Importing data

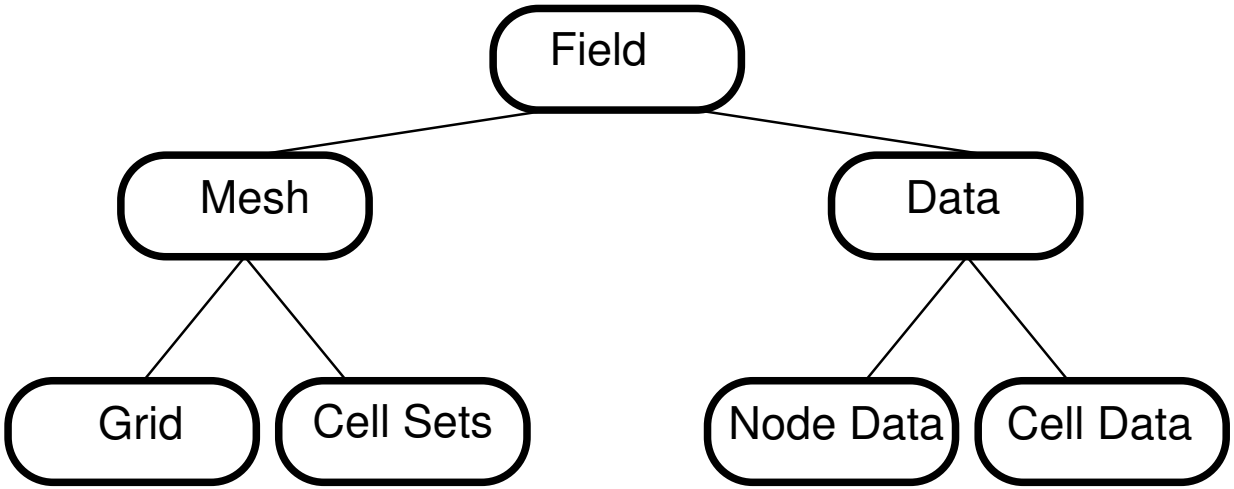
Importing data

- The Express Field
- Overview of representing different data types
- How to import data into AVS/Express

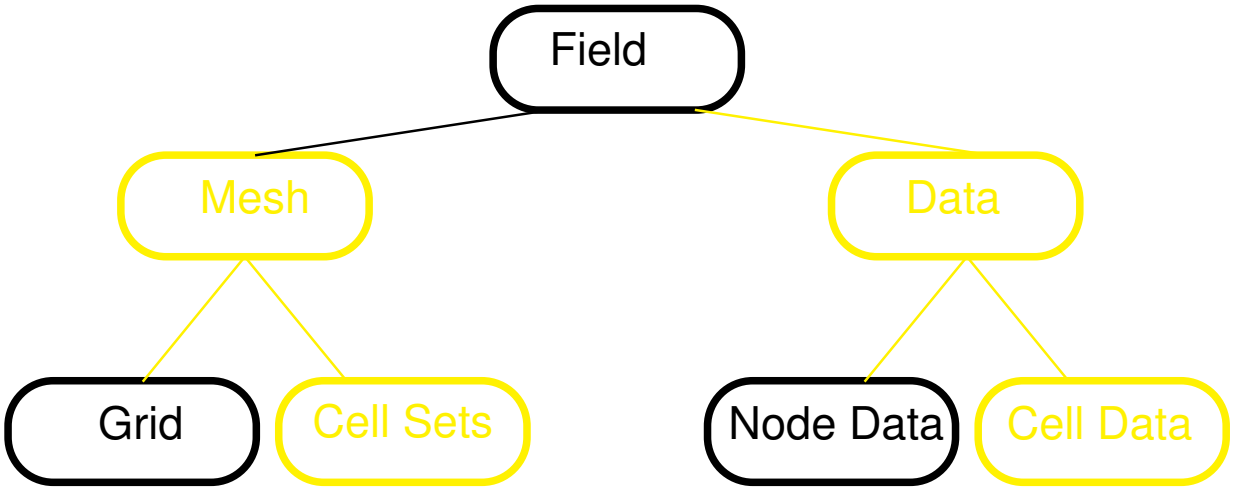
What can you import?

- Read (and Write) netCDF
- GISWorldDataBank and GISDigitalElevation
- Read (and Write) Image
  - BMP, GIF, JPEG, PBM, SGI image, Sun Raster, TIFF, avs .x
- User defined data is discussed later
- The Express FLD provides support for AVS5 field, UCD and GEOM
- Lets look in more detail at the FLD...

