A Visualization Tool for CFD Models of Convectively Cooled Printed Circuit Boards[*]

Gerald W. Recktenwald
Peter A. Gotseff
Department of Mechanical Engineering
Portland State University
Portland, Oregon

March 1995

Abstract

An interactive tool for visualizing flow and temperature fields over models of printed circuit boards (PCBs) has been developed using AVS (Application Visualization System from AVS, Inc.). The visualization tool serves as a post-processor to control-volume finite-difference CFD codes used to simulate the convective heat transfer from the electronic devices on the PCB. Using the AVS application development environment we created a custom module to render solid objects and other physical features of the computational domain, in addition to the three dimensional field data. The custom module also provides information on the objects when the user selects them with the mouse. The end result is an interactive post-processing application that allows the user to query the objects embedded in the solution domain. In this paper we provide overviews of AVS, the PCBCAT analysis codes, and how the custom module allows interactive display of PCBCAT results. We also present a sample of output from the visualization system.

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 (Recktenwald, 1995a, 1995b). The control volume finite difference technique is used to create a model of the the fluid flow, the heat conduction in the components attached to the board, and the thermal convection in the fluid above the board. The PCBCAT are based on a novel computational fluid dynamics (CFD) model involving two dimensional solution of the flow field coupled with a three dimensional model of heat transport by conduction and convection. This approach eliminates the need for specification of heat transfer coefficients as is necessary in conventional conduction-based analysis codes used in electronic cooling design work. It also is significantly faster and easier to use than a fully three dimensional CFD model.

The visualization work began as a debugging tool during development of the analysis codes. It has since evolved into an interactive post-processing tool for end-users of the PCBCAT. The Application Visualization System (hereafter referred to as AVS) from AVS Inc. is a commercial visualization package. AVS can be used as a standard visualization application, or it can be used as the basis for developing customized visualization tools. Other commercial packages also have

[*]to be presented at the 1995 ASME National Heat Transfer Conference, 6–9 August 1995, Portland, Oregon

these capabilities. We chose AVS because it is widely used, and it was already available on the workstations we use for development of the PCBCAT.

Our goal was to combine the display of the field data—a standard visualization task—with a display of solid objects, inlets, outlets and other structural features that define the calculation domain. In this paper we will refer to these model features as geometric objects. These objects define the physical problem, but in many conventional CFD codes such objects are present only as variations in the nodal material properties, prescribed nodal velocities, etc. In the PCBCAT these objects are an intrinsic part of the analysis code.

Displaying the objects that define the model geometry is enabled by two things. First, AVS is extensible, meaning that one can write custom modules for displaying objects, manipulating data, and responding to interactive user input. Second, the use of geometric objects in the PCBCAT analysis code allows the object data structures to be exported directly to AVS. The data structures contain the physical dimensions and locations of objects, along with material properties and overall results of the calculations such as average temperature and the total heat transfer rate.

Using the PCBCAT pre-processor output file as a source of geometric information, the AVS-based post-processor renders the PCB component geometries along with the field data. The result is a computer generated three dimensional graphical representation of the physical model combined with its associated temperature and flow fields. The user can point and click on components for identification, average temperature, and total heat transfer rate. The extensive capabilities of AVS along with additional customized features make an intuitive computational tool superior to conventional field rendering.

This paper is organized as follows. First we present an overview of the features and design philosophy of AVS. Then we describe the PCBCAT analysis tools as an input/output system that supplies data to the visualization system. Next we explain how the PCBCAT data types are incorporated into AVS so that geometric objects in the domain may be rendered along with the field data. Following that we discuss the basic design and functionality of the `pick_pcbcat` interactive AVS module. We conclude by presenting an example of using our post-processing tool.

Overview of AVS

AVS is a software package specifically designed for visualization of scientific data (Advanced Visuals Systems Inc., 1992). 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. AVS modules are categorized by their performance of four key functions.

1. **Data Input**. Data input modules either import or generate data and convert it into an AVS data type. Imported data can reside as text or binary files or also within specialized data formats such as HDF (described later).

