user must take care, however, to specify the correct signs of the tangential velocity components.

INTERACTIVE

Purpose: Allow the user to choose between running the program in interactive mode or running in batch mode. This is primarily useful for debugging the program.

Synopsis: `INTERACTIVE`

Arguments: none

Notes: This is an optional keyword. If the `INTERACTIVE` keyword is present the program runs in interactive mode, and the user is prompted for some inputs during execution. In batch mode the values of all control values are either default values or values specified in the user input file. This feature is used primarily for debugging and may be eliminated in future releases.

ITER_CONTROL

Purpose: Specify the tolerance used to determine whether the models have converged.

Synopsis: `ITER_CONTROL convergeLevel maxit`

Arguments:

`convergeLevel`	(string) qualitative description of the convergence criteria. Three levels of convergence can be chosen: “loose”, “medium” or “tight”. The tighter the convergence criteria the more iterations will be needed in general. Refer to Section 3.5.2 and Table 3.3 in Chapter 3 for more information.
`maxit`	(integer) maximum number of iteration. This is an upper limit, set by the user, to prevent the program from running indefinitely.

MATERIAL_DBASE

Purpose: Specify the path to material database. This command allows an alternative material database to be used in defining all material properties used in the analysis. The material database is only read by the preprocessor. The default database is `material.data` and is located in the `pcbcat/dbase` directory. Refer to Chapter A for additional information.

Synopsis: `MATERIAL_DBASE  pathname`

Arguments:

`pathname` (string) the path name to the database file.

Suppose that you wish to use your own material database, which is contained in a file named "`myMaterial.dat`". (Refer to Chapter 8 for a discussion on how to create a custom material database.) You must not only tell the PCBCAT the name of this file, but also its location in your directory structure. This information is contained in the full path to the file. Suppose that the `myMaterial.dat` file is located in the directory `/usr/home/joe/pcb`. The PCBCAT command for setting the path to this material database would be

```
MATERIAL_DBASE  /usr/home/joe/pcb/myMaterial.dat
```

Notes: This keyword is optional. If no `MATERIAL_DBASE` keyword is used, material properties will be read from the default database "material.data". If you specify materials that are not in the material database the PCBCAT preprocessor will print an error message and stop. You will then need to either edit the default material database or provide your own material database. The PCBCAT uses only one material database, either the default or the database specified with the `MATERIAL_DBASE` command.

OUTLET

Purpose: Define the location of the outlet.

Synopsis: if face = “west” or face = “east”:

```
OUTLET   face  yorg  ylen  zorg  zlen
```

or if face = “north” or face = “south”:

```
OUTLET   face  xorg  xlen  zorg  zlen
```

Arguments:

`face` (string) specifies the face on which the outlet is located. Valid values of face are west, east, south, north, down, and up. The value of face determines the meaning of the next four input values, which specify the origin and length of the outlet.

`xorg` (float) x-direction origin of the outlet

`xlen` (float) x-direction length of the outlet

`yorg` (float) y-direction origin of the outlet

`ylen` (float) y-direction length of the outlet

`zorg` (float) z-direction origin of the outlet

`zlen` (float) z-direction length of the outlet

Notes: Outlets cannot be located on the up or down boundary.

PATCH

Purpose: Specify a uniform heat flux patch over a limited area on the domain boundary. This is useful for simulating experimental data sets involving idealized electronic components.