The Express Field

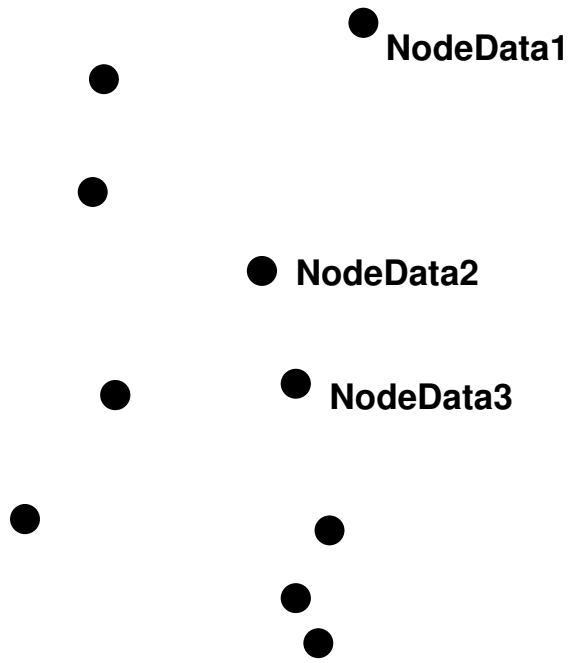


Grid and node data

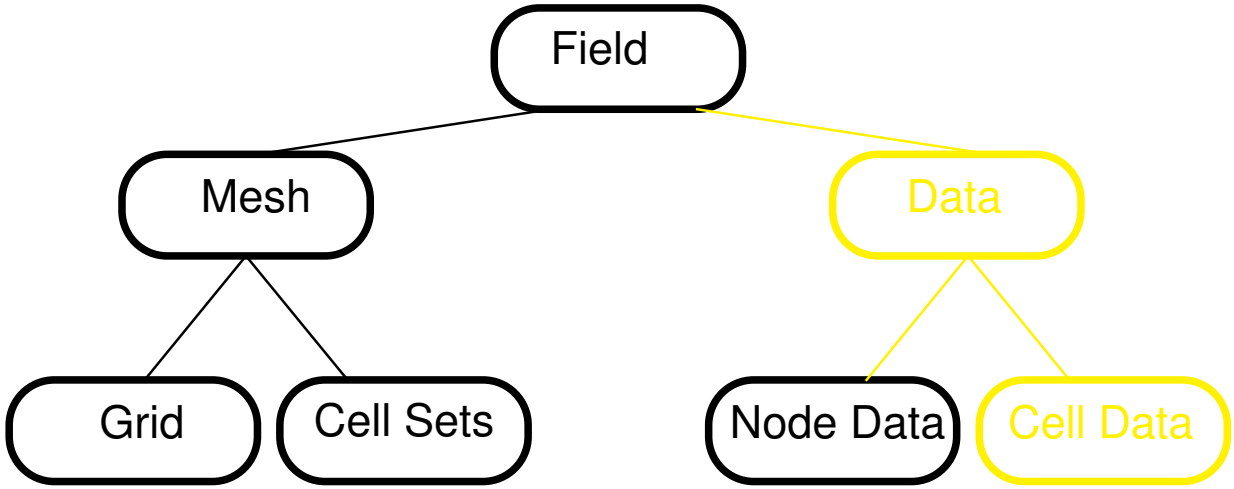


- Positions and data components
- Data components can be scalar and vector

Grid and node data

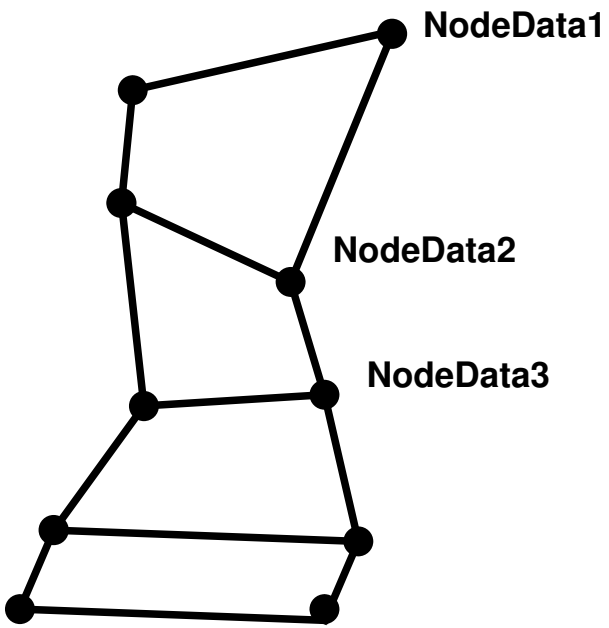


Connectivity information

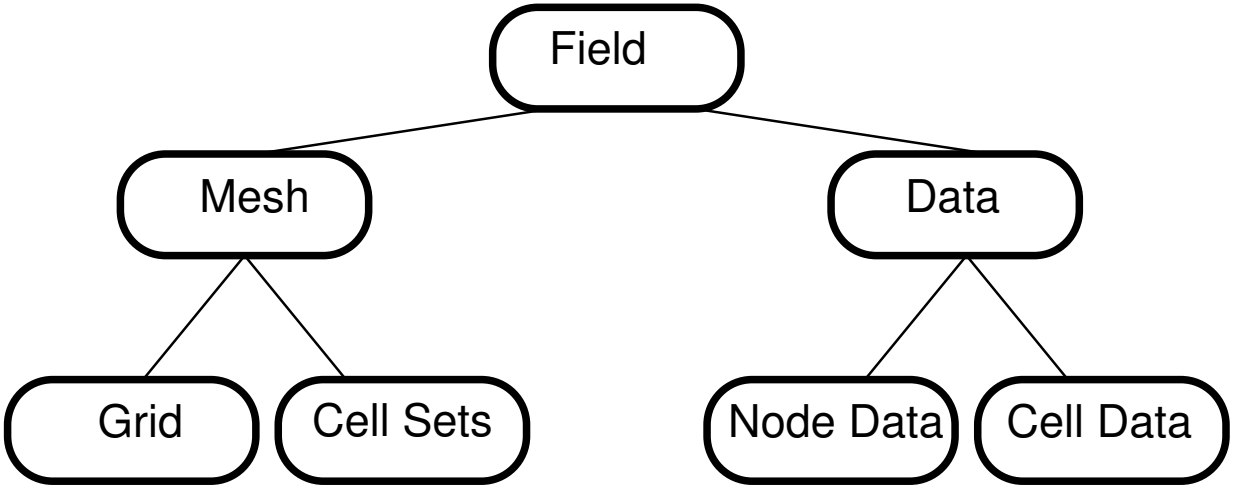


- cell is a geometric entity
- cell_sets are a collection of cells

Connectivity

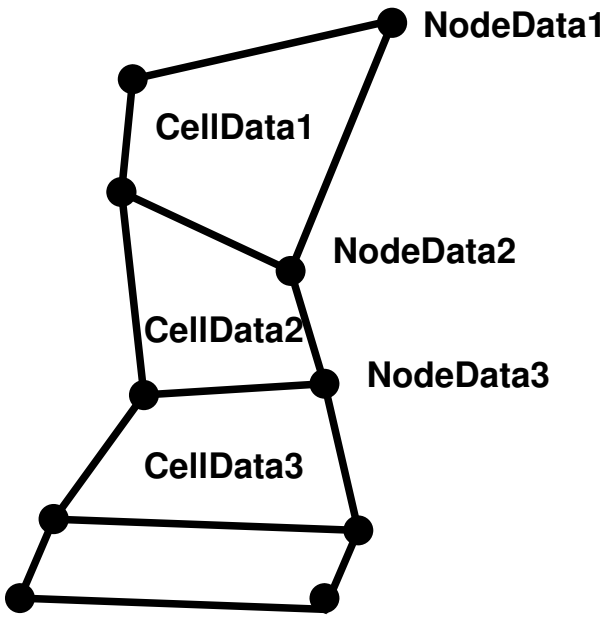


Cell Data



- Cell data completes the definition

Cell_data


NodeData1
CellData1
NodeData2