2. **Filter**. Filter modules perform mathematical transformations on internal AVS data. These modules mathematically manipulate the field values or change the size of the data set. The result is an AVS data type that may be of the same or different type that the data input to the filter.

3. **Mapper**. Mapper modules convert AVS data into a geometry data type, which allows the data to be rendered by the data output modules. The mapper modules typically work by sub-sampling the field, interpolating and then creating an AVS geometry object representing some isosurface, slice plane or some other substructure within the field. This AVS geometry can then be passed on to a data output module.

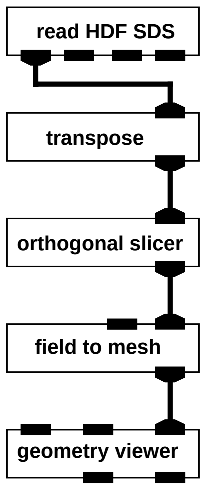

Figure 1: A simple AVS network to render a two dimensional slice through a three dimensional data set. The data is imported from an HDF (Hierarchical Data Format) file.

4. **Data Output**. Data output modules render AVS geometry data types on screen or store this data on an external device. Data output modules are also responsible for final manipulation of the geometry objects such as viewing angles, size, orientation, object properties, and lighting.

A sequence of data manipulation and visualization steps are implemented by assembling a group of AVS modules into a "network". Each module performs one of the four key functions described above. The AVS user creates the network with a graphical tool called the network editor. Individual modules have color-coded input and output ports corresponding to the AVS data type that may be passed through the port. Using the network editor to connect the appropriate ports of compatible modules allows data to flow between the modules. When completed a network is essentially a customized application for viewing and manipulating a specific type of user data. Note that assembling a network requires no programming.

Figure 1 is an example of a simple network that renders a two dimensional slice through a three dimensional data set. Each labeled box is a separate module. Lines connecting the modules represent data flow paths from the output port on the bottom of one module to the input port at the top of another module. In the network shown in Figure 1, the "read HDF SDS" module has three unused output ports, and the "geometry viewer" module has two unused input and two unused output ports. The "read HDF SDS" and "generate colormap" modules perform data input. The "transpose", "orthogonal slicer" and "color range" modules are filter modules. The "field to mesh" module is a mapper, and the "geometry viewer" module is a data output module.

The network defines the flow of data between modules. If a module allows variable parameters which control the processing or flow of data, that module will have a separate control panel which can be displayed when using the network editor. A control panel may contain file browsers or dialog boxes for data input, sliders and dials for specifying numerical parameters, or toggle switches for triggering specific functions. For example, a dialog box is used to specify the data file imported by the "read HDF SDS" module, and a control panel allows the user to interactively specify the orientation and location of the slice taken by the "orthogonal slicer" module.

Although the typical AVS user will work with just the network editor and the standard modules, there are many other elements to the AVS software package. AVS has module development tools, a scripting language, subsystems for rendering of geometry, images and graphs, and three volumes of primary documentation. Users can create their own modules by writing programs that call functions in the AVS Application Programmers Interface.

When velocity and temperature fields are rendered with standard visualization techniques the user can only infer the location of inlets, outlets, blocks and other fixed objects. It is especially difficult to locate fluid-solid interfaces in the temperature field. Though material properties are discontinuous at these interfaces, the temperature field is not. The solution, of course, is to superimpose images of important geometric objects on the rendered image of the field data. This requires that geometry information for each physical object be passed from the analysis codes to the visualization software. This is natural in the PCBCAT since, as is described in the following section, the geometrical description of each object is maintained in addition to the computational grid.

Overview of the PCBCAT

The PCBCAT are implemented as three separate programs that work together.

- A preprocessor
- The depth averaged flow model
- The three dimensional energy equation model

The PCBCAT have a minimal, text-based user interface implemented as a preprocessor. The user builds a model by creating a plain text file containing commands with associated arguments. Though this is somewhat old fashioned by the standards of modern computer packages, the physical model is defined in a compact language based on an intuitive description of physical objects in the domain (Recktenwald, 1995b). These objects are ultimately passed on to AVS for rendering and display. In the future we hope to augment the text-based preprocessor with a graphical user interface. This will require only minimal reworking of the problem-specification code in the preprocessor.