Synopsis: if face = up or face = down:

```
PATCH  label face xorg xlen yorg ylen thermBC (T or Qin) heated
```

or if face = west or face = east:

```
PATCH  label face yorg ylen zorg zlen thermBC (T or Qin) heated
```

or if face = north or face = south:

```
PATCH  label face xorg xlen zorg zlen thermBC (T or Qin) heated
```

Arguments:

`label`	(string) a user-defined label for the patch. The label can be up to 16 characters long and must not contain any spaces. Example of valid labels are “Heater1”, “Heater_1”, “base_patch”.
`face`	(string) specifies the face on which the heated patch is located. Valid values of face are “west”, east, “south”, “north”, “down”, and “up”. The value of face determines the meaning of the next four input values, which specify the origin and length of the patch.
`xorg`	(float) x-direction origin of the patch
`xlen`	(float) x-direction length of the patch
`yorg`	(float) y-direction origin of the patch
`ylen`	(float) y-direction length of the patch
`zorg`	(float) z-direction origin of the patch
`zlen`	(float) z-direction length of the patch
`thermBC`	(string) that defines the type of thermal boundary condition being imposed. It must be one of the following: “Dirichlet”, “FluxBC”, or “Symmetry”
`Tin`	(float) temperature of the boundary if `thermBC = DIRICHLET`
`Qin`	(float) total heat transfer from the boundary if `thermBC = FLUXBC`. The value of `Qin` is ignored if `thermBC = SYMMETRY` because a symmetry boundary has zero heat flux by definition.
`flowBC`	(string) that defines the type of flow boundary condition being imposed. It must be one of the following: “wall”, or “symmetry”.
`heated`	(integer) use “1” if the patch is heated, “0” if the patch is unheated. This flag allows the patch to be turned on and off without adjusting the total heat transfer rate (`Qin`).

Notes: Only one argument, `T` or `Qin`, is specified after the `thermBC` argument.

PROBE

Purpose: Specify a point at which the velocity and temperature is to be reported. Think of a probe as a instrument capable of simultaneously measuring all three velocity components and the temperature. You may locate a probe anywhere in the domain, including within solid objects.

Synopsis: `PROBE   scale_type  xprobe  yprobe  zprobe`

Arguments:

`scale_type`	(string) specifies the interpretation of the next three position variables. If `position_type`="absolute" the position variables are the absolute coordinates of the probe. If `position_type`="relative" the position variables are interpreted as fractions of the domain dimension in each coordinate direction. Thus `Probe relative 0.5 0.5 0.5` puts the probe in the geometric center of the three-dimensional domain, regardless of the actual physical size of the domain.
`xprobe`	(float) probe position in x-direction
`yprobe`	(float) probe position in y-direction
`zprobe`	(float) probe position in z-direction

VERBOSE

Purpose: Allow the user some primitive control on the amount of output created by the preprocessor and the analysis codes. This keyword turns on an internal flag, which causes messages on the progress of the calculations to be printed to the screen (or standard output).

Synopsis: `VERBOSE`

Arguments: none

Notes: This is an optional keyword that may be useful in debugging.

Chapter 6

Example Models

This Chapter describes a set of additional example problems distributed with the PCBCAT. These examples involve simulations of experiments performed by other researchers.

Chapter 7

Device Database

Electronic devices are added to a PCBCAT model with the `DEVICE` command, which specifies the location and orientation of a device. The dimensions, material properties, and nominal power dissipation rate of devices are all defined in the device database. This allows for a logical break between the definition of a device and its use in a particular PCBCAT model. Devices are defined only once in the device database. A given device can then be added to a PCBCAT model with a minimum of effort. This also helps to reduce the chance of data entry error.

One of the parameters of the `DEVICE` command is the power level at which the device is operating on a particular board. The nominal power level is defined in the database and the actual operating power level is controlled by the scaling factor in the `DEVICE` command.

This chapter describes the creation and management of entries in the device database.

Entries in the device database are a combination of strings and numbers. The strings are used to label the device (i.e., give it a name) and to specify the materials from which the device is made. The numbers in the database specify the physical dimensions of the device its nominal power.

```
label   power
        packageMaterial   xLenp   yLenp   zLenp
        dieMaterial       xOrgd   xLend   yOrgd   yLend   zOrgD   zLend
        baseMaterial      zLenb
```

Chapter 8

Material Database

Material properties for fluids and objects used in the analysis are defined by property values stored in a material database. All materials used in the analysis must have their thermophysical properties defined in the database. The database is a plain text file that can be modified with any text processor.