CellData2
NodeData3
CellData3

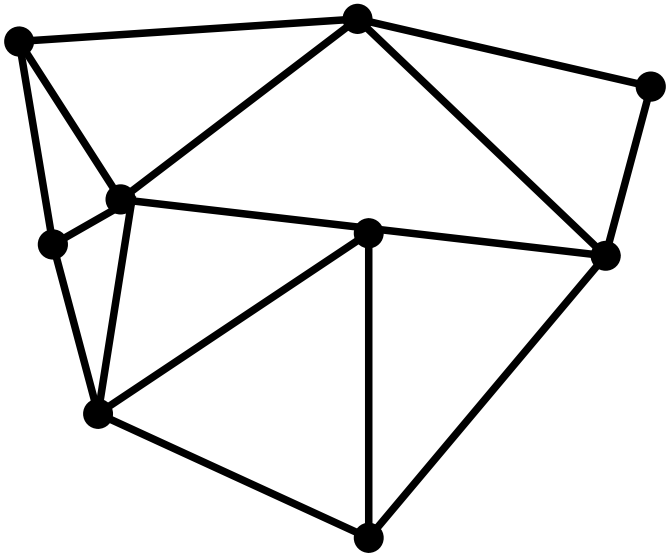
What do you specify

- Some portions of the FLD do not need to be specified explicitly but can be calculated from other information
  - connectivity
  - grid (node) coordinates
- This all depends upon the data you are trying to represent

Mesh types

- There are four basic mesh types:
  - unstructured (c.f. UCD)
  - structured (c.f. AVS5 irregular field)
  - rectilinear (c.f. AVS5 rectilinear field)
  - uniform (c.f. AVS5 uniform field)
- Here are some examples...

