FLD of Dreams

Or
"How to import arbitrary data into AVS/Express"
(And not get hurt)

Paul G. Lever
International AVS Centre & Manchester Visualization Centre

Introduction

- Overview of course
- Understanding data
- How to import data
- What visualization tasks can be applied
- Plenty of examples
- Exercises to try out and take-away

- Q&A Session this afternoon

Overview of Course

- Data
  - What information it contains
  - What you already know about it
  - What you need to think about
  - Get the meta-data
- Importing data
  - Method used depends on data
  - Picking the right approach
  - Information required by the “Reader”
  - How to provide that information

Overview of Course (2)

- Working with data
  - How does AVS/Express manage it
  - How does it store it internally
  - What modules can work with your data
  - What modules can't
- Visualization of data
  - Using appropriate techniques
  - Using multiple techniques
  - Effects of the data and the meta-data

Overview of Course (3)

- Content not presented in that order
- Will try and introduce new concepts at the same time in the four key areas:
  - Understanding Data and Meta-Data
  - Importing Data
  - Working with Data
  - Visualizing Data

Data

- The dream:
  - Import your data
  - Visualize your data
  - Understand your data
- The nightmare:
  - ASCII, binary; single file, multiple files
  - Block, interleave; scalar, vector, multivariate
  - Grids, nodes, cells; 1D, 2D, 3D or 4D
  - Uniform, rectilinear, (un)structured
  - Data types; headers; offset, skip, stride
  - Computational & Coordinate space; extents; labels
  - Limitations; and more besides….

Data (2)

- What is missing?
- Understanding of the data
  - Before importing it
  - And before visualizing it
- If you understand what goes in
- You will have a better understanding of:
  - What comes out
  - What's right
  - And what's wrong

Data (3)

- Knowledge of how to import it
  - Use the right approach
  - Provide the right meta-data
  - Minimum requirements
  - Support given by the “Reader” tool

Classification

- Not all data is the same
- Non-spatial data
  - Simple lists of items
  - No spatial relationship between items
  - Visualization techniques:
    - Bar charts
    - Histograms
    - Graphs
    - Pie Charts
  - Or spatial relationship not important
    - May be ignored
    - Not necessary for visualization task

Classification (2)

- Spatial Data
  - Is a spatial relationship between items
  - Each item has a location (coordinates)
    - Within a given coordinate system
    - E.g., Cartesian or Polar
  - May have more than one location
    - E.g., Position in file; array; 3D space
  - Data may be associated with each location
  - Location coordinates are also data!

Classification (3)

- Spatial Data (2)
  - Items have a relationship with other items
    - A connectivity between a set of items
  - Data may be associated with each relationship
    - A set of items have data assigned to them as a unit
  - Items may be in more than one relationship
  - Items may be organized
    - An underlying structure
    - E.g., a grid

Classification (4)

- Spatial Data (3)
  - Organization and Relationships affect:
    - The coordinates
    - The data at each coordinate
    - The connectivity
    - The data for each connectivity set
  - More specifically it affects:
    - The data that has to be provided
    - What data is implicit because of the given organization and relationships
    - What data must be explicitly provided because of the given organization and relationships

Tables and Fields

- All this can become very complex
  - The classification could be taken further
  - E.g., complexity of relational databases
- We are interested in AVS/Express
  - And importing “simpler” data
  - Not going to look at Database support
- More specifically we want to look at:
  - Reading tables (column data)
  - Reading fields (of types AVS/Express provides support for)

Tables

- Look at some simple column data
  - Describe it
  - Look at it, i.e., the contents of the file
  - Extrapolate what we know about it
    - Or what we should know about it
  - Look at the Read_Column_File tool
  - Examine what it requires
  - Match our meta-data and its parameters
  - Think about the visualization tasks
  - Try some visualization techniques
  - And see the results

Exercise: Population Data

- We have some population data
- The description of the data is:
  - It lists a bunch of towns on the Isle of Wight
  - And gives the population count for each town
  - Broken down into counts for males and females
  - And it includes the total count
  - The data is given in the file “town.txt”

Exercise: Population Data (2)

- Let's take a look at the data

The following figures for the Isle of Wight population are based on the official Government Census conducted in 1991.

Area	Males	Females	Total
Cowes_Castle	1715	1923	3638
Cowes_Central	1233	1281	2514
Cowes_Medina	1554	1642	3196
Cowes_Northwood	1777	1903	3680
East_Cowes	1901	1996	3897
Newport_Carisbrooke	2475	2689	5164
Newport_Central	814	925	1739
Newport_Mountjoy	1491	1732	3223
Newport_Pan	2067	2323	4390
Newport_Parkhurst	2562	1757	4319
Newport_Wootton_&_Fairlee	2939	3259	6198
Osborne	1394	1600	2994
Ryde_St_Johns	2340	2701	5041
Ryde_St_Helens	1860	2130	3990
Ryde_East	2654	3046	5700
Ryde_Ashey_&_Binstead	2782	2975	5757
Ryde_West	2657	3007	5664
Arreton_&_Newchurch	1728	1812	3540
Bembridge	1628	1939	3567
Brading	970	1107	2077
Brighstone_&_Shorwell	1132	1292	2424
Calbourne_&_Shalfleet	1178	1181	2359
Chale_&_Niton	1316	1449	2765
Freshwater	2438	2829	5267
Gatcombe_&_Godshill	1253	1286	2539
Lake	1997	2408	4405
Sandown	2477	2822	5299
Shanklin_North	1925	2246	4171
Shanklin_South	1831	2053	3884
Totland	1208	1449	2657
Ventnor	2770	3208	5978
Wroxall	801	855	1656
Yarmouth	410	475	885

Exercise: Population Data (3)

- What can we extrapolate?
  - It has two lines at the top that should be ignored as they are not part of the data
  - It then has a header line
    - Giving the column titles
    - They are [Area][Male][Female][Total]
    - It's on one line
    - Each title is separated by a TAB character
  - Then, on each subsequent line it gives:
    - A string for the town name
    - An integer number for the male population
    - An integer number for the female population
    - An integer number for the total population

Exercise: Population Data (4)

- More yet:
  - Those entries on each line are too separated by a TAB character
  - There are <N> lines, meaning <N> towns
    - Minus one for the header!
- Therefore:
  - Town name is in the 1st column
  - Male Population is in the 2nd column
  - Female population is in the 3rd column
  - And the total is in the 4th column
  - Hence we have four columns per line!

Exercise: Population Data (5)

- The Read_Column_File tool
  - Standard module in AVS/Express
  - Drag & Drop into network
  - Comes with its own User Interface
- What does it want from you?
  - A filename where the data can be found
  - How many lines should it skip?
  - Is there a header line it should read?
  - How many columns (can work it out itself)?
  - What is in each column (automatic)?

Exercise: Population Data (6)

- Matching meta-data to the parameters
  - It wants filename: “town.txt”
  - It wants the separator character: “TAB”
  - It wants number of lines to skip: “2”
  - Should it read a header line: “YES” (tick)
  - At this stage it will automatically look at the file and try to work things out for itself
    - It gets it right!
    - Should say there a 4 columns and you should be able to examine the settings for each column
    - It should have set the type for each column correctly

Exercise: Population Data (7)

- Visualization Techniques
  - View as a graph
    - Use the Annotation and Graphing Kit
    - Will need to extract the arrays from Read_Column_File
    - Example network includes this for you to try
  - Exercise
    - Very simple: just load the example application
    - Then set the Read_Column_File settings and press "Read File"
    - Look at the settings it has automatically derived from examining the data and play with settings to see what happens
    - Look at the graph generated by the application

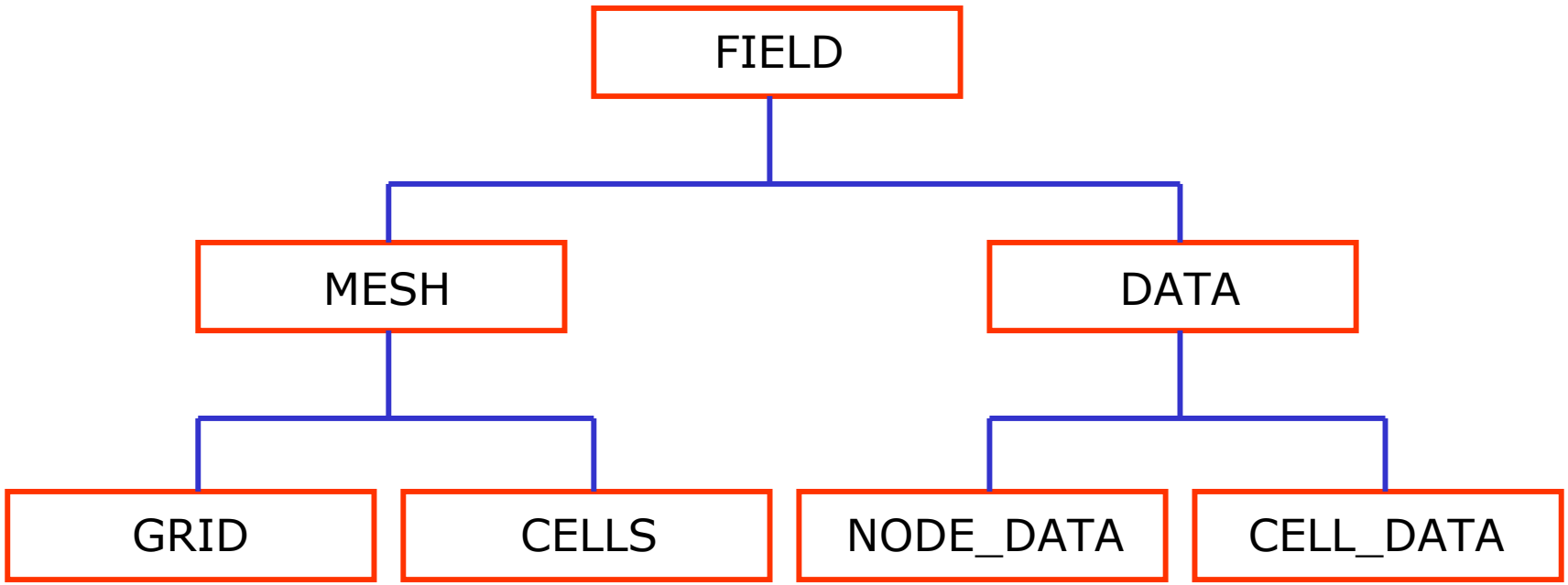
Exercise

- Exercise 3 in the workbook

Fields

- AVS/Express Field Terms
  - Each item is known as a “Node”
  - It has a “Coordinate”
  - All the coordinates together form a “Grid”
  - Data on that node is known as “Node Data”
  - A relationship/connectivity is a “Cell”
  - Data for that cell is “Cell Data”
  - A collection of cells (of one cell type) is a “Cell Set”
  - All the Cells Sets together and the Grid constitute a “Mesh”