Each PCBCAT user can build and maintain a customized material database. At run time the database for a particular model is specified by the `MATERIAL_DBASE` command. Alternatively a user can modify the default database, "`material.data`", which is located in the `pcbcat/dbase` directory (see Appendix A).

The database is a plain text file with the structure depicted in Figure 8.1. The default material database provided with the PCBCAT distribution is shown in Figure 8.2. The thermophysical properties of a particular material appear on one line of the file. The first entry is the name of the material. The name may be any continuous character string of up to 31 characters. Upper and lower case alphabetic characters and numbers are allowed, but the material names are *not* case sensitive. Spaces are not permitted in material names, and the special characters "/" and "\" should not be used. Thus "air", "HighTempEpoxy",and "Si-C" are valid names, whereas "Silicon die" and "Sn/Pb" are not valid names.

Material properties are assigned by matching the material name specified in the user input file (see, e.g., the `COOLANT` and `BLOCK` keywords) with material names in the first column of the material database. The match is made without considering the case of the characters in the material name. It is essential that the user spells the material the same way in both the material database and in the user input file. If the preprocessor cannot match a material name in the user input file with a material name in the database, the program prints an error message and quits.

Following the material name are the density (kg/m^3), specific heat ($J/kg/m^3$), thermal conductivity ($W/m/K$) and dynamic viscosity (Pa/s) or ($kg/m/s$). These data are separated by spaces or tabs, no commas. The material properties are in SI units only. For fluids, be sure to specify c_p, the specific heat at constant pressure, not c_v, the specific heat at constant volume. For solids the viscosity should be a large value such as 1.0×10^{15}. The absolute magnitude of the solid viscosity is arbitrary so long as it is several orders of magnitude larger than any of fluid viscosity values.

name$_1$	ρ_1	$c_{p,1}$	k_1	μ_1
name$_2$	ρ_2	$c_{p,2}$	k_2	μ_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 8.1: Structure of material database file

water	998.2	4182.0	0.6	1.003e-3
air	1.2047	1004.0	25.63e-3	1.817e-5
aluminum	2770.0	875.0	177.0	1.0e15
steel	7854.0	434.0	60.5	1.0e15
copper	8933.0	385.0	401.0	1.0e15
epoxy	1200	1750.0	3.0	1.0e15
SiliconDioxide	2220	745.0	1.38	1.0e15

Figure 8.2: Default material database file supplied with the PCBCAT distribution.

Bibliography

[1] AVS Inc. *AVS User's Guide.* Advanced Visual Systems, Inc., Waltham, MA, 1992.

[2] S. Garimella and P. Eibeck. Fluid dynamic characteristics of the flow over an array of large roughness elements. *Journal of Electronic Packaging*, 113(4):367–373, 1991.

[3] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, MD, second edition, 1989.

[4] S. Kim and N. Anand. Laminar heat transfer between a series of parallel plates with surface-mounted discrete heat sources. *Journal of Electronic Packaging*, 117:52–62, Mar 1995.

[5] S. Patankar. *Numerical Heat Transfer and Fluid Flow.* Hemisphere, Washington D.C., 1980.

[6] G. W. Recktenwald. Prediction of device temperatures with depth-averaged models of the flow field over printed circuit boards. In *Proceedings of 1995 International Mechanical Engineering Congress and Exposition*, San Francisco, CA, 1995. ASME.

[7] G. W. Recktenwald. Using the pcbcat to model convective heat transfer from electronic devices on printed circuit boards. In *Proceedings of 1995 International Mechanical Engineering Congress and Exposition*, San Francisco, CA, 1995. ASME.

[8] G. W. Recktenwald and P. A. Gotseff. A visualization tool for CFD models of convectively cooled printed circuit boards. In *Proceedings of 1995 ASME National Heat Transfer Conference*, Portland, OR, 1995. ASME.