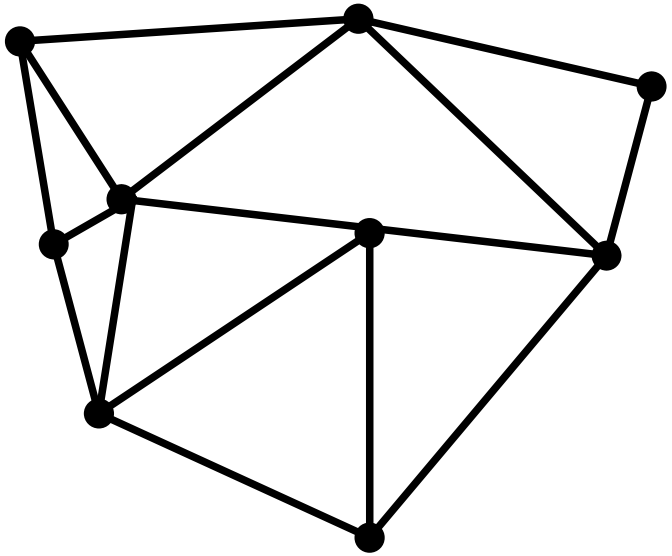
Unstructured meshes



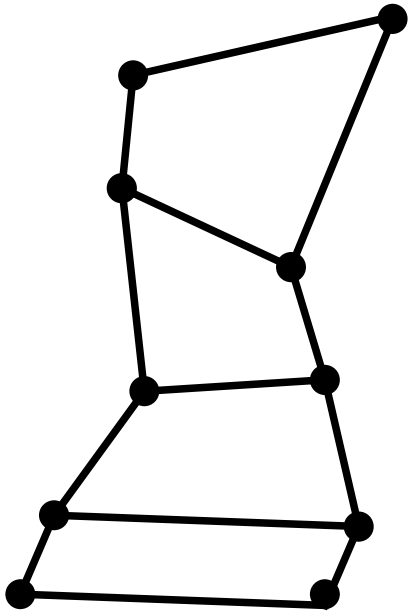
- The only type of mesh where cell sets are required

Structured meshes

unstructured

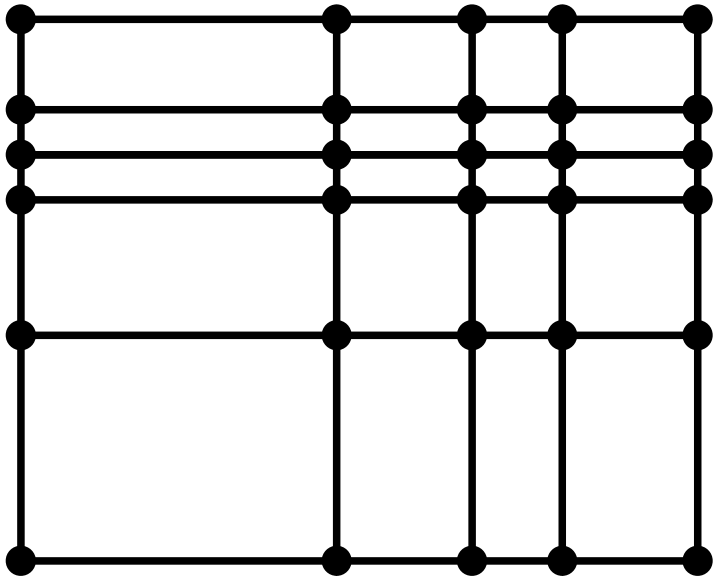


structured (2,5)



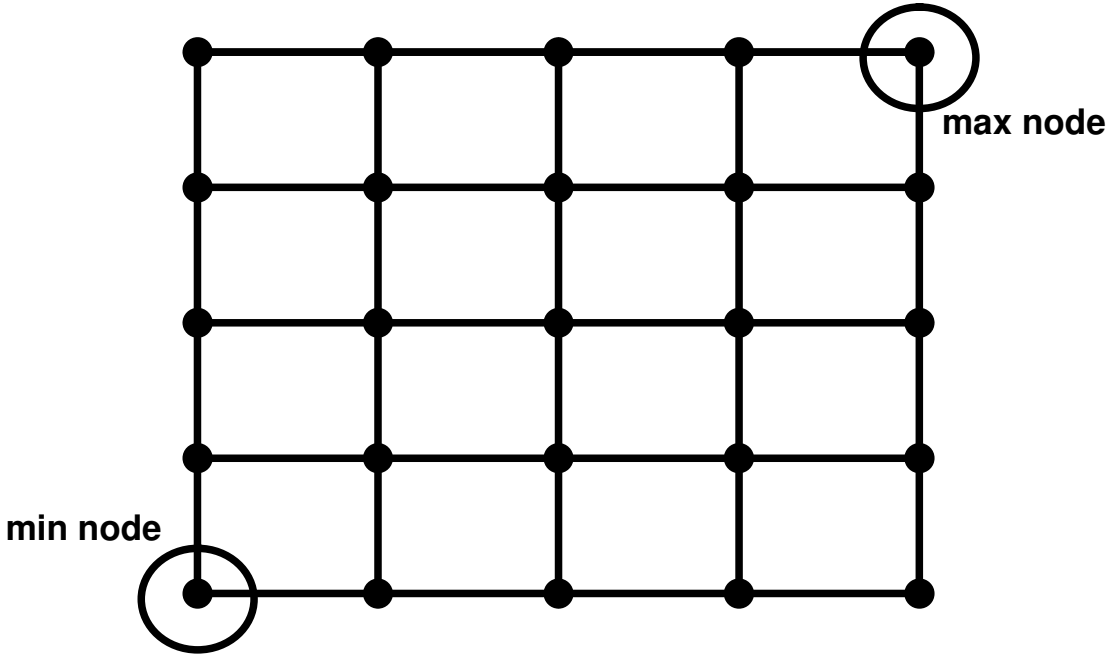
- Connectivity is implied by dimensions (all structured meshes have dimensions specified)