Field Hierarchy



Field Hierarchy (2)

- Field Definitions
  - Mesh
    - "A geometric description that specifies where and how the data is located and organized in space"
  - Data
    - "Scalar or vector values defined at specific domain locations"
- Mesh Definitions
  - Grid
    - "Defines the location of nodes in space"
  - Cells
    - "Specify node (grid) connectivity"

Field Hierarchy (3)

- Data Definitions
  - Node_Data
    - “Data located on the nodes”
  - Cell_Data
    - “Data associated with the cells”

Field Types

- Grid & Mesh Organization
  - The “Grid” can be “Unstructured”
    - A collection of nodes with no obvious pattern
  - The “Grid” can be “Structured”
    - The nodes are arrayed in 1D, 2D or 3D grids
  - The “Mesh” can be “Unstructured”
    - Collection of of higher-order cells (e.g., triangles)
  - The “Mesh” can be “Structured”
    - Nodes are arrayed in 1D, 2D or 3D grids
    - If the mesh is structured the grid must be too
    - But if the grid is structured is doesn’t mean that the mesh has to be too

Field Types (2)

- Data Organization
  - Node Data is mapped to the Grid
    - Data is assigned to the nodes themselves on the provided coordinates
    - If the Grid is Unstructured, so is the Node Data
    - If the Grid is Structured, so is the Node Data
  - Cell Data is mapped to the Mesh
    - Data is assigned to each defined cell
    - If the Mesh is Unstructured, Cell Data is assigned to each cell that has been defined
    - If the Mesh is Structured, Cell Data is assigned to the implicit cells of the grid structure

Field Types (3)

- AVS/Express defines four Base Types
  - Based on the type of "Mesh"
  - And on the "computational to physical space" mapping that is used
- Computational Space
  - How the nodes (and the data) are organized
  - Data is computationally N-dimensional array
- Coordinate (Physical) Space
  - The physical space in which the field exists
  - The dimensionality of the coordinates
    - 1D (x), 2D (x,y) or 3D (x,y,z) per node

Field Types (4)

- The four Base Types are:
  - Unstructured
  - Structured: Irregular
    - Nodes are arrayed as 1D, 2D or 3D grid
    - There is no alignment of the nodes on the grid
    - Connectivity is implied as Lines, Quads or Hexahedrons respectively
  - Structured: Rectilinear
    - Nodes are arrayed as Irregular
    - There is some alignment of nodes
  - Structured: Uniform
    - Same but with more rigid alignment of nodes

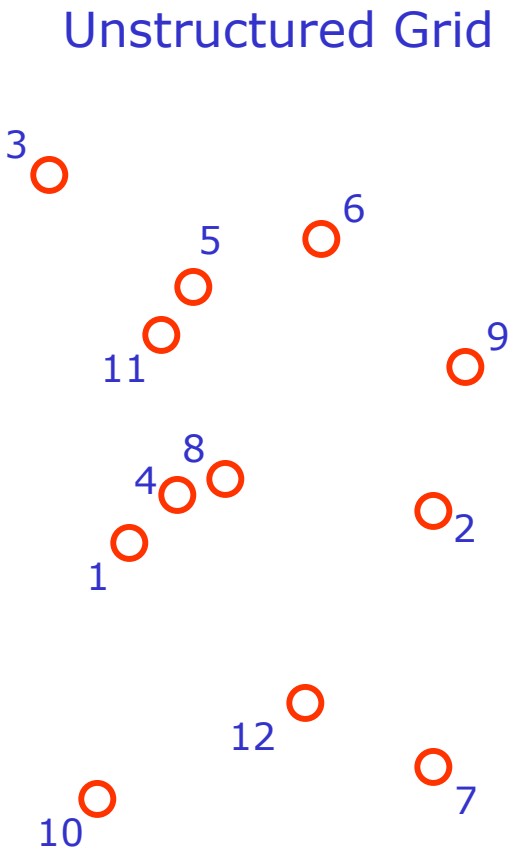
Field Types (5)

- Computational to Coordinate Mapping
  - Unstructured Grid is just a list of nodes
    - Is technically 1D but there is no connectivity implied
  - Structured means data is in an array
    - 1D array has implied connectivity
    - Each subsequent node is connected to the previous
    - 2D and 3D arrays have implied connectivity
  - E.g., an image is a 2D array of pixels
    - You would not normally treat an image as a 1D list of values as that removes their inherent structure
  - E.g., a volume is a 3D array of voxels

Field Types (6)

- Computational to Coordinate Mapping
  - Do not have to match
    - Coordinate space is independent of computational
  - it is common for computational space to be less than or equal to coordinate space
    - It is unusual to have 3D computationally arrayed data which is only mapped to a 2D coordinate world for example but it is possible
  - Example
    - Latitude and Longitude are 2D coordinates (lat,long)
    - Mapped to a sphere (exists in a 3D world) requires 3D coordinates (x,y,z)
    - This is a 2D to 3D mapping

Unstructured Grid



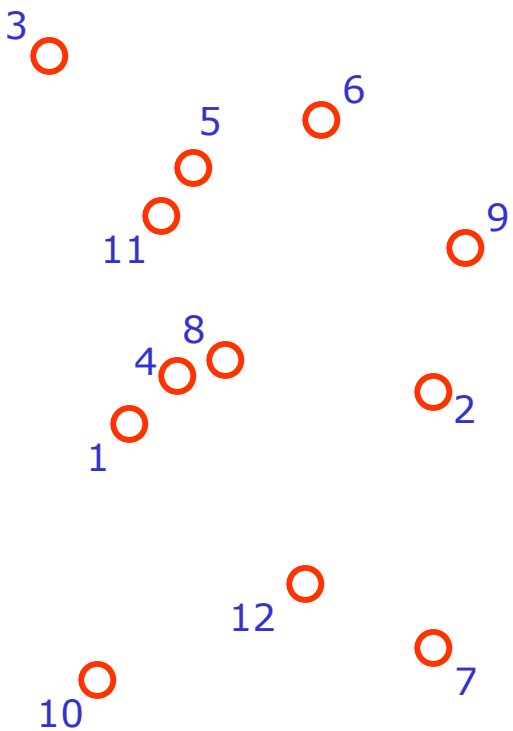
Just a list of nodes

No implicit connectivity between the nodes

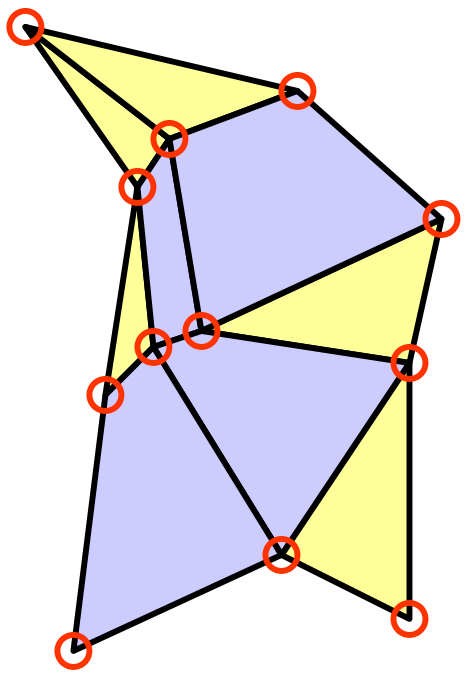
Must be explicitly given

Unstructured Grid & Mesh

Unstructured Grid



Unstructured Mesh
(2 Cell Types: Tri and Quad)



Structured Fields

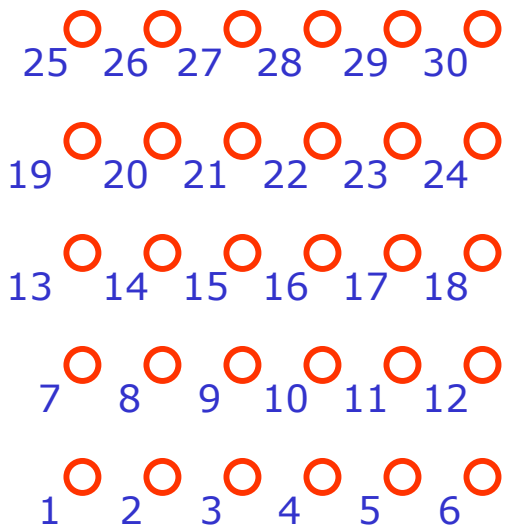
- Unstructured Fields
  - Have the most complex requirements
  - Will return to definition later
- Structured Fields
  - Much simpler
  - Will examine them now
  - Show how to import those with the AVS "Read Field" tool
  - Show how to visualize them

2D Computational & Uniform Grid

Structured Computational Grid
2D: (6 x 5 nodes)
The structure implies the connectivity between the nodes

25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6

Structured Uniform Grid
2D: (6 x 5 nodes)