The preprocessor commands define physical objects such as inlets, outlets and electronic components without reference to the grid system used in the analysis. User-editable material property and device libraries are provided to aid in model development and to minimize data entry errors. The user specifies tolerances on the largest acceptable size of the computational cells and the preprocessor constructs the computational grid. The grid is generated only *after* the physical problem, including boundary conditions, is specified.

The CFD analysis is performed on a structured Cartesian grid. The grid is created such that control volume faces are aligned with the edges of physical objects which are constructed from two dimensional patches and three dimensional blocks. The patches are used to impose boundary conditions. The blocks are used to prescribe material properties and heat sources. Electronic devices, which are defined in the device library, are modeled as a combination of three block-type objects. The PCBCAT are written in C, which allows for the explicit creation of the patch, block, device, etc. data types. The object data types are used in all parts of the analysis code except for the solution of the linearized equations.

The PCBCAT programs execute in sequence and communicate via data files. The flow of information between these programs is depicted in Figure 2. A model of a particular board is completely described by a text file that is represented in Figure 2 by the box labeled "`user.input`". The preprocessor reads `user.input` and creates a temporary file labeled "`pcbcat.cntl`". The `pcbcat.cntl` file is a translation of the user input data into a format that is easily readable by the analysis codes. Whereas the data in `user.input` is relatively free form and contains comment statements, the `pcbcat.cntl` file has a rigid structure and no extraneous text. The `pcbcat.cntl` file contains the complete problem description including the PCBCAT data structures used in the analysis.

The `pcbcat.cntl` file controls the execution of the depth-averaged model and the three dimensional energy equation model. For visualization purposes only the `fname.HDF` and `fname.objs` files

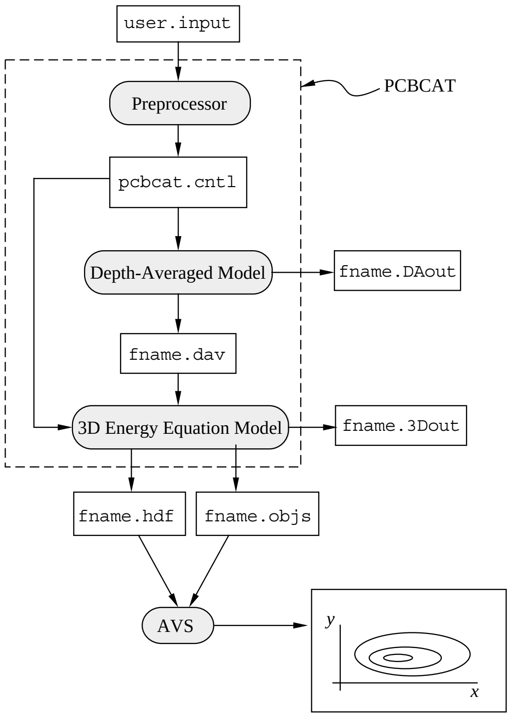

Figure 2: Schematic of the flow of data between the programs. Executable programs are depicted as shaded oval boxes. Data files input to or output from the programs are shown as rectangular boxes. The box in the lower right corner represents the graphical output from AVS, in this case a contour plot.

created by the energy equation model are important. The “fname” part of the file names is a generic name representing the actual file name requested by the user in the input to the preprocessor. The field data output from the analysis codes is stored in `fname.hdf`. The object geometry data is stored in `fname.objs`.

Field data from the PCBCAT analysis codes is stored in Hierarchical Data Format (HDF), which was developed at the National Center for Supercomputing Applications at the University of Illinois at Champaign, Urbana. We chose HDF because it is a widely supported, machine-independent data format. HDF files are fully self describing, i.e. any software designed to read HDF will only need the HDF file itself and not some additional description of what is contained within it. HDF files are readable by many different data visualization programs, e.g., Spyglass Transform, SGI Explorer, AVS, and MATLAB. The HDF file contains the geometric data necessary to specify the spatial distribution of the dependent variables (velocity and temperature), but not the geometric description of solid objects in the domain. Storing the geometry objects in a separate file (cf. `fname.objs` in Figure 2) compromises the utility of HDF somewhat, and in principle we could also store the geometry object data in the HDF file. We may do this in the future. At this point we opted for the more straightforward strategy of exporting the PCBCAT object descriptions directly without packing this information into the HDF file.