Rectilinear meshes



- You only specify the grid (node) coordinates along each axes

Uniform mesh



- You just provide grid (node) coordinates for minimum and maximum

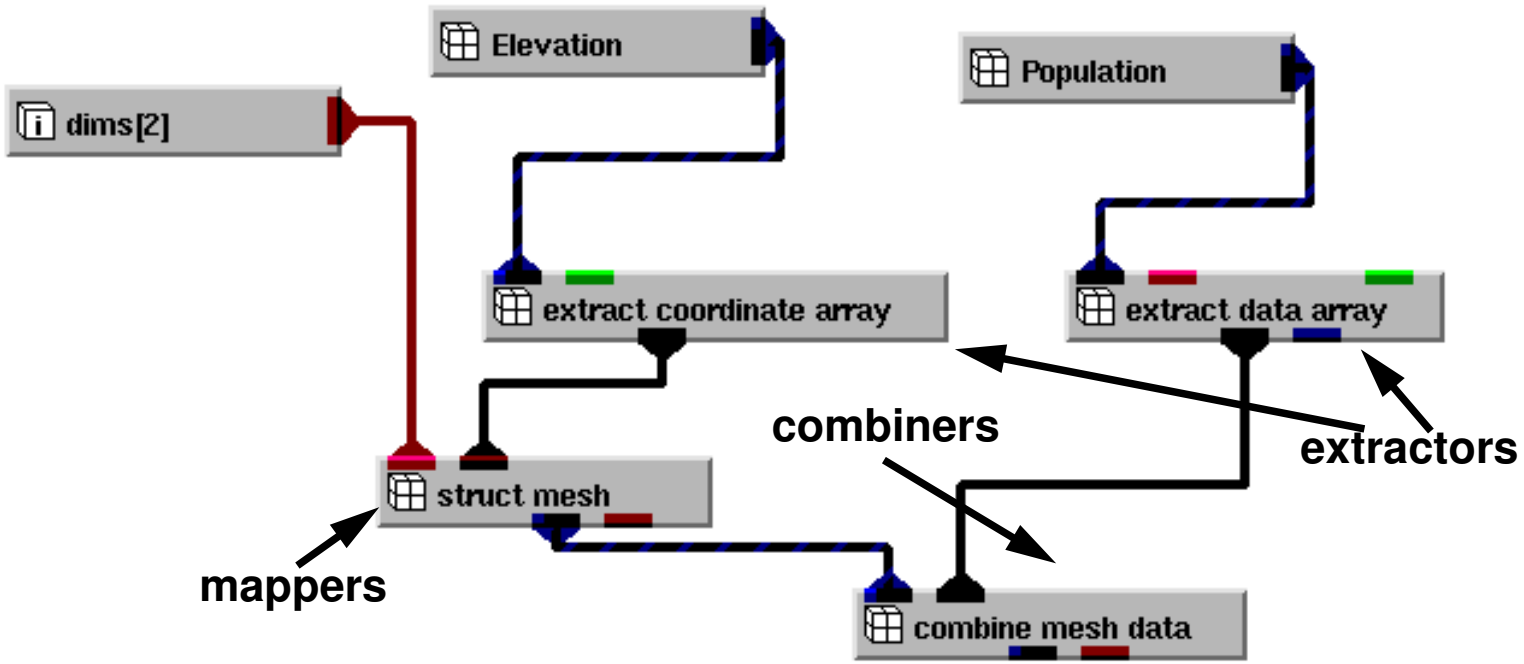
Mapping data into fields

- There are six different methods...
  - Read_Column_File
  - AVS Field file format
  - Database query
  - File access objects
  - Custom reader for arrays of data (module you write code for)
  - Custom reader to generate Express FLD
- Increase in complexity down the list
- Which one do you use?

Which one do you use?

- The good news is that there are specific reasons for each one...
- If you import the data into the system as arrays then you need to make use of:
  - mappers - produce a field from data arrays
  - combiners - merge mesh and data information
  - extractors - select part of Express FLD structure

Mappers, combiners and extractors



Summary

- Read Column File
  - Data is arranged in columns, ASCII
  - Output is a field
  - Useful for simulating database query
- AVS Field File
  - format of file can be described using a simple header, Binary or ASCII
  - Output is a field
- Database query
  - data is stored in relational database, need AVS/Express Database Kit
  - Output is an array (need appropriate mappers)

Summary (cont)

- File Access Objects (Add File Import Module)
  - data which can be easily mapped into arrays, binary or ASCII
  - Output is an array (need appropriate mappers)
  - Can calculate key values from components in datafile eg., dims, offsets
- Custom reader module (write your own code)
  - data must be (or is easier to) parse with code to produce arrays
  - Output is an array (need appropriate mappers)
- Custom reader module (write your own code)
  - code uses FLD access libraries (C) or classes (C++), output is a field
  - use if data is complex format e.g., unstructured field and no field mapper exists

User defined data

- There is no explicit type for this data
- You simply create an object in Express to represent the data
  - e.g., Patient Scan Data

```
group {
int      pid
string   pname
Field    scandata
int      DateofScan
}
```

- Other modules could process Field
- You could create a UDT for holding the date...