Appendix A

Installing and Running the PCBCAT on a SUN Workstation

This appendix gives instructions for installing the PCBCAT tools on a SUN workstation. The instructions depend somewhat on how you obtained the PCBCAT distribution archive.

The PCBCAT are available via anonymous ftp from `ftp.ee.pdx.edu`. Refer to the cover pages of this manual for instructions on downloading the files via the internet. The PCBCAT may also be obtained on floppy disk or tape for a nominal fee by writing to the author. Getting the PCBCAT from the internet is strongly recommended since this will result in the quickest turn around. You will also be able to get future upgrades to the codes and documentation this way.

Once you have a copy of the installation archive on your computer you must perform the following steps before you can run the PCBCAT

1. unpack the archive
2. add the `pcbcat/bin` directory to your default search path
3. define the `PCBCATDIR` environment variable

These steps are described in more detail below. In addition it is a good idea to run the test problems provided with the PCBCAT distribution. This will give you some experience with the tools and, more importantly, it will assure you that the tools have been installed successfully. Refer to section A.6 for a quick test of the installation. Chapters 2 and 6 provide additional example problems.

A.1 Installing the PCBCAT Archive Obtained by Anonymous ftp

The instructions in this section assume that you have a copy of the PCBCAT installation archive on your local workstation. This will be the case if you have downloaded the archive from the anonymous ftp server at Portland State University.

Installation of the tool set involves uncompressing the archive and unpacking a `tar` archive. First create a working directory for the tools. It is a good idea to call this directory `pcbcat` or some other mnemonic name. If more than one user will be running the PCBCAT you should install the tools in a central directory, e.g., `/usr/local/pcbcat`. Doing so will probably require the intervention of your system administrator. If you install the PCBCAT for your private use then any subdirectory of your home directory will work. In the remainder of this chapter we will simply refer to the installation directory as `pcbcat`.

Copy the `pcbcat.tar.Z` file to the `pcbcat` directory. Change your current working directory to `pcbcat` (i.e., `cd pcbcat`) and type

```
uncompress pcbcat.tar.Z
tar -xvf pcbcat.tar
```

In the preceding command the distinction between upper and lower case characters is important. Executing the "`tar -xvf pcbcat.tar`" command will extract all the files and directories for the tools. This should take about one minute. Assuming that the installation was successful, skip to section A.3

A.2 Installing the Sun bar Archive on Diskette

This section gives instructions for installing the PCBCAT tools from a 3.5 inch diskette.

Installation of the tool set involves unpacking a SUN `bar` archive. First create a working directory for the tools. It is a good idea to call this directory `pcbcat` or some other mnemonic name. If more than one user will be running the PCBCAT you should install the tools in a central directory, e.g., `/usr/local/pcbcat`. Doing so will probably require the intervention of your system administrator. If you install the PCBCAT for your private use then any subdirectory of your home directory will work. In the remainder of this chapter we will simply refer to the installation directory as `pcbcat`.

Insert the 3.5 inch diskette into the drive located on the right side of your workstation. Change your current working directory to `pcbcat` (i.e., `cd pcbcat`) and type

```
bar xvfp /dev/rfd0 PCBCAT
```

In the preceding command the distinction between upper and lower case characters is important. The last character of "`rfd0`" is a zero, not capital "O". Executing the "`bar xvfp /dev/rfd0 PCBCAT`" command will extract all the files and directories for the tools. This should take about one minute.

A.3 Directory Structure for the PCBCAT

Figure A.1 is a graphical representation of the directory structure that results from the installation procedures described above. The `bin` directory contains shell scripts and the executables for the depth-averaged and three dimensional energy equation models. These files are briefly described in Table A.1. The `dbase` directory contains the default material and device databases. The `tutorial` directory contains files used in exercises in Chapter 2, *Tutorial*. The `example` directory contains additional sample input files that are described in Chapter 6, *Examples*.