The `pick_pcbcat` Module

The custom AVS module developed for the PCBCAT project performs two basic functions. First it reads the geometry information from the `fname.objs` file. Second, it responds to interactive user input. The custom module is called `pick_pcbcat`. The “pick” in the name refers to the ability to support interactive user queries about the objects in the calculation domain. The user clicks on objects with the mouse and the `pick_pcbcat` module displays information such as object name, average temperature and total heat transfer rate. The `pick_pcbcat` module is written in C and linked to the AVS development library. The `pick_pcbcat` module consists of functions which respond to calls from the AVS control executive.

Figure 3 is a schematic of how the `pick_pcbcat` module works in an AVS network. The figure is a combined representation of an AVS network (cf. Figure 1) and the flow of information between PCBCAT modules (cf. Figure 2). Figure 3 contains some AVS modules, e.g., “read HDF SDS”, “field to mesh”, and “geometry viewer”, which are represented by shaded boxes. It also shows data files as unshaded boxes, and internal data exchanges as unshaded boxes with rounded corners.

The `pick_pcbcat` module communicates directly with the geometry viewer. The reading, transformation, and rendering of the field data is independent of reading and rendering of the geometric objects in the domain. In fact a very useful AVS network consists of just the `pick_pcbcat` and geometry viewer modules. This minimal network displays the model geometry and allows the user to query objects in the domain.

The `pick_pcbcat` module first reads the data in `fname.objs`, which is output from the PCBCAT models. From this information the `pick_pcbcat` module constructs the geometry data types for display by AVS. The geometry data types consist of polygons that define the surface of each object. Each instance of the geometry data type is given a name which is used later in processing of the user queries. Complicated objects may be represented hierarchically as a parent with children. The construction of these objects is represented schematically in Figure 3 by the flow of “object geometry” from the `pick_pcbcat` module to the geometry viewer.

The operation of the `pick_pcbcat` module is determined by settings in its control panel. With the control panel the user selects which `.objs` data file is actually read by `pick_pcbcat`. The control panel also allows the user to choose whether the object geometry is rendered with wireframes or with shaded solids, and whether object information is displayed when objects are selected with the mouse.

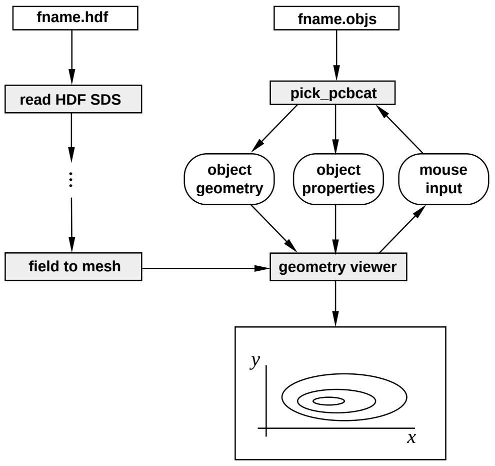

Figure 3: Flow of information between AVS, the `pick_pcbcat` module and the PCBCAT data files.

User input is processed by providing a service routine in the code that defines the `pick_pcbcat` module. The service routine responds to data flowing back upstream to it from the geometry viewer. This flow is depicted by the “mouse input” data in Figure 3. What the `pick_pcbcat` module actually receives is the name of the AVS geometry object that was clicked upon by the the user. This name is matched against the list of names created by `pick_pcbcat` when it read the PCBCAT object data from `fname.objs`. When `pick_pcbcat` matches the name of the AVS geometry object with the PCBCAT object, it returns to AVS a command for displaying the data requested by the user. Maintenance and processing of the PCBCAT object lists requires little additional programming because the PCBCAT codes are organized around these lists. In fact, the `pick_pcbcat` code is built by linking to the same library used to develop the PCBCAT analysis codes.