Importing data using Read Field

- The overall scheme
- Describing the data
- Some examples

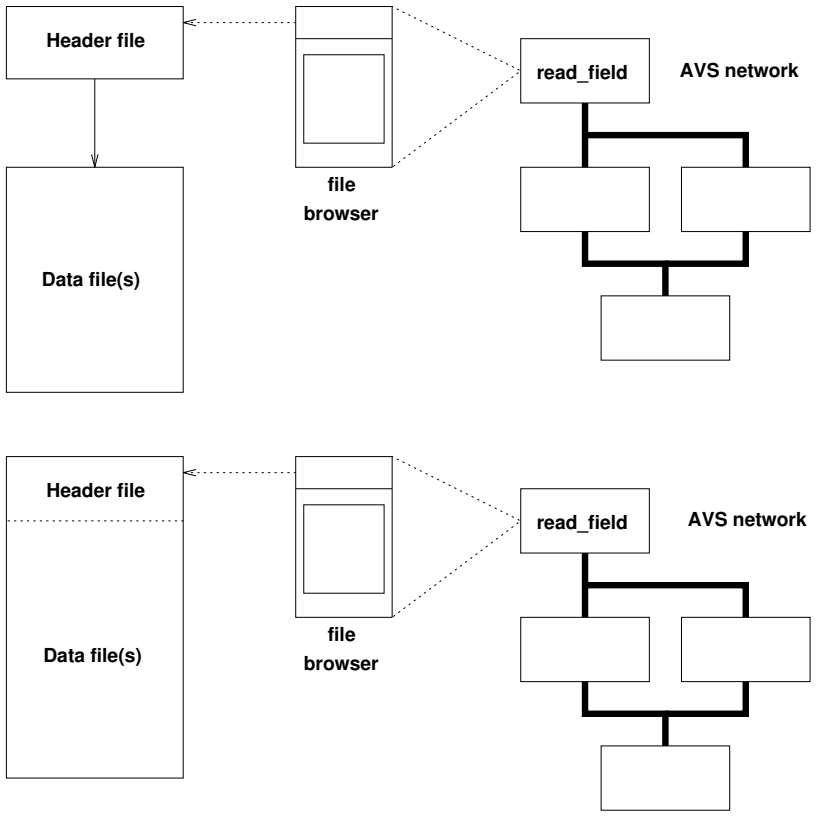
Overall scheme

- The AVS field datatype is one of many ways to present arrays of data to AVS/Express.
- The data is described by an **ASCII header file**
- The file can optionally
  - contain the data or
  - *reference* the data via an *external* file or files
- Referencing the file(s) is a better way of working

`Read_Field` module

- The **`Read_Field`** module is used to import data into an AVS/Express network
- The **`Read_Field`** module has a file browser to select the header file

Overall scheme



Describing the data

Dimensions

- You must specify the dimensions of the data:
  - 1D: list of scattered data
  - 2D: Image
  - 3D: volume of data
- These specify the **computational** space
- You must specify the dimensions of the data as it is passed to AVS/Express
- These specify the **coordinate** space

Mapping methods

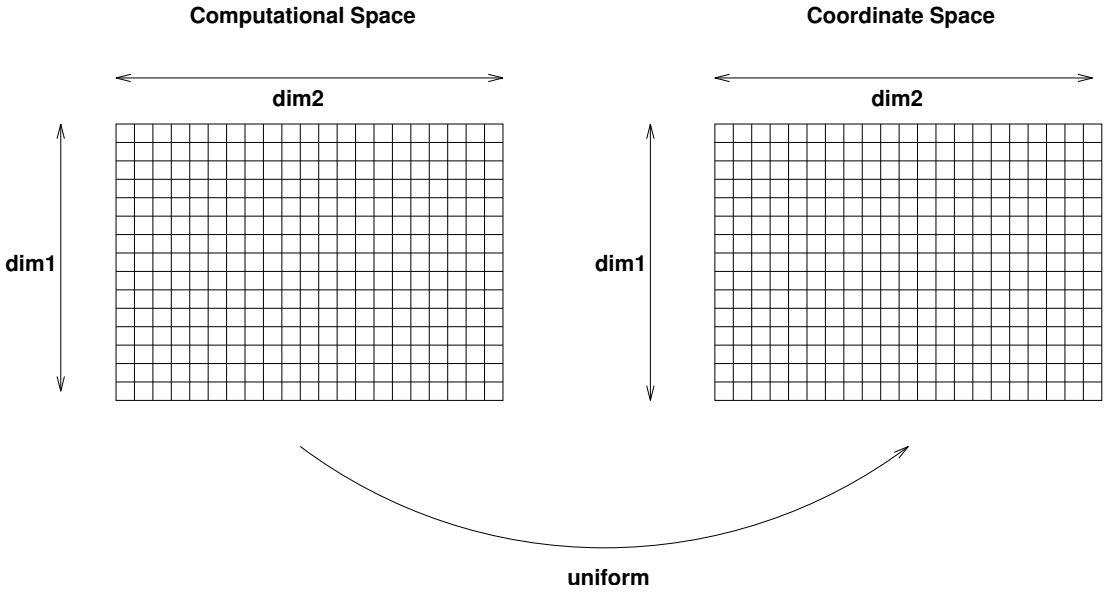
- Uniform
  - used to import regular grids of data
- Rectilinear
  - each data point is mapped to a physical coordinate
  - orthogonal Axes but unequal spacing
- Irregular
  - dimensions of computational and coordinate space may be different