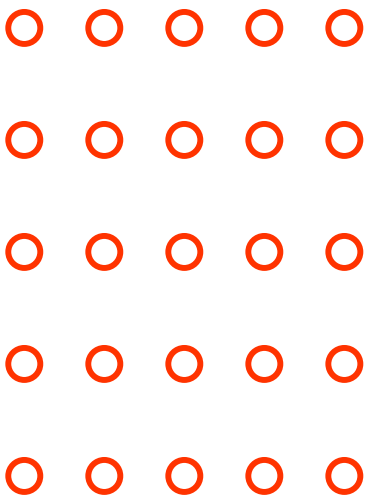


NB. This computational 2D grid is the same for all 2D structured Grids & Meshes

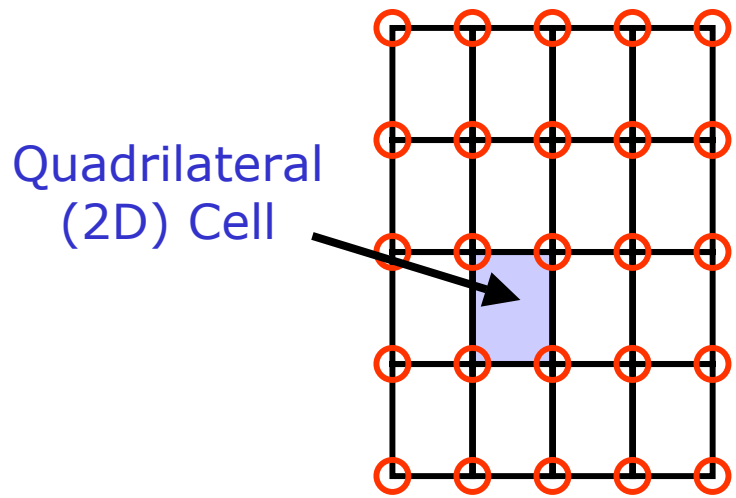
NB. 3D computational grids continue this in the next plane

2D Uniform Grid & Mesh

Uniform (Structured) Grid
2D: (5 x 5 nodes)



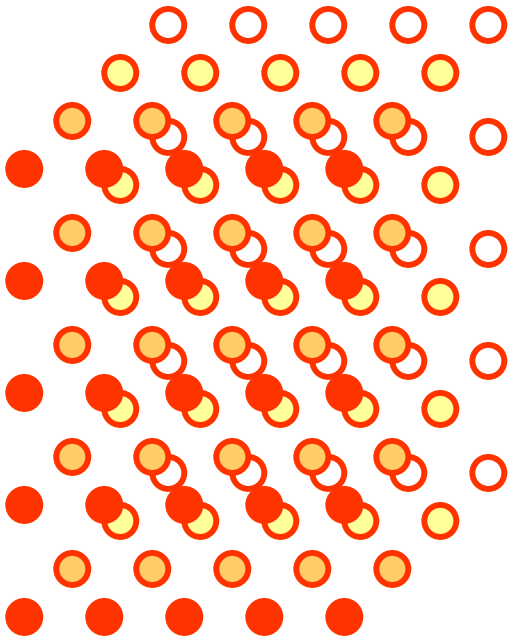
Uniform (Structured) Mesh
2D: (4 x 4 Quads)

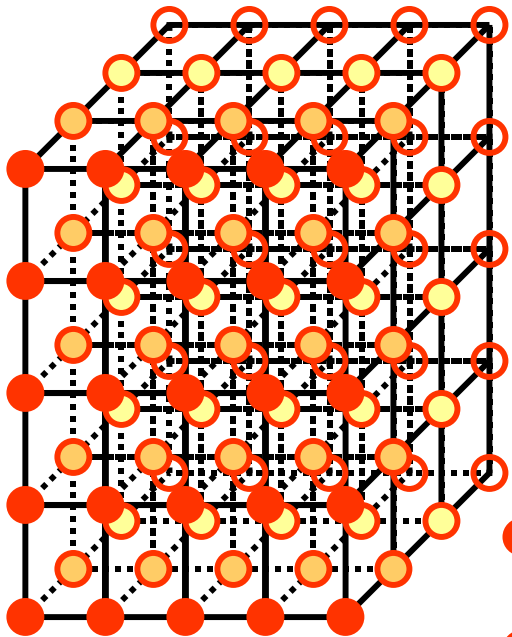


NB. Even spacing but
may be different per axis

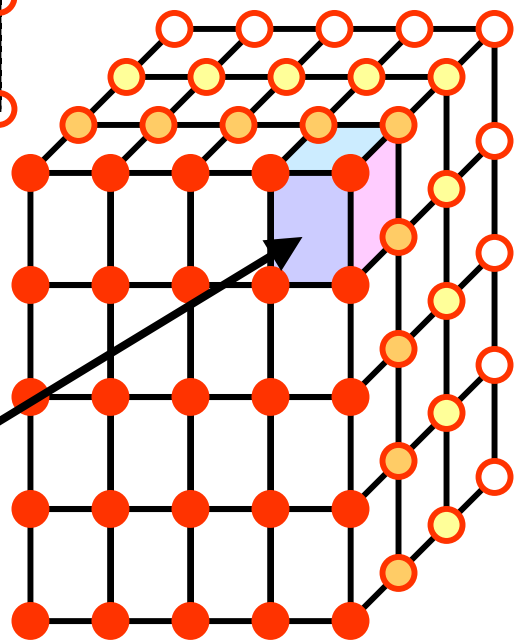
3D Uniform Grid & Mesh

Uniform (Structured) Grid
3D: (5 x 5 x 4 nodes)





Uniform
(Structured)
Mesh
3D: (4 x 4 x 3
Hex's)



Hexahedral
(3D) Cell

Meta-Data for Uniform Field

- Uniform Grid & Mesh implies:
  - All nodes on a grid, uniformly spaced
    - No missing nodes and no extra nodes
  - 2D grid must have A x B nodes
  - 3D grid must have A x B x C nodes
    - We have N nodes, the product of A and B (and C)
    - We need N coordinates
  - 2D connectivity forms a “Quadrilateral Mesh”
  - 3D connectivity forms a “Hexahedral Mesh”
  - May be data on each node
    - May need N data values
  - May be data for each cell
    - May need (A-1) x (B-1) x (C-1) data values

Meta-Data for Uniform Field (2)

- What information do we need?
  - What is the computational space?
    - Need to know if it's 1D, 2D or 3D arrayed data
  - What are the array dimensions?
  - What is the coordinate space?
    - Need to know if the physical coordinates are positioned in 1D, 2D or 3D space
  - What are the coordinates?
    - Need N (product of array dimensions) n-space coordinates

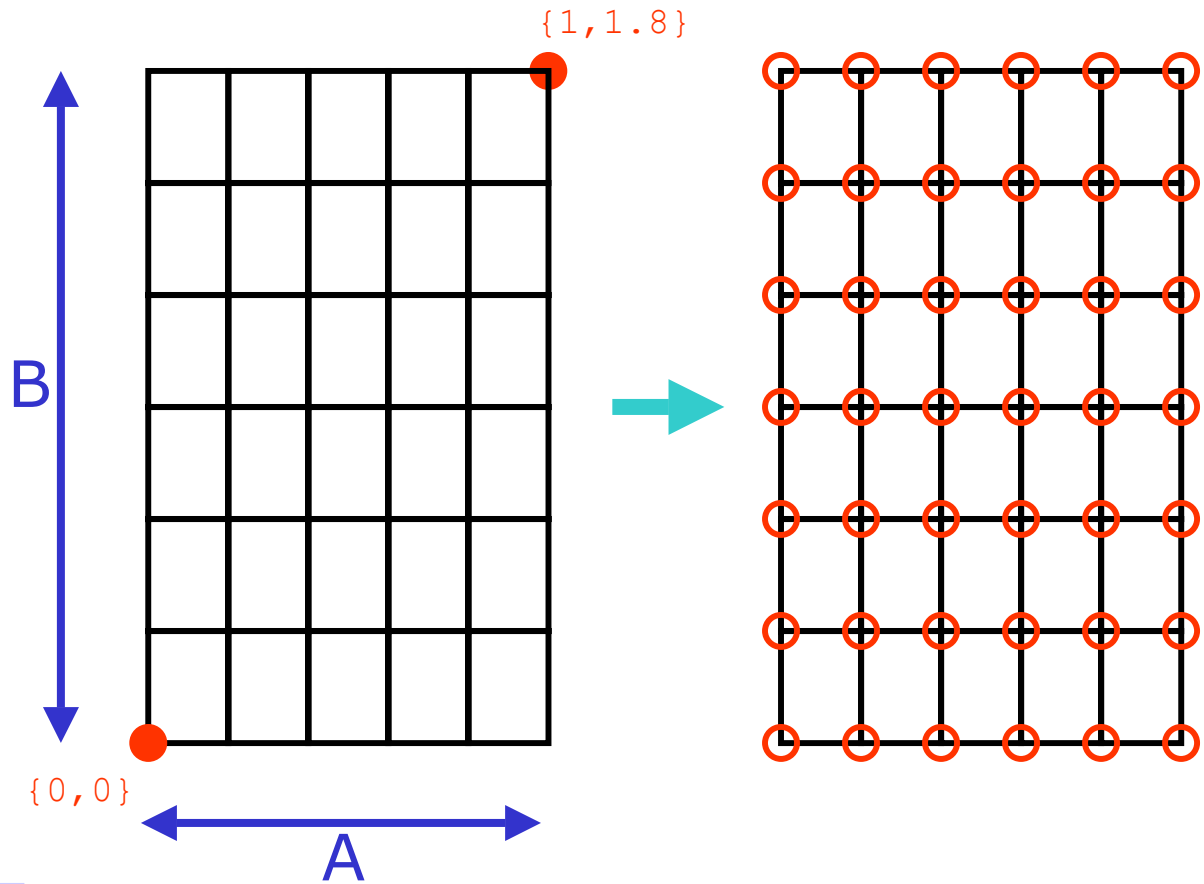
Meta-Data for Uniform Field (3)

- What information do we need?
  - What data do we have?
    - Need to know if there is Node Data and/or Cell Data
    - NB. AVS/Express "Read Field" tool does not support Cell Data
  - How many data components are there?
  - Are they scalar or vector?
  - What primitive type are they?
    - Single- or Double- Precision Floating Point, Integers, Shorts, or Bytes
  - Any labels or units defined for each data component?

