AVS

A Tutorial

Scientific Visualization
Charlie Patterson
(charliep@cc.gatech.edu)

Starting AVS

AVS only runs on the DEC Alpha workstations (we have two floating licenses meaning that any two Alphas can run it at the same time, but no more). The starting command is

```
% avs
```

AVS' Home

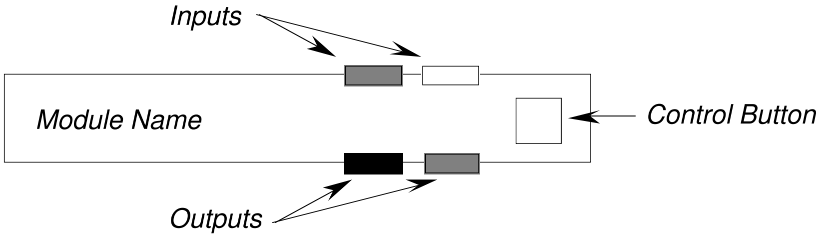
AVS actually resides in */usr/local/avs*. To find its examples, modules, and networks (described later), you can check there.

> ***For Daring Users:***
>
> Once you are used to AVS and have your own data files, you may want to use the following starting command instead:
>
> `% avs [-d` *`dataDirectory`* `] [-netdir` *`networkDirectory`* `]`
>
> That is, when you attempt to open a network or some data, AVS starts looking in these directories. If you do not specify which directory to look in first for data or networks, AVS will repeatedly start you at '/usr/local/avs/data' and you will have to specify a complete path to get to your data.
>
> If you do not have a .avsrc file, '/usr/avs/runtime/.avsrc' is used. This default startup file specifies where to find the modules, though there are many more things that can be put here, like where to look first for your data.

Modules and Networks

AVS is actually a programming language, but a visual one. The smallest programming unit in AVS is a "module," which is like a subroutine - it has inputs and outputs and somehow manipulates data. (See figure below) Modules are visually shown as rectangles with their inputs at the top and outputs at the buttom. A module might, for instance:

• read in a particular type of data (e.g., CFD, Chemistry, or generic arrays)
• reduce the total number of samples in a volume of data
• combine two sets of data end to end
• combine two sets of data by adding them together
• find the edges in an image
• produce a ray-traced rendering of a volume
• draw data as points in space in a 3D image



Connecting together the inputs of one module to the outputs of another will ultimately form a "network" of modules, which is similar to an entire program.

> ***To connect two modules***
> (the output of one must be compatible with the input of the next) use the middle mouse button and touch the correct output of the first module, then, (holding in the mouse button) drag to the correct input of the second modules.
>
> ***To disconnect two modules***
> use the right mouse button and touch the correct output of the first module, then, (holding in the mouse button) drag to the correct input of the second modules.

Typically, a network will begin with modules which read in data, continue with modules which filter and rearrange data, then send the data on to modules which convert from data to drawing primitives, and finally use modules which draw the primitives (or write the data).

AVS Fields Types

You will probably deal most intimately with the AVS Data Type field. This is because your input data will more than likely need to be converted to a field. Other AVS data types exist, like colormaps, geometries, and images, but they are more useful internally to your networks in a way as to see your fields (data). Also, UCD or unstructured cell data may be important for some input data, but it is more complex and will be left out of this tutorial.

Field Designation

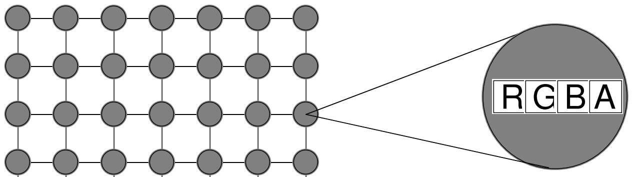
Fields store **values** in any number of array-like **dimensions**.

A **value** can be any number of homogenous **elements**.

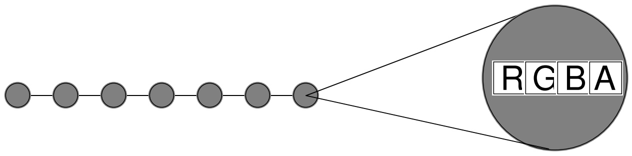
An **element** can be a byte, integer, floating point, or string.

Examples:

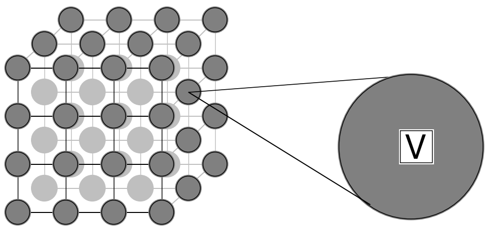
2D field 4-vecter bytes
(i.e. image with RGB and Alpha ranging from 0 to 255 each)



1D field 4-vector bytes (scalar means l-vector)
(i.e. colormap)



3D field scalar float
(i.e. CAT scan)



Many modules deal with arbitrary dimension fields and vectors; and arbitrary numeric data. In these cases, the modules do not specify numbers or data types.