Mapping methods

UNIFORM

- The data is not transformed as it is imported (direct & implicit)
- The dimensions of the computational and coordinate space should be the same
- Used to import regular arrays of data
- No need to supply physical coordinates

Uniform mapping

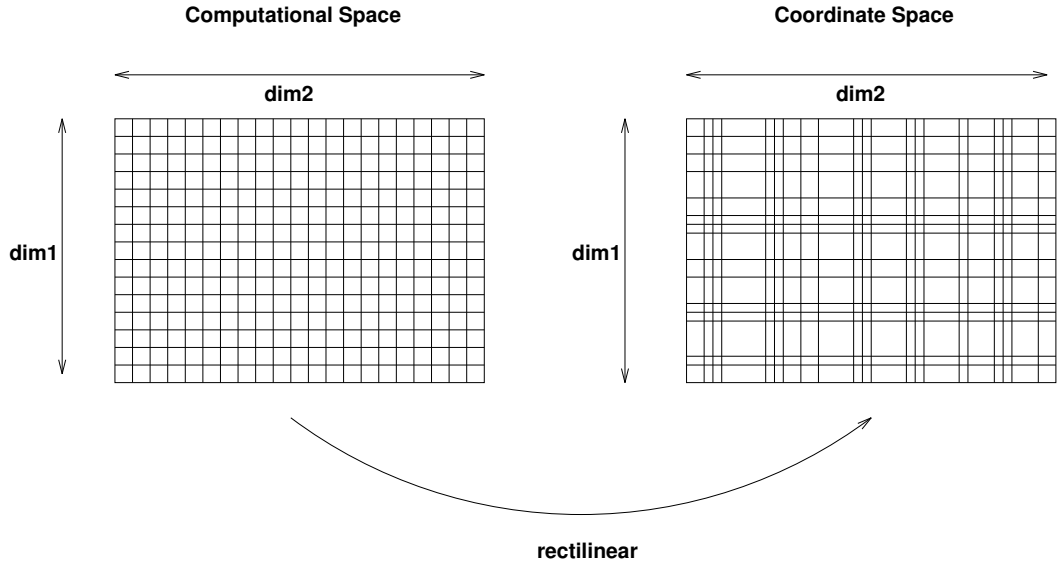


Mapping methods

RECTILINEAR

- Each dimension in the data has an explicit coordinate mapping
- The spacing along the coordinate axes can be non-uniform
  - e.g., logarithmic axes
- The dimensions of the computational and coordinate space should be the same.

Rectilinear mapping

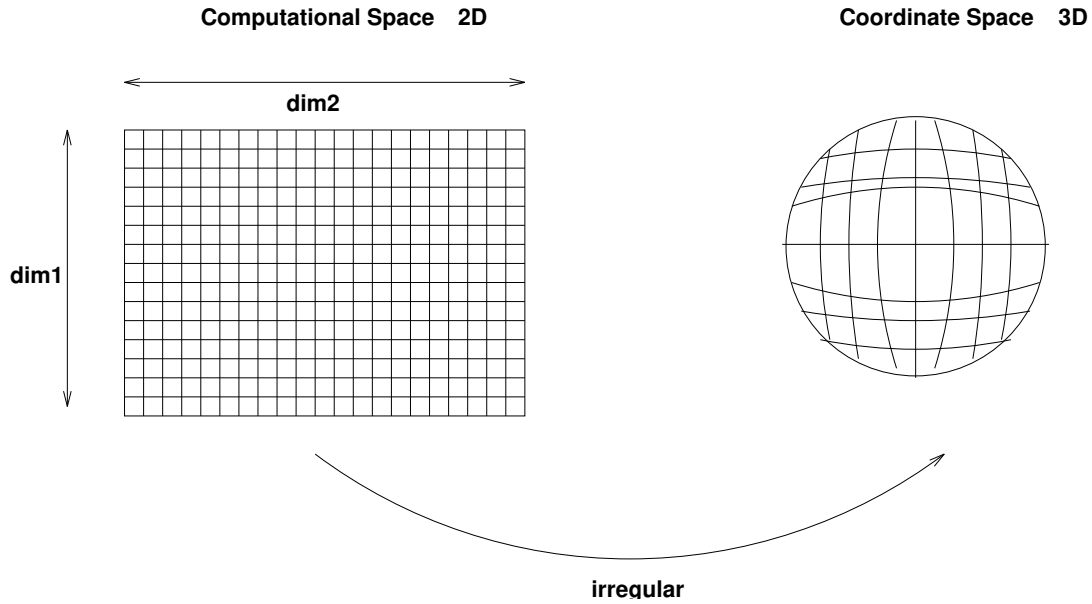


Mapping methods

IRREGULAR

- Each data element is mapped to a point in the coordinate space
  - coordinates must be supplied
- The dimensions of the computational and coordinate space can be *different*
  - e.g., 1D scattered data in 3D space
- Sometimes called *Curvilinear Coordinates*