Meta-Data for Uniform Field (4)

- Could this be simplified?
  - Not all the coordinates are needed
    - There is “uniform” spacing
    - We know how many grid points there are in each axis, A and B (and C)
  - We only need to specify two coordinates
    - Two diagonally opposite coordinates
    - The minimum and maximum extents coordinates
    - Both n-D coordinates for the n-D space
  - All other coordinates can be derived
    - It is common to not even specify the extents
    - By default the extents will be set to the array dimensions, e.g, 5 x 5 grid {0,0} to {4,4}

2D Uniform Mesh Simplified



Meta-Data:

- Type: Uniform
- Comp: 2D
- Dims: [A][B]
- Space: 2D
- Min = {0,0}
- Max = {1,1.8}
- Data[A][B] = {…..}

Derived:

- N = A x B
- Mesh = "2D Quad"
- Coords[A][B][2] = {{0.0,0.0}, {0.2,0.0}, {0.4,0.0}, {0.6,0.0}, {0.8,0.0}, {1.0,0.0}, {0.0,0.3}, {0.2,0.3}, {0.4,0.3}, {0.6,0.3}, …. {1.0, 1.8}}

Uniform Field Data

- What data do we have?
  - How many components?
  - And are they scalar or vector?
- Data must be organized on the same grid as the coordinates
  - Order of the data must follow the order that the nodes are defined in the computational array
    - I.e., Node[1] has Data[1]

Read Field Tool

- AVS/Express has a Field reader
- Meta-Data is given in a FLD file
  - It is an ASCII header file that defines a field
  - It has strict requirements about the information or meta-data it needs
    - Information is given on a line-by-line basis
    - Each line has a keyword and a value
    - Some keywords are mandatory
    - Some are optional
    - Some are required depending on other keyword values
    - Some keywords have sub-keywords and values
  - This description should match your data

Field File Keywords

```
# AVS field file
ndim = [1|2|3|…]
dim1 = [A]
dim2 = [B]
nspace = [1|2|3]
veclen = [1-N]
data = [byte|float|integer]
field = [uniform|rectilinear|irregular]
{labels = Name Name Name …}
{units = Name Name Name …}

variable 1 file=[filename] filetype=[binary|ascii] {skip=[?]} \
  {offset=[?]} {stride=[?]}
{variable 2 …}
{variable [VECLEN] …}

{coord 1 file=[filename] filetype=[binary|ascii] {skip=[?]} \
  {offset=[?]} {stride=[?]}
{coord [NSPACE] …}
```

IAC manchester

Field File Keywords (2)

- [] denote where values have to be given
  - The [] themselves are not required
- {} denote optional or dependent keywords
  - Following list will detail which
- \ denotes continuation of line
  - All keywords, sub-keywords and values must be on a single line
  - The \ character is for display purposes only and should not be present in the Field File itself
- First line mandatory
  - Comment tells AVS it's a Field File

Field File Keywords (3)

- ndim
  - Computational Space
  - Matches the dimensionality of the array
  - Usually 1,2 or 3
- dim1, dim2 … dimN
  - Array dimensions
  - Must be one entry for each Computational Space defined in ndim
- nspace
  - Coordinate Space
  - The physical space that the coordinates exist in
  - Usually 1,2 or 3

Field File Keywords (4)

- veclen
  - Number of data elements
  - This is the total of all elements for all components
    - E.g., two scalar and a 3 element vector will result in a total of 5 elements
    - Components, i.e., the two scalar and the one vector component can be extracted later inside AVS/Express
- data
  - The type of the data elements
  - Can be “float”, “integer”, “double”, “byte”
    - All data components must be the same type

Field File Keywords (5)

- field
  - Denotes the structure of the field
    - AVS Field File only supports structured meshes
    - Can be either “uniform”, “rectilinear” or “irregular”
- label
  - Optional label names for the elements
  - Must be same number of labels as elements
  - White-space separates each label
  - Vector component has a label for each element
    - E.g., wind_u wind_v wind_w

Field File Keywords (6)

- units
  - Optional unit (label) name for each element
  - Must be same number of units as elements
  - White-space separates each label
  - Vector component has a label for each element
    - E.g., cms cms cms

Field File Keywords (7)

- variable
  - One variable line per data element
  - Must be veclen variable lines
  - This is information about how to get the data for one element
  - file
    - Sub-keyword that specifies the filename of where the data can be found
    - Each variable defined can each have a different file or they can share any number of them
    - This sub-keyword can be optional if the data is in the FLD file itself; this approach is best avoided

Field File Keywords (8)

- variable
  - filetype
    - Denotes whether the data file is in binary or ascii format
    - This will affect how the data is read and how the other sub-keywords of the variable line are treated
    - The file type value can only be either “ascii” or “binary”
  - skip
    - For ascii files this denotes how many lines should be skipped at the start of the file (to avoid headers) to reach the start of the data
    - For binary files this denotes how many bytes should be ignored at the start of the file

Field File Keywords (9)

- variable
  - offset
    - As more than one data element may be held within one data file, each variable must be moved to the first data value for that element
    - Data may be interleaved within a file
    - E.g., two elements will alternate: A0, B0, A1, B1, …
    - The skip setting will point to the first value: A0
    - The variable setting for element B will need an offset of 1 to point to the correct first value it needs: B0
    - Data may be in blocks within a file
    - E.g., one element first: A0, A1, A2 … AN, B0, B1 … BN
    - To reach the first B value skip will have to be set to N
    - To reach a third element C skip will have to be 2N

Field File Keywords (10)

- variable
  - stride
    - Again for multiple elements within one file, the variable setting needs to know how to reach subsequent values
    - In the example of two interleaved elements, to reach the A1 value and A2 and so on, we must jump over the B values inbetween – so stride must be set to 2
    - For files where there is no interleaved data the stride value is 1 which is the default setting if not specified

Field File Keywords (11)

- coord
  - Same as variable but gets access to variable elements, one for each of n-space setting
    - 3D n-space requires 3D coordinate, hence 3 coord elements
  - May share file with data
  - May be interleaved with data or blocked or combination of both
  - Coordinate data is always of type “float”

Uniform Field & Field File

- How to import a uniform field:
  - 2D uniform mesh
  - With just scalar data
  - With multivariate/vector data
  - Extend to 3D uniform mesh
  - Get data from one or multiple files
  - Using interleaved and blocked data
  - Using ASCII and Binary data

2D Scalar ASCII Data File

Description of Data:

We have a data file called "mri.dat" which contains a 2D slice of data from an MRI scanner. There are 512 x 512 nodes in the 2D grid, each node has a single byte of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.

mri.dat: (sample)

```
124 76 54 124 127 134 156 161 190
173 44 44 3 0 0 0 0 0 0 34 244 …
```

```
# AVS field file
ndim = 2
dim1 = 512
dim2 = 512
nspace = 2
veclen = 1
data = byte
field = uniform

variable 1 file=mri.dat
  filetype=ascii
```

This will allow the MRI data to be read successfully

Binary Data File

Description of Data:

We have a data file called "mri_bin.dat" which contains a 2D slice of data from an MRI scanner. There are 512 x 512 nodes in the 2D grid, each node has a single byte of data associated with it. The file simply contains the data; no header information. The data is written in binary format.

mri_bin.dat: (sample)

```
0A1F16575E5F727C00FFDC8F1A393BBB
BF…
```

NB. Viewed here as Hexadecimal

```
# AVS field file
ndim = 2
dim1 = 512
dim2 = 512
nspace = 2
veclen = 1
data = byte
field = uniform

variable 1 file=mri_bin.dat
  filetype=binary
```

This will allow the binary MRI data to be read successfully

2D Vector Data File

Description of Data:

We have two data filea called "wind_u.dat" and "wind_v.dat" which each contain a 2D slice of data from a climate simulation. There are 128 x 128 nodes in the 2D grid, each node has two float values representing a 2D vector. The files contain data for one element of the 2D vector, with one line of header information. The data is written in ASCII format.

wind_u.dat: (sample)

```
Wind U-component

1.2 1.3 1.4 2.1 0.2 0.2 0.3 0.9
-1.4 -1.5 -1.7 …
```

NB. wind_v file is similar

```
# AVS field file
ndim = 2
dim1 = 128
dim2 = 128
nspace = 2
veclen = 2
data = float
field = uniform
Labels = wind_u wind_v
variable 1 file=wind_u.dat
  filetype=ascii skip=1
variable 2 file=wind_v.dat
  filetype=ascii skip=1
```

Image Data File (PPM)

Description of Data:

We have a data file called "survivor.ppm" which contains a 2D image. There are 452 x 304 pixels nodes in the 2D grid, each node has a three bytes of data associated with it, corresponding to Red, Green and Blue values. The file contains the data and 4 lines of header information. The data is written in binary format.

survivor.ppm: (sample)

```
P6
Creator: XV Version 3.10a
452 304
255
rjfjr94940s9d09ds9%&*&Ygihy&jj…
```

```
# AVS field file
ndim = 2
dim1 = 452
dim2 = 304
nspace = 2
veclen = 3
data = byte
field = uniform
Labels = R G B
variable 1 file=survivor.ppm
  filetype=binary skip=41 stride=3
variable 2 file=survivor.ppm
  filetype=binary skip=42 stride=3
variable 3 file=survivor.ppm
  filetype=binary skip=43 stride=3
```

NB. Header consists of 41 bytes

3D Vector Data File

Description of Data:

We have three data files called "wind3_u.dat" , "wind3_v.dat" and "wind3_w.dat", which each contain a 3D volume of data from a climate simulation. There are 128 x 128 x 128 nodes in the 3D grid, each node has three float values representing a 3D vector. The files contain data for one element of the vector with one line of header information. The data is written in ASCII format.