Example

We now present an example of the output from the visualization system. The example is from a simulation of an experimental data set involving flow past a uniform array of heated blocks (Garimella and Eibeck, 1990). We use this data set as a benchmark for validating the quantitative accuracy of the PCBCAT. The flow occurs in a duct 36 cm wide and 2.2 cm high. Thirty Aluminum blocks $2.54 \times 2.54 \times 1.0$ cm high are mounted to the bottom of the duct. In the experiment only one heater was powered at a time. In the calculations that follow we (arbitrarily) have all heaters powered. The fluid is water (which was used in the experiments) and the Reynolds number of the flow based on duct height is 1000.

Figure 4 is a hardcopy of the geometry viewer window. Unfortunately the quality of the images we were able to print does not match that of the images on the screen. The temperature field is rendered on a horizontal plane near the bottom of the domain. The user can select slice planes normal to any one of the coordinate axes, and at any position in the domain. The slice shown in Figure 4 is effective at displaying the overall temperature field. The heated blocks are rendered as solid gray surfaces with shading to indicate perspective. The “heater_61” label was generated by the `pick_pcbcat` module in response to a mouse click on the heater in the sixth row and first column. The name heater_61 was actually specified in the input file to the PCBCAT preprocessor. (A more descriptive object name can be chosen at the user’s discretion.) The average temperature, 26.9 ℃ and total power dissipation, 5 W, of the heater are obtained by `pick_pcbcat` from the `fname.objs` file for this simulation.

Rendering the heaters as solid objects obscures their internal temperature distribution. By toggling a switch in the `pick_pcbcat` control panel the user can choose to render the objects as wireframes instead of solids. Figure 5 shows a zoomed-in image of Figure 4 with wireframe rendering of objects. The heater blocks are made of Aluminum so their temperature is essentially uniform. In this case the user could infer the location of the blocks, but the wireframe makes such inferences unnecessary.

Conclusions

From our experience with AVS and the PCBCAT we draw the following conclusions.

- Using the AVS visualization system provides significant leveraging of programmer time. Most of the data manipulation and display can be achieved by assembling standard modules into a network. Customization efforts are only focused on the addition of features, not the duplication of standard features.
- There is a significant learning curve for AVS module development. However, once the programmer is familiar with AVS, productivity is good.
- Developing customized modules greatly enhances AVS as a visualization tool. In our case we added geometric information that makes the final display a more intuitively and aesthetically appealing. The PCBCAT geometric objects become part of the image that may be rotated, resized and zoomed.
- The design of the PCBCAT greatly simplified the visualization process. Physical objects such as inlets, outlets, and solid structures are fully integrated into the analysis codes. This made it trivial to export object-based geometric data and results to the visualization system.
- AVS (and comparable visualization systems) are useful to researchers developing custom CFD programs. We will continue to use our modules and develop new ones.

Acknowledgments

Development of the PCBCAT analysis codes was supported by a grant from Intel Corporation.

References

1. Advanced Visuals Systems Inc., 1992, *AVS Programmers Manual*, Waltham, MA.
2. Garimella, S. V. and P. A. Eibeck, 1990, "Heat transfer characteristics of an array of protruding elements in single phase forced convection", *Int. J. Heat Mass Transfer*, **33**(12): 2659-2669.
3. Recktenwald, G.W., 1995a, "Prediction of Device Temperatures with Depth-Averaged Models of the Flow Field over Printed Circuit Boards", submitted for presentation at the 1995 ASME Winter Annual Meeting.
4. Recktenwald, G.W., 1995b, "Using the PCBCAT to Model Convective Heat Transfer from Electronic Devices on Printed Circuit Boards", submitted for presentation at the 1995 ASME Winter Annual Meeting.