Irregular mapping



Describing the data elements

- The number of data components
- The datatype
  - byte, integer, float, double
- Mapping method
- Location and layout of the data files
- A description of each data component
  - specifying the *name*
  - specifying the *units*

Format of the header file

- The file should have an extension of **`.fld`**
- The first line should be:

```
# AVS field file
```

- The rest of the file contains a number of keywords and values
  - `keyword = value`
- Comment lines begin with a #

Keywords in the header file

- `ndim` = number of dimensions in computational space
- `dim1` = specify the size of dimension 1
  - can be thought of as the x dimension
- `nspace` = number of dimensions in coordinate space
- `veclen` = number of data components per element
- `data` = type of the data components
- `field` = mapping method
- `label` = label for each data component
- `unit` = units for each data component

Reading the data

- Once the data structure has been defined you specify the location and format of the data file(s).
  - The data file(s) can be binary or ASCII
- Each data component needs a line:

```
variable N < number of format keywords >
```

- Each item of coordinate information needs a line:

```
coord N < number of format keywords >
```

- Each line can reference different files

Reading the data (2)

- There are five format keywords for reading the data
  - `file =` location of data file
  - `filetype =` format of the data file, `binary` or `ascii`
  - `skip`: number of lines or bytes before the data is found
  - `offset`: number of columns to jump over before the data is read (ASCII only)
  - `stride:` how many steps forward before the next item is reached
- In the case of ASCII the steps are in delimited items

Format

- All the format keywords must be on a single line
- In each of the examples a “-” indicates that the line is continued

```
This is a very very very very very very -
very long line.
```

  - you cannot do this in the real header files though!

- The format keywords are also optional

Example 1: image data

- The dataset called `image.dat` is a 2D image (512x256) consisting of greyscale values in the range 0 - 255
- The data is in ASCII format

```
0 10 0 15 200 255 ...
20 3 10 22 220 320 ...
...
```

- With no header information

Solution for example 1

```
# AVS field file
#
ndim = 2        # No. of computational dimensions
dim1 = 512
dim2 = 256
nspace = 2      # No. of coordinate dimensions
veclen = 1
data = integer
field = uniform
variable 1 file = image.dat filetype = ascii -
    skip = 0 offset = 0 stride = 1
```

Example 2: volume data

- The dataset called `volume.dat` consists of a 3D array of density values in a regular volume, 128x128x128
- The file has a line of header information
- The data

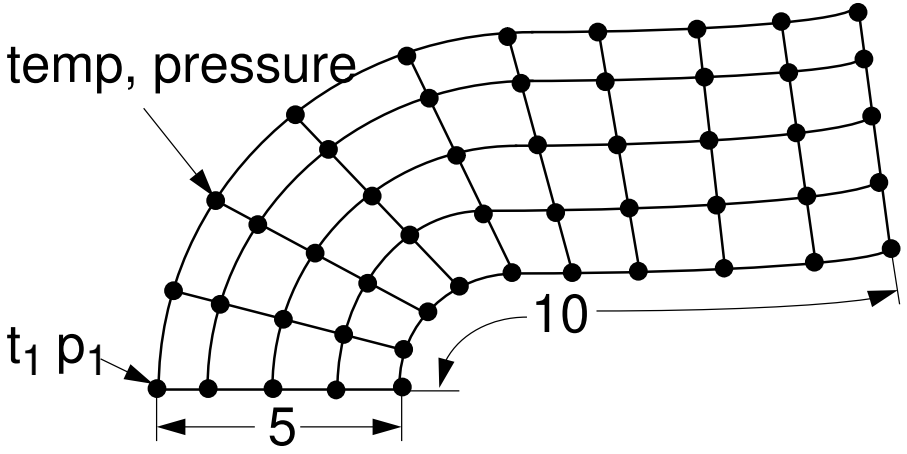
```
Patient: Steve 10/11/93
0 255 2 112 7 235...
5 236 6 123 8 236... ... ...
6 246 ...
```

- Try this one yourself...

Solution for example 2

```
# AVS field file
#
ndim = 3
dim1 = 128
dim2 = 128
dim3 = 128
nspace = 3
veclen = 1
data = integer
field = uniform
label = density
variable 1 file = volume.dat filetype = ascii -
    skip = 1 offset = 0 stride = 1
```

Example 3: irregular data



flow.dat

t_1 p_1 t_2 p_2 t_3 p_3 ...
...

coord.dat

x_1 y_1 x_2 y_2 x_3 y_3 ...
...

Solution to example 3

```
# AVS field file
#
ndim = 2
dim1 = 5
dim2 = 10
nspace = 2
veclen = 2
data = float
field = irregular
label = temperature pressure
variable 1 file = flow.dat filetype = ascii -
    skip = 0 offset = 0 stride = 2
variable 2 file = flow.dat filetype = ascii -
    skip = 0 offset = 1 stride = 2
coord 1 file = coord.dat filetype = ascii -
    skip = 0 offset = 0 stride = 2
coord 2 file = coord.dat filetype = ascii -
    skip = 0 offset = 1 stride = 2
```