wind_w.dat: (sample)

```
Wind w-component

1.2 1.3 1.4 2.1 0.2 0.2 0.3 0.9
-1.4 -1.5 -1.7 …
```

```
# AVS field file
ndim = 3
dim1 = 128
dim2 = 128
dim3 = 128
nspace = 3
veclen = 3
data = float
field = uniform
Labels = wind_u wind_v wind_w
variable 1 file=wind3_u.dat
  filetype=ascii skip=1
variable 2 file=wind3_v.dat
  filetype=ascii skip=1
variable 3 file=wind3_w.dat
  filetype=ascii skip=1
```

3D Vector Single ASCII Data File

Description of Data:

We have one data file called "wind3_uvw.dat", which contains a 3D volume of data from a climate simulation. There are 128 x 128 x 128 nodes in the 3D grid, each node has three float values representing the 3D vector. The file contains data for the vectors stored as blocks: all u elements, then all v elements, then all w elements. There is one line of header information. The data is written in ASCII format and there are 8 values per line.

wind3_uvw.dat: (abstract)

```
Wind uvw-components
u0 u1 u2 u3 u4 u5 u6 u7 (EOL)
u8 u9 …
v0 v1 v2 v3 v4 v5 v6 v7
… w0 w1 … wN
```

```
# AVS field file
ndim = 3
dim1 = 128
dim2 = 128
dim3 = 128
nspace = 3
veclen = 3
data = float
field = uniform
Labels = wind_u wind_v wind_w
variable 1 file=wind3_uvw.dat
  filetype=ascii skip=1
variable 2 file=wind3_uvw.dat
  filetype=ascii skip=262143
variable 3 file=wind3_uvw.dat
  filetype=ascii skip=524289
```

NB. Each block occupies 262144 lines.

3D Vector Interleaved Data File

Description of Data:

We have one data file called "wind3i_uvw.dat", which contains a 3D volume of data from a climate simulation. There are 128 x 128 x 128 nodes in the 3D grid, each node has three float values representing the 3D vector. The file contains data for the vectors stored interleaved with each other: u,v,w then next u,v,w. There is one line of header information. The data is written in ASCII format and there are 8 values per line.

wind3i_uvw.dat: (abstract)

```
Wind uvw-components
u0 v0 w0 u1 v1 w1 u2 v2
w2 u3 v3 w3 …
```

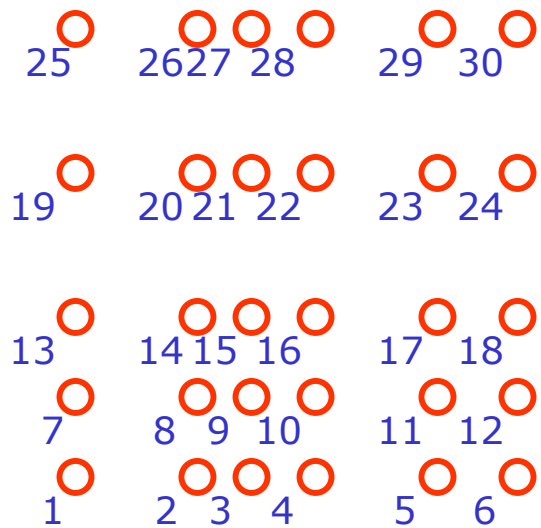
```
# AVS field file
ndim = 3
dim1 = 128
dim2 = 128
dim3 = 128
nspace = 3
veclen = 3
data = float
field = uniform
Labels = wind_u wind_v wind_w
variable 1 file=wind3i_uvw.dat
  filetype=ascii skip=1 offset=0
variable 2 file=wind3i_uvw.dat
  filetype=ascii skip=1 offset=1
variable 3 file=wind3i_uvw.dat
  filetype=ascii skip=1 offset=2
```

2D Computational & Rectilinear Grid

Structured Computational Grid
2D: (6 x 5 nodes)
The structure implies the connectivity between the nodes

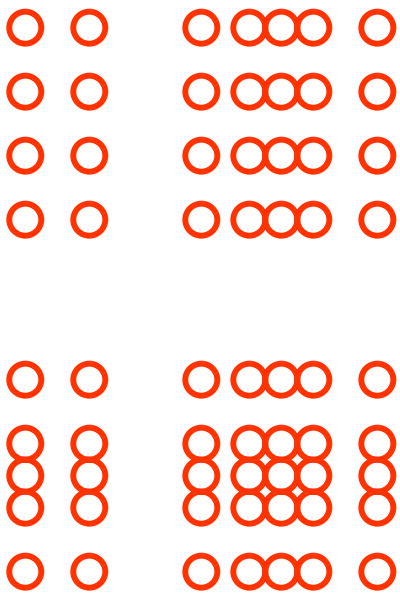
25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6

Structured Rectilinear Grid
2D: (6 x 5 nodes)

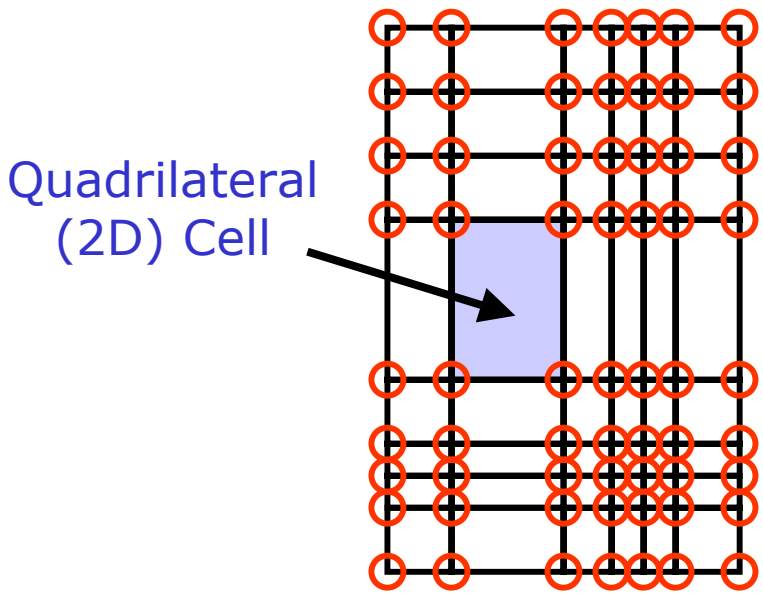


2D Rectilinear Grid & Mesh

Rectilinear (Structured) Grid
2D: (7 x 9 nodes)



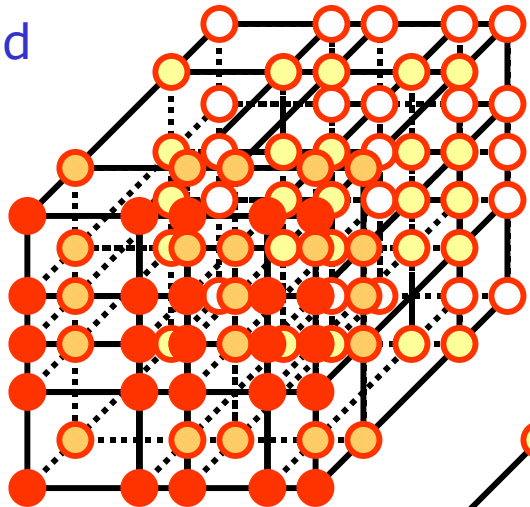
Rectilinear (Structured) Mesh
2D: (6 x 8 Quads)



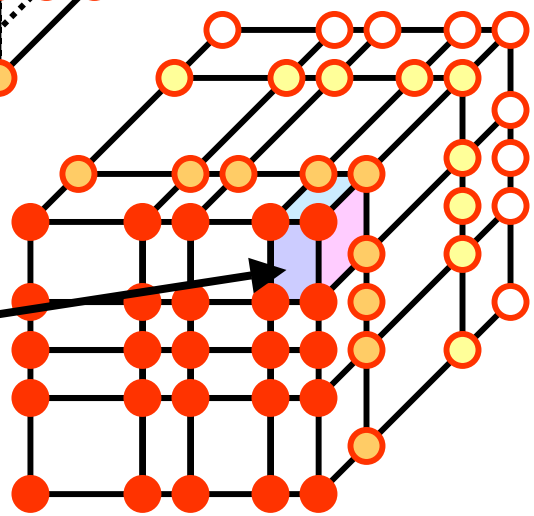
NB. Uneven spacing but axis aligned
Spacing may be different per axis

3D Rectilinear Grid & Mesh

Rectilinear (Structured) Grid
3D: (5 x 5 x 4 nodes)



Rectilinear (Structured) Mesh
3D: (4 x 4 x 3 Hex's)



Hexahedral (3D) Cell

Meta-Data for Rectilinear Field

- Rectilinear Grid & Mesh implies:
  - All nodes on a grid, uneven spacing but axis-aligned
  - 2D grid must have A x B nodes
  - 3D grid must have A x B x C nodes
    - We have N nodes, the product of A and B (and C)
    - We need N coordinates
  - 2D connectivity forms a “Quadrilateral Mesh”
  - 3D connectivity forms a “Hexahedral Mesh”
  - May be data on each node
    - May need N data values
  - May be data for each cell
    - May need (A-1) x (B-1) x (C-1) data values

Meta-Data for Rectilinear Field (2)

- What information do we need?
  - What is the computational space?
    - Need to know if it's 1D, 2D or 3D arrayed data
  - What are the array dimensions?
  - What is the coordinate space?
    - Need to know if the physical coordinates are positioned in 1D, 2D or 3D space
  - What are the coordinates?
    - Need N (product of array dimensions) n-space coordinates