The shell scripts in the `bin` directory make some important assumptions about the file organization. The absolute path to the `pcbcat` directory needs to be defined in an environment variable (see section A.5, below). The default material and device databases are assumed to be in the `dbase` directory. User-defined material and device databases can be used by including the `MATERIAL_DBASE` and `DEVICE_DBASE` commands in the input file to the PCBCAT. Refer to the command definitions in Chapter 5 for more information.

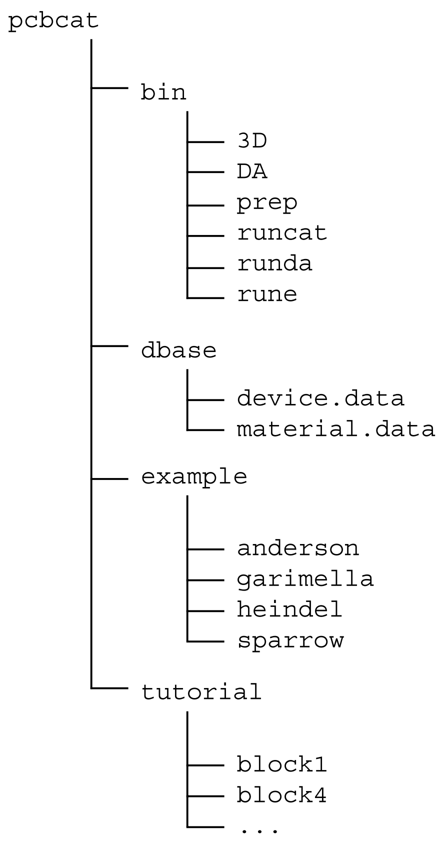

Figure A.1: Directory structure after installation of PCBCAT.

file name	Description
`3D`	executable for the 3D energy equation model
`DA`	executable for the depth-averaged flow model
`prep`	executable for PCBCAT preprocessor
`runcat`	script to execute `prep`, `DA` and `3D` in sequence
`runda`	script to execute `prep` and `DA` in sequence
`rune`	script to execute `prep` and `3D` in sequence

Table A.1: Contents of the `bin` directory.

A.4 Setting the Path

The `bin` directory contains the executable scripts and programs that constitute the PCBCAT. You will have to include a path to the `bin` directory in order for your computer to recognize the scripts as valid commands. The most convenient way to do this is to edit your `.login` or `.cshrc` (whichever applies in your system configuration) and add the path to the `pcbcat/bin` directory in your default path.

Suppose that the PCBCAT were installed in the `/usr/local/pcbcat` directory of your computer. If you run the C shell, insert the following line in your `.cshrc` file to add `/usr/local/pcbcat/bin` to the search path the *next* time you log in.

```
setenv  PATH ${PATH}:/usr/local/pcbcat/bin
```

A.5 Defining the PCBCATDIR Environment Variable

Including the `pcbcat/bin` directory in your path allows you to invoke the PCBCAT shell scripts from any directory. The `prep` program still needs to know the absolute path to the `dbase` directory. This path is constructed from the `PCBCATDIR` environment variable, which you must define. The `PCBCATDIR` variable must be set to the full path of the `pcbcat` directory. If you are running the C shell this environment variable is defined by including the following lines in your `.cshrc` file

```
setenv  PCBCATDIR  pcbcat_path
```

where `pcbcat_path` is the *absolute* path to the `pcbcat` directory. For example, suppose that the PCBCAT have been successfully installed in the `/usr/local/pcbcat` directory. Then the `PCBCAT` environment variable is set with

```
setenv  PCBCATDIR  /usr/local/pcbcat
```

If you add the `setenv ...` statement to your `.cshrc` file you must log in again (or spawn another shell) before the assignment takes effect. You can verify that the PCBCATDIR variable is set correctly by typing

```
ls  $PCBCATDIR/dbase
```