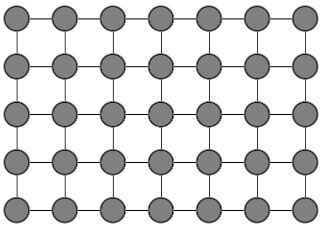
Example:

field scalar byte	(any dimension field)
field 2D scalar	(any data type [integer, float, byte])
field 3D float	(any sized vector)
field scalar	(any dimension field, any sized vector)
field	(anything goes)

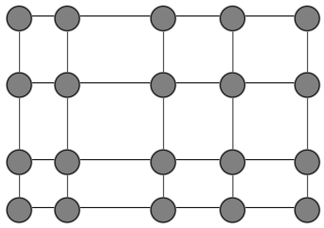
Point Locations Within a Field

Points of a field don't have to be considered at equal distances. There are actually three ways to locate points in a field:

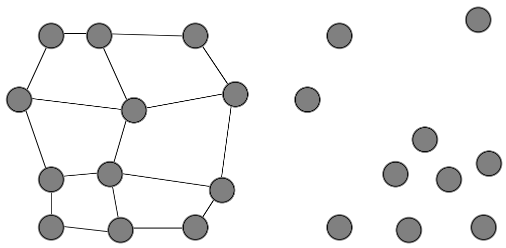
1) Uniform - at equal distances from each other



2) Rectilinear - axis are still orthogonal but spacing isn't equal



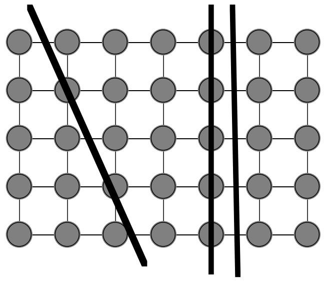
3) Irregular - points are located anywhere



Notice that it helps to break the usual visualization of matrices as values fitting in boxes, and think of values as fitting at the crossing of lines.

Continuous Representation

AVS may also consider a field of values as continuous by interpolating between the points. (Often, input is a discrete sampling of continuous information.) AVS may, for instance, cut a 1D slice from a 2D field which is not orthogonal to an axis. In this case, it will need to reconstruct in-between values.



Converting Data To AVS Field Type

AVS can usually obtain your data by use of a simple conversion file. AVS also understands several common input formats from chemistry, physics, and other visualization programs.

Example:

The following is the conversion file for air species data saved as a huge array. It is stored in time by latitude by longitude order

```
# AVS field file
#
#
ndim=3                        # total number of dimensions
dim1=40                       # values in dimension 1
dim2=40
dim3=32
nspace=3                      # for simple fields, the same as 'ndim'
veclen=1                      # vector length (values per grid point)
data=float
field=uniform
units=ppb                     # not necessary
#
variable 1 file=no2.asc filetype=ascii skip=0 offset=0 stride=1
```

Some Starter Modules

field tools	image tools	color tools	3D tools
read field	read image	generate	render geometry
combine scalars	read tiff	colormap	display pixmap
extract scalar	crop	color range	field to mesh
extract vector	clamp	colorizer	orthogonal slicer
combine scalars	contrast		arbitrary slicer
write field	composite		
print field	write image		
statistics			

> ***For Daring Users:***
> Many more modules exist than you will find upon starting AVS. Many of them are at the IAC ftp site (avs.ncsc.org) and are not on the Alphas right now.

Module Building Overview (Advanced Topic)

Modules are executable programs which act as subroutines. They pass data through sockets to each other. These links are under the control of the AVS Kernel. You must write the following functions and then AVS will work them into a module for you:

Description Function

Name the module and its type (input, output, mapper, or filter) Name the computatio, destruction, and initialization functions (described below)

Name and type its inputs, outputs, and parameters Add limitations to these parameters when you don't need complete flexibility. This eases coding and usage.

Functions are written and supplied in libraries to help with your descriptions.

Computation Function

Compute the outputs from the inputs and parameters.

Many functions are written and supplied in libraries for this task. They are specific to fields, colormaps, and other AVS data types.

Initialization/Destruction Function (optional)

Tell AVS what to do when your module is introduced into a network or removed from one.

Used to allocate/deallocate special memory, a window, etc.

Functions are written and provided in libraries to help set up these functions.

Help File

A final, professionally done module should include help documentation, which appears in the module control window.