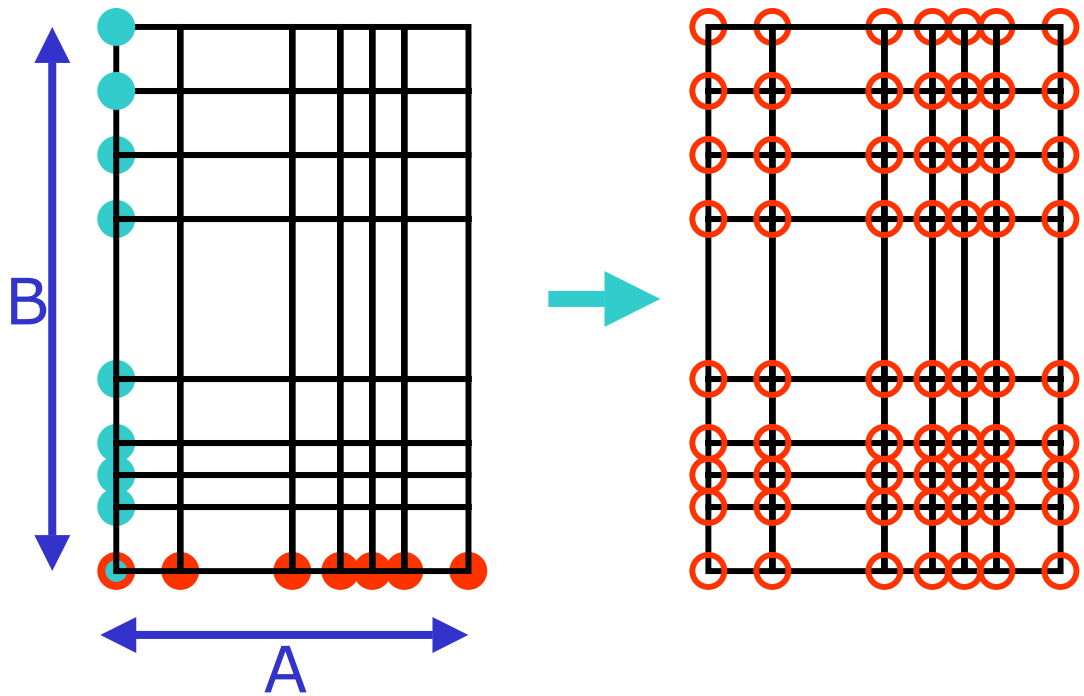
Meta-Data for Rectilinear Field (3)

- What information do we need?
  - What data do we have?
    - Need to know if there is Node Data and/or Cell Data
    - NB. AVS/Express "Read Field" tool does not support Cell Data
  - How many data components are there?
  - Are they scalar or vector?
  - What primitive type are they?
    - Single- or Double- Precision Floating Point, Integers, Shorts, or Bytes
  - Any labels or units defined for each data component?

Meta-Data for Rectilinear Field (4)

- Could this be simplified?
  - Not all the coordinates are needed
    - There is “rectilinear” spacing
    - We know how many grid points there are in each axis, A and B (and C)
  - We only need to specify A+B ordinates
    - Coordinate values along each defined axis
    - Define A x-axis ordinates (just x-element)
    - Define B y-axis ordinates (just y-element)
    - Both 1D ordinates for the 1D axis
    - Combined to make 2D space coordinates
  - All other coordinates can be derived

Rectilinear Grid



Information:

- 2D space
- A x B nodes
- X-axis Coords[A][2] = {{0.0,0.0}, {0.2,0.0}, {0.55,0.0}, …}
- Y-axis Coords[B][2] = {0.0,0.0}, {0.0,0.2}, {0.0,0.3,}…}
- Data[A][B] = {…..}
- Connectivity = "2D Mesh"

Derived:

- Coords[A][B][2] = {{0.0,0.0}, {0.2,0.0}, {0.55,0.0}, …}
- Min Extents = {…}
- Max Extents = {…}

Rectilinear Field Data

- What data do we have?
  - How many components?
  - And are they scalar or vector?
- Data must be organized on the same grid as the coordinates
  - Order of the data must follow the order that the nodes are defined in the computational array
    - I.e., Node[1] has Data[1]

Rectilinear Field & Field File

- How to import a rectilinear field
  - 2D rectilinear mesh
  - 3D rectilinear mesh
  - Get coordinates from one or multiple files
  - Using interleaved and block coordinates

2D Rectilinear Scalar ASCII Data File

Description of Data:

We have a data file called "fin.dat" which contains a 2D slice of data from a Fluid Flow simulation. There are 12 x 15 nodes in the 2D grid, each node has a single byte of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.
There are also two coordinate files called "fin.x" and "fin.y" which contain a set of rectilinear coordinates for two axes of the 2D grid. Both just contain data in ASCII format. "fin.x" has 12 values. "fin.y" has 15 values. Coordinate values are always of type "Float".

fin.x: (sample)

```
0.0 1.0 2.0 6.0 6.5 7.0 8.0 9.0
13.0 13.5 14.0 15.0
```

```
# AVS field file
ndim = 2
dim1 = 12
dim2 = 15
nspace = 2
veclen = 1
data = byte
field = rectilinear

variable 1 file=fin.dat
  filetype=ascii

coord 1 file=fin.x filetype=ascii
coord 2 file=fin.y filetype=ascii
```

Variable requirements have not changed. Now have coord keywords to read Rectinlinear grid coordinates.

2D Rectilinear Single Coordinate File

Description of Data:

We have a data file called "fin.dat" which contains a 2D slice of data from a Fluid Flow simulation. There are 12 x 15 nodes in the 2D grid, each node has a single byte of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.
There is one coordinate file called "fin.xy" which contains both sets of rectilinear coordinates for the two axes of the 2D grid. All x coordinates are listed first, then all y coordinates. File is in ASCII format. Coordinates are "Floats".

fin.xy: (sample)

```
0.0 1.0 2.0 6.0 6.5 7.0 8.0 9.0
13.0 13.5 14.0 15.0 -2.0 -1.0 -
0.5 0.0 0.2 …
```

```
# AVS field file
ndim = 2
dim1 = 12
dim2 = 15
nspace = 2
veclen = 1
data = byte
field = rectilinear

variable 1 file=fin.dat
  filetype=ascii

coord 1 file=fin.xy filetype=ascii
coord 2 file=fin.xy filetype=ascii
  offset=12
```

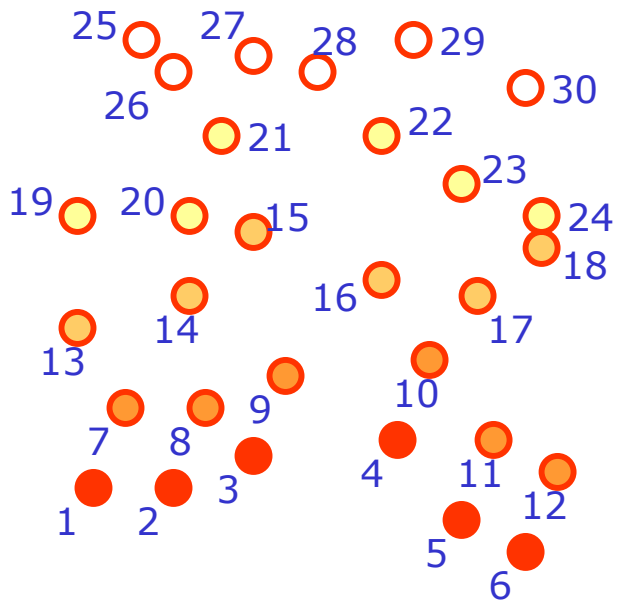
Not always possible to interleave rectilinear coordinates as they may not have same dimension.

2D Computational & Irregular Grid

Structured Computational Grid
2D: (6 x 5 nodes)
The structure implies the connectivity between the nodes

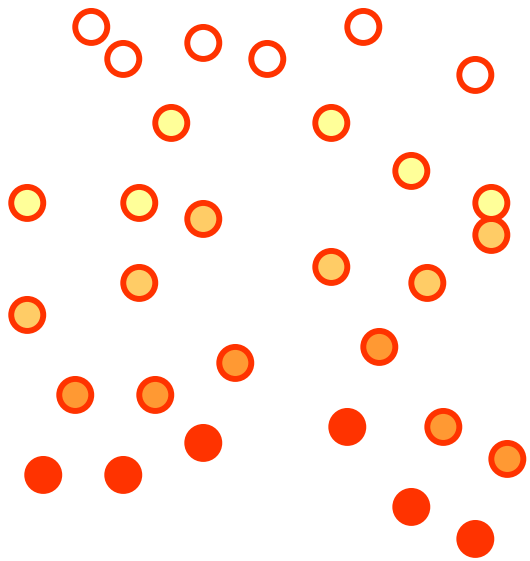
25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6

Structured Irregular Grid
2D: (6 x 5 nodes)

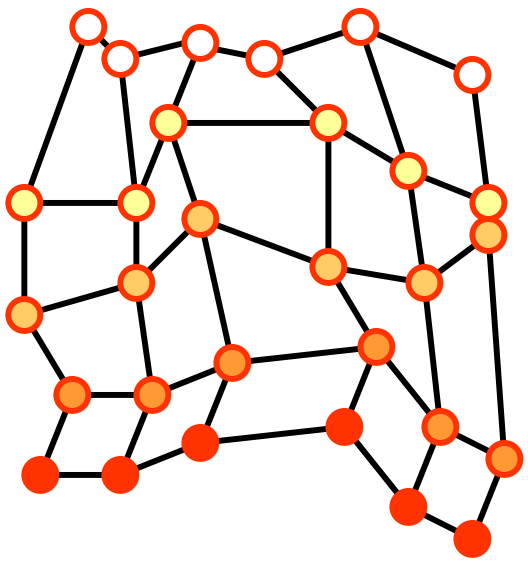


2D Irregular Grid & Mesh

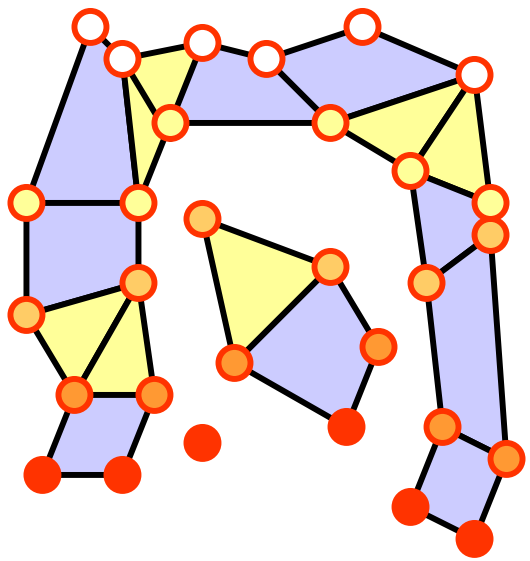
Structured Irregular Grid
2D: (6 x 5 nodes)