The correct result will be a listing of the contents of the `dbase` directory, namely

```
device.data    material.data
```

A.6 Verifying the Installation

You can test the installation by running any of the example problems in the `tutorial` or `example` directories. First, verify the contents of the `tutorial` directory by typing

```
ls  $PCBCATDIR/tutorial
```

Your computer should respond with

```
block1  block4
```

Run the `block1` tutorial problem by typing

```
runcat  $PCBCATDIR/tutorial/block1
```

The output should resemble that in Figure 2.4 in Chapter 2

A.7 Running the PCBCAT

The PCBCAT tools are a collection of programs that work together. Although these programs function as a integrated analysis package it is helpful to know how the data flows between them. The information flow for a complete analysis is depicted in Figure A.2.

The `runcat` and `rune` shell scripts are the commands you invoke to run the preprocessor (`prep`) and the analysis codes (`DA` and `3D`). A model of a particular board is completely described by a text file that we refer to as a "user-input file". Both `runcat` and `rune` invoke the preprocessor, and both have only one input argument, viz. the user-input file containing preprocessor commands. These scripts have the following syntax

```
runcat  in_file_path

runda  in_file_path

rune  in_file_path
```

where `in_file_path` is the path to the user input file. Since the input file is specified by its path it does not have to be in the current working directory. In contrast all output files from the PCBCAT are created in the current working directory.

For the sake of concreteness suppose that you have created a PCBCAT model contained in the file named "`user.input`" in the current working directory[1]. If the file `user.input` in in the current working directory the most straightforward way to run the PCBCAT model is to type

```
runcat  user.input
```

This command initiates the operations depicted in Figure A.2

The preprocessor parses the PCBCAT commands contained in "`user.input`" and creates a plain text file, `pcbcat.cntl`. If `prep` terminates normally the `runcat` script calls the depth-averaged model, `DA`.

The `DA` code reads the `pcbcat.cntl` file and computes the depth-averaged flow field. The `DA` code creates two output files, `fname.DAout` and `fname.dav`, where `fname` is a string constructed from the string argument of the `FILE_NAME` command (see section 4.1). The `fname.DAout` file is

[1]Remember that the user-input file can have any name so long as it is a plain text file containing valid preprocessor comments or commands. As a matter of style this name should indicate the contents, as is the case for the files in the `pcbcat/tutorial` and `pcbcat/example` directories.