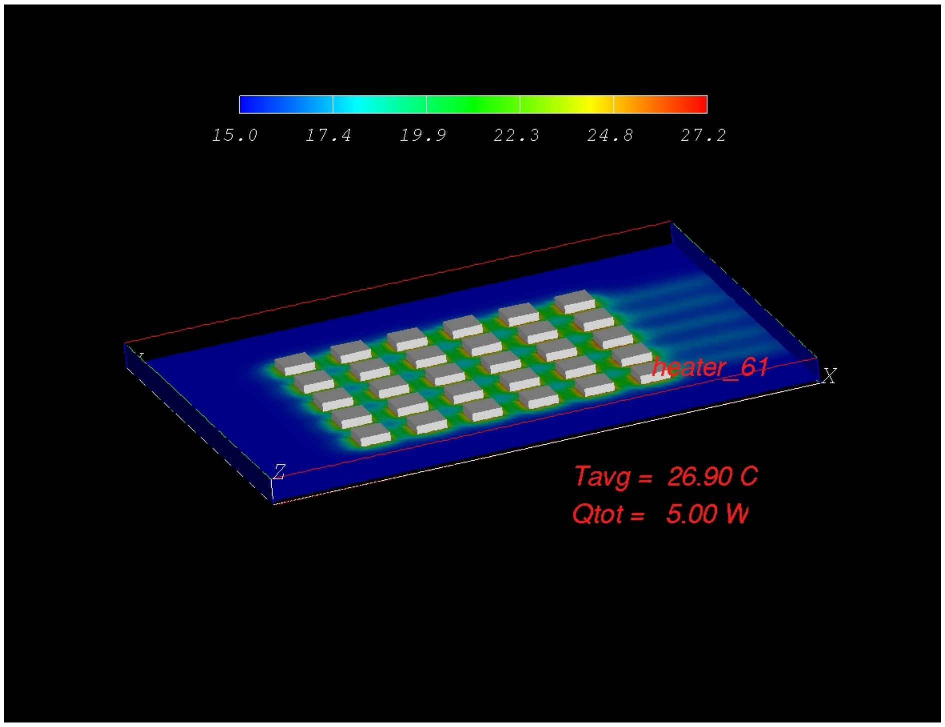

Figure 4: Temperature field for flow past a 6 by 5 array of heated blocks. The average temperature and total power dissipation of the block in row 6, column 1 is displayed. The label “heater_61” is chosen by the user in the input file to the PCBCAT preprocessor.

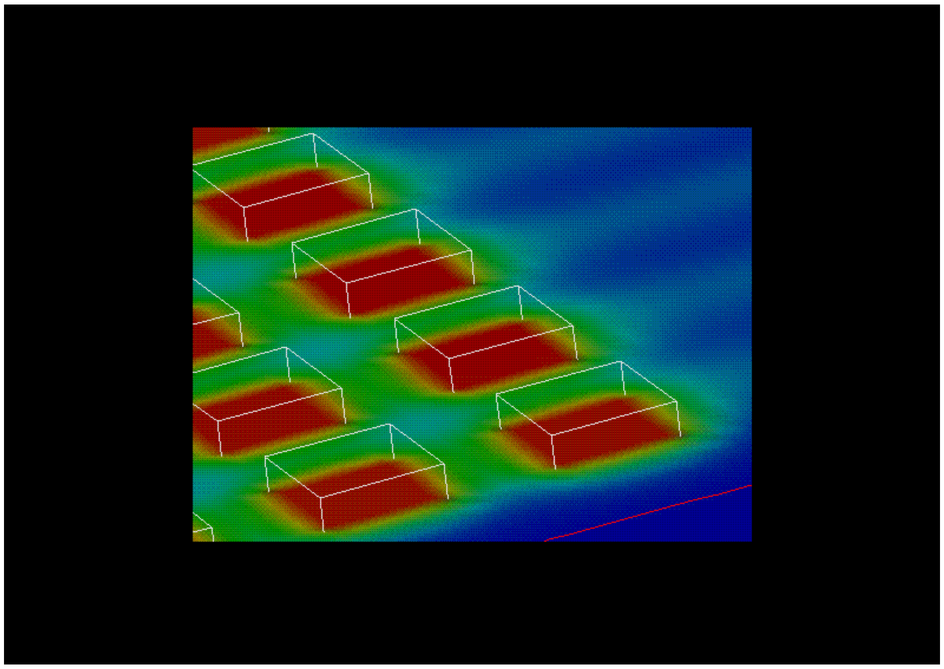

Figure 5: Zoom-in of the temperature field near the block in row 6, column 1. The blocks are rendered as wireframes to make their internal temperature variations visible.