Structured Mesh
2D: (5 x 4 Quads)



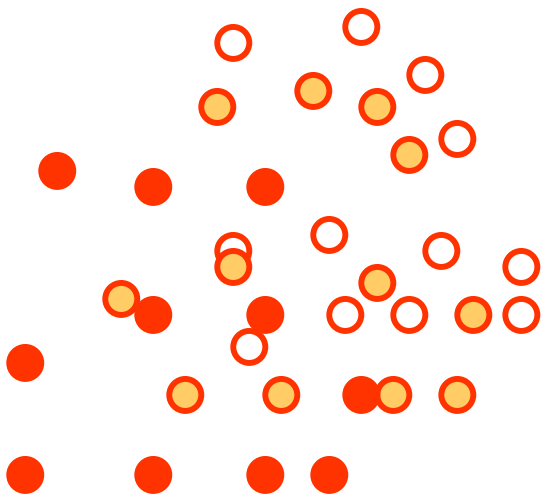
Unstructured Mesh
(Tri and Quad cells)



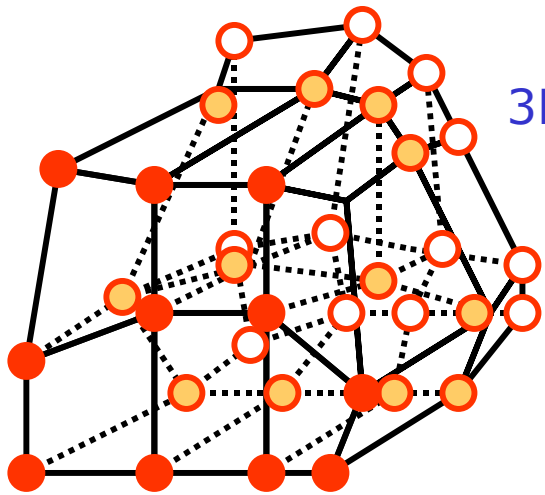
NB. Unstructured allows disjoint polygons and not all nodes are necessarily used

3D Irregular Grid & Mesh

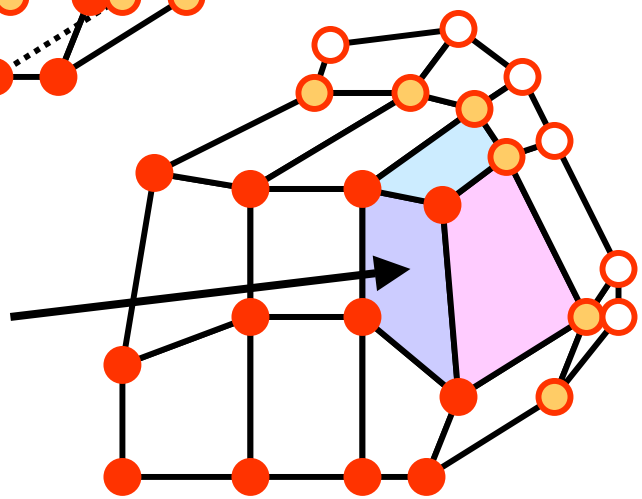
Structured Irregular Grid
3D: (4 x 3 x 3 nodes)



Structured Mesh
3D: (3 x 2 x 2 Hex's)

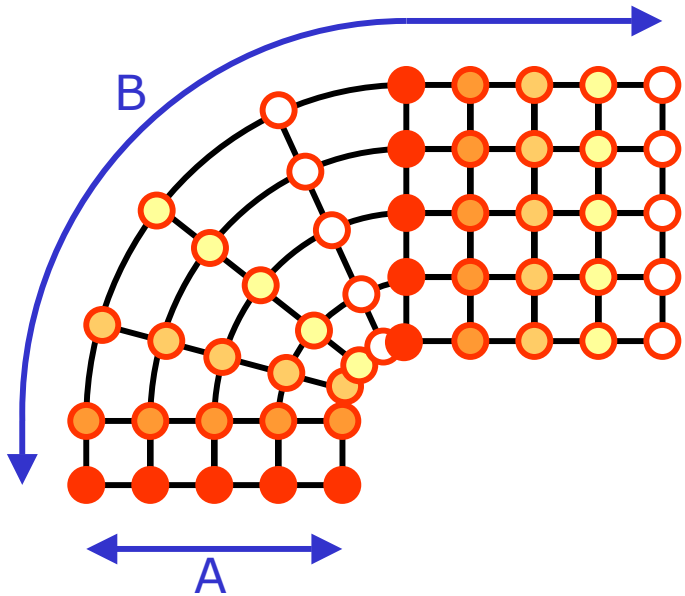


Hexahedral
(3D) Cell

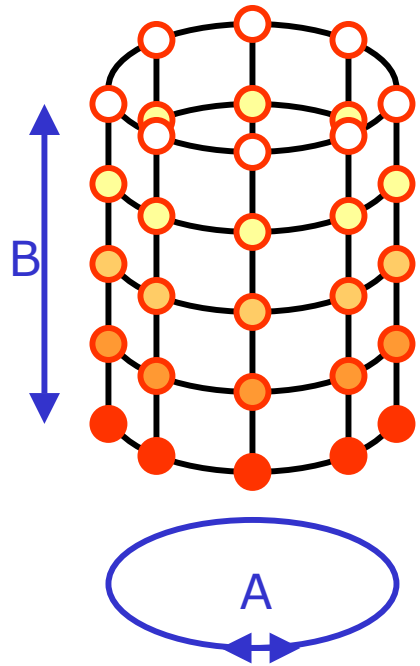


Other Irregular Grids

2D Grid, 2D Coordinates



2D Grid, 3D Coordinates



Meta-Data for Irregular Field

- Irregular Grid & Mesh implies:
  - All nodes on a grid, uneven spacing
  - 2D grid must have A x B nodes
  - 3D grid must have A x B x C nodes
    - We have N nodes, the product of A and B (and C)
    - We need N coordinates
  - 2D connectivity forms a “Quadrilateral Mesh”
  - 3D connectivity forms a “Hexahedral Mesh”
  - May be data on each node
    - May need N data values
  - May be data for each cell
    - May need (A-1) x (B-1) x (C-1) data values

Meta-Data for Irregular Field (2)

- What information do we need?
  - What is the computational space?
    - Need to know if it's 1D, 2D or 3D arrayed data
  - What are the array dimensions?
  - What is the coordinate space?
    - Need to know if the physical coordinates are positioned in 1D, 2D or 3D space
  - What are the coordinates?
    - Need N (product of array dimensions) n-space coordinates

Meta-Data for Irregular Field (3)

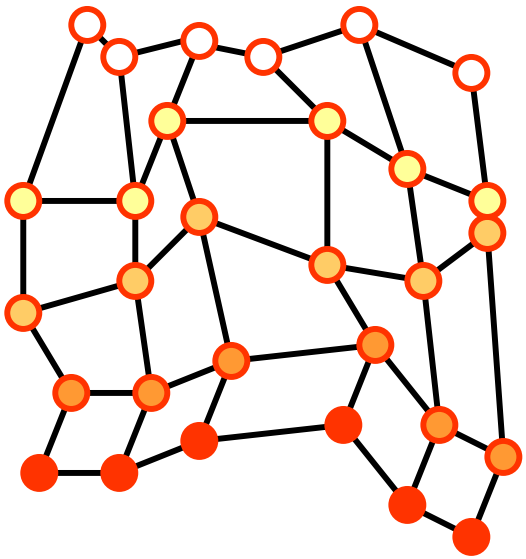
- What information do we need?
  - What data do we have?
    - Need to know if there is Node Data and/or Cell Data
    - NB. AVS/Express "Read Field" tool does not support Cell Data
  - How many data components are there?
  - Are they scalar or vector?
  - What primitive type are they?
    - Single- or Double- Precision Floating Point, Integers, Shorts, or Bytes
  - Any labels or units defined for each data component?