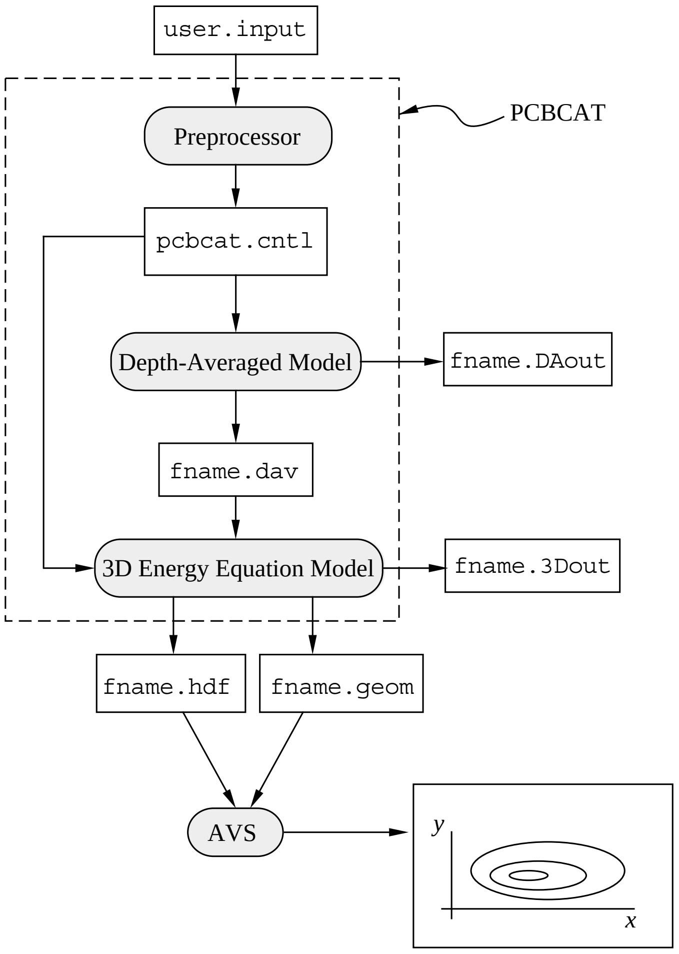

Figure A.2: Schematic of the flow of data between the programs. Executable programs are depicted as shaded oval boxes. Data files input to or output from the programs are shown as rectangular boxes. The box in the lower right corner represents the graphical output from AVS, in this case a contour plot.

a text file summarizing the results of the depth-averaged computations. The `fname.dav` file is a binary file containing the depth averaged velocity field.

If `DA` runs without a fatal error then the `runcat` script calls the three-dimensional energy equation model, `3D`. The `3D` code reads the `pcbcat.cntl` file and the `fname.dav` file, and performs the thermal analysis of the board. The `3D` code creates the `fname.3Dout`, `fname.HDF`, and `fname.geom` files. Refer to Chapter 4 for a detailed description of the output files. When the `3D` code stops, the PCBCAT is finished analyzing the given problem, and the `runcat` script deletes the `pcbcat.cntl` file. Figure A.2 depicts postprocessing with the commercial system AVS, for which we have developed additional visualization tools.

Another way to think about the flow of information between the codes is to consider the operations performed by the `runcat` script. These steps closely follow the graphical representation in Figure A.2. Again, assume that user-input file is named `user.input`. The `runcat` script performs the following steps

1. Pass the user.input file to the preprocessor.
2. If there was an error in the preprocessor, delete the `pcbcat.cntl` file and stop. Otherwise continue.
3. Run the `DA` model.
4. If there was an error in the DA model, delete the `pcbcat.cntl` file and stop. Otherwise continue.
5. Run the `3D` model.
6. Delete the `pcbcat.cntl` file and stop.

The `runda` script runs the preprocessor and solves the depth-averaged flow equations only. The output of the `DA` code is the same as when it is called by the `runcat` script. The `runda` script performs the following steps

1. Pass the user.input file to the preprocessor.
2. If there was an error in the preprocessor, delete the `pcbcat.cntl` file and stop. Otherwise continue.
3. Run the `DA` model.
4. Delete the `pcbcat.cntl` file and stop.

The `rune` script runs the preprocessor and solves the three-dimensional energy equation only. This script is useful when you wish to compare several board power levels or change the only thermal boundary conditions for the same flow field. Since the flow field is independent of the temperature field (constant properties and no buoyancy effects are assumed) the flow field does not need to be recomputed unless the hydrodynamic boundary conditions change. This analysis assumes that the depth-averaged velocity field stored in `fname.dav` is appropriate. Any change to the `user.input` file that changes the flow field requires that you recompute the flow field with the `runcat` script.

The `rune` script performs the following steps

1. Pass the user.input file to the preprocessor.
2. If there was an error in the preprocessor, delete the `pcbcat.cntl` file and stop. Otherwise continue.

3. Run the `3D` model.
4. Delete the `pcbcat.cntl` file and stop.

A.8 A Technical Note for Script Editors

The `prep`, `DA`, and `3D` codes return 0 (zero) if execution terminates normally. If an error is encountered in any of these codes a value of -1 is returned to the shell. The `runcat`, `runda`, and `rune` scripts simply test for a non-zero return value as an indication of an error condition.