Meta-Data for Irregular Field (4)

- Could this be simplified?
  - No, all the coordinates are needed
    - There is “irregular” spacing
    - We know how many grid points there are in each axis, A and B (and C)
  - Therefore we must specify A x B coordinates
    - Coordinate values for each node
    - All 2D/3D coordinates for 2D/3D space
  - No derived coordinates
    - Irregular meshes require the most information from the user to describe the structure

Irregular Grid

2D Grid, 2D (or 3D) Coordinates



Information:

- 2D space
- A x B nodes
- Coords[A][B][2] = { {0.0,0.0}, {0.2,0.0}, {0.4,0.15}, … }
- Data[A][B] = {…..}
- Connectivity = "2D Mesh"

Derived:

- Min Extents = {…}
- Max Extents = {…}

Irregular Field & Field File

- How to import an Irregular Field
  - 1D computational grid/list
  - 2D computational grid
  - Add 2D coordinates
  - Extend to 3D coordinates
  - Extend computational to 3D
  - Using interleaved and block coordinates
  - Using mixed data and coordinate files

2D Irregular Scalar ASCII Data File

Description of Data:

We have a data file called "temp.dat" which contains a 2D slice of temperature data from a Climate simulation. There are 96 x 73 nodes in the 2D grid, each node has a single float of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.
There are also two coordinate files called "world.x" and "world.y" which contain a set of irregular coordinates for the 2D grid. Both just contain data in ASCII format. Both have 7008 values.

world.x: (sample)

```
0.0 1.0 2.0 6.0 6.5 7.0 8.0 9.0
13.0 13.5 14.0 15.0 …
```

```
# AVS field file
ndim = 2
dim1 = 96
dim2 = 73
nspace = 2
veclen = 1
data = float
field = irregular

variable 1 file=temp.dat
  filetype=ascii

coord 1 file=world.x filetype=ascii
coord 2 file=world.y filetype=ascii
```

2D Irregular Single Coordinate File

Description of Data:

We have a data file called "temp.dat" which contains a 2D slice of temperature data from a Climate simulation. There are 96 x 73 nodes in the 2D grid, each node has a single float of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.
There is one coordinate file called "world.xy" which contains a set of irregular coordinates for the 2D grid. The file holds all the x values in one block followed by all the y values. The file is in binary format.

world.xy: (sample)

0A0A0A0A1B1B1B1B2C2C2C2C…
3E3E3E3E5A5A5A5A9D9D9D9D…

```
# AVS field file
ndim = 2
dim1 = 96
dim2 = 73
nspace = 2
veclen = 1
data = float
field = irregular

variable 1 file=temp.dat
  filetype=ascii

coord 1 file=world.xy
  filetype=binary skip=0
coord 2 file=world.xy
  filetype=binary skip=28032
```

NB. One float occupies 4 bytes, therefore 7008 values occupies 28032 bytes.

2D Irregular Interleaved Coordinates

Description of Data:

We have a data file called "temp.dat" which contains a 2D slice of temperature data from a Climate simulation. There are 96 x 73 nodes in the 2D grid, each node has a single float of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.
There is one coordinate file called "worldi.xy" which contains a set of irregular coordinates for the 2D grid. The file holds all the x and y values interleaved. The file is in binary format.

world.xy: (sample)

0A0A0A0A1B1B1B1B2C2C2C2C56565656…

```
# AVS field file
ndim = 2
dim1 = 96
dim2 = 73
nspace = 2
veclen = 1
data = float
field = irregular

variable 1 file=temp.dat
  filetype=ascii

coord 1 file=worldi.xy
  filetype=binary skip=0 stride=2
coord 2 file=worldi.xy
  filetype=binary skip=4 stride=2
```

NB. Skip jumps over bytes. Stride jumps over elements.

3D Irregular Single File

Description of Data:

We have a file called "bluntfin.dat" which contains a 3D volume of a density scalar, a momentum vector (3D) and a stagnation scalar. There are 40 x 32 x 32 nodes in the 3D irregular grid, each node has a five floats of data associated with it and three coordinate values. The file contains the data and the coordinates interleaved with each other, in the order density, momentum-u, momentum-v, momentum-w, stagnation, x, y, z. There is 12 bytes of header information. The data is written in binary format.

See next slide!

bluntfin.dat: (sample)

```
0A0A0A0A1B1B1B1B2C2C2C2C56565656
34343434A9A9A9A94B4B4B4B77777777
0A0A0A0A1B1B1B1B2C2C2C2C56565656
34343434A9A9A9A94B4B4B4B77777777...
```

3D Irregular Single File (2)

```
# AVS field file
ndim = 3
dim1 = 40
dim2 = 32
Dim3 = 32
nspace = 3
veclen = 5
data = float
field = irregular
labels = density momentum-u
  momentum-v momentum-w stagnation

variable 1 file=bluntfin.dat
  filetype=binary skip=12 stride=5
variable 2 file=bluntfin.dat
  filetype=binary skip=16 stride=5
variable 3 file=bluntfin.dat
  filetype=binary skip=20 stride=5
```

```
variable 4 file=bluntfin.dat
  filetype=binary skip=24 stride=5
variable 5 file=bluntfin.dat
  filetype=binary skip=28 stride=5

coord 1 file=bluntifn.dat
  filetype=binary skip=32 stride=5
coord 2 file=bluntfin.dat
  filetype=binary skip=36 stride=5
coord 3 file=bluntfin.dat
  filetype=binary skip=40 stride=5
```

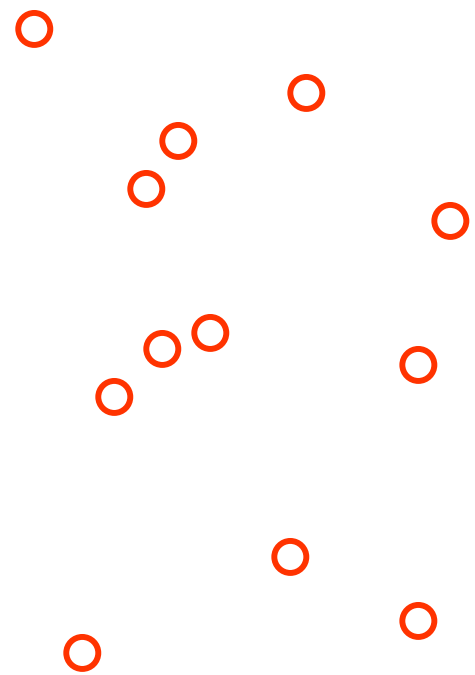
Exercise

- Exercise 4 in the workbook

Unstructured Fields

- Continue with more complex fields
- Just provide a quick overview of Cells
- Show an example UCD (Unstructured Cell Data) file which describes geometric unstructured grids and meshes.

Unstructured Nodes



Required:

- 2D space
- N (12) nodes
- Coords[N][2] = {
  {x0,y0}, {x1,y1}
  ,{x2,y2}...
  }
- Data[N] = {
  {val0}, {val1}, {val2},
  {val3}, …
  }
- Connectivity = “None”

2D Scatter Data File

Description of Data:

We have a data file called "scatter.dat" which contains a (1D) list of of data. There are 16 nodes in the list and each node has a single float of data associated with it. The file simply contains the data; no header information. The data is written in ASCII format.

There is also a coordinate file called "scatter.xy". It contains a set of irregular coordinates in 2D space for each of the 16 nodes. The coordinates are interleave. The file has no header and is in ASCII format.

scatter.dat: (sample)

```
34.5 66.0 1.0 128.45 111.71 …
```

scatter.xy: (sample)

```
1.0 1.0 3.0 7.8 3.0 9.9 4.0 0.2 …
```

```
# AVS field file
ndim = 1
dim1 = 16
nspace = 2
veclen = 1
data = float
field = irregular

variable 1 file=scatter.dat
  filetype=ascii

coord 1 file=scatter.xy
  filetype=ascii offset=0 stride=2
coord 2 file=scatter.xy
  filetype=ascii offset=1 stride=2
```

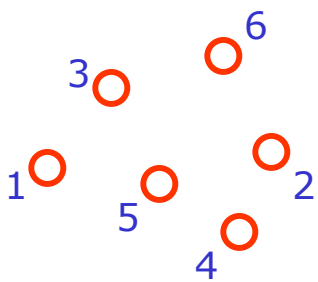
IAC manchester

Cells

- Relationships between nodes
  - Connectivity of a number of nodes
  - Geometric
- Types of cell
  - Point Cells
  - Line and Polyline Cells
  - Triangle and Polytriangle Cells
  - Quad Cells
  - Tetrahedrons, Hexahedrons
  - Prisms, Pyramids…

Types of Cells

- Point Cells
  - Just the nodes on their own
  - Is present to complete the classification



Nodes:

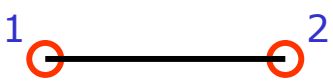
1: {0.0,0.0}
2: {1.0,0.2}
3: {0.25, 0.3}
4: ….

Cells:

1: 1
2: 2
3: 3
4: …

Types of Cells (2)

- Lines
  - Single straight lines
  - Requires 2 nodes, one for each end
  - Nodes may be shared
  - Nodes may be repeated (coincident)

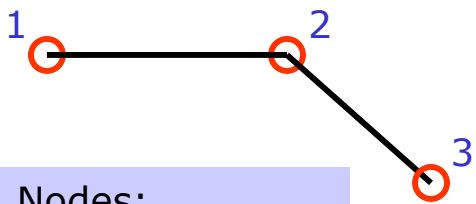


Nodes:

1: {0.0,0.0}
2: {1.0,0.0}

Cells:

1: 1-2

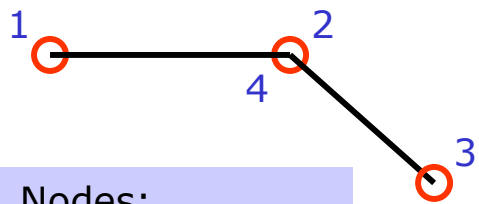


Nodes:

1: {0.0,0.0}
2: {1.0,0.0}
3: {1.8,-0.8}

Cells:

1: 1-2
2: 2-3



Nodes:

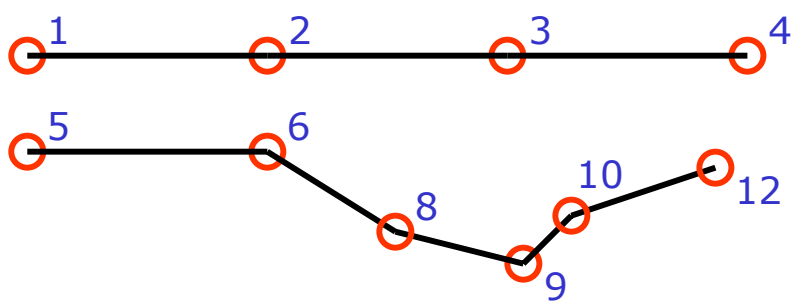
1: {0.0,0.0}
2: {1.0,0.0}
3: {1.8,-0.8}
4: {1.0,0.0}

Cells:

1: 1-2
2: 4-3

Types of Cells (3)

- Polyline Cells
  - Single line but with many (2+) nodes
  - Approximate curves or form irregular shapes
  - Nodes can be shared
  - Nodes can be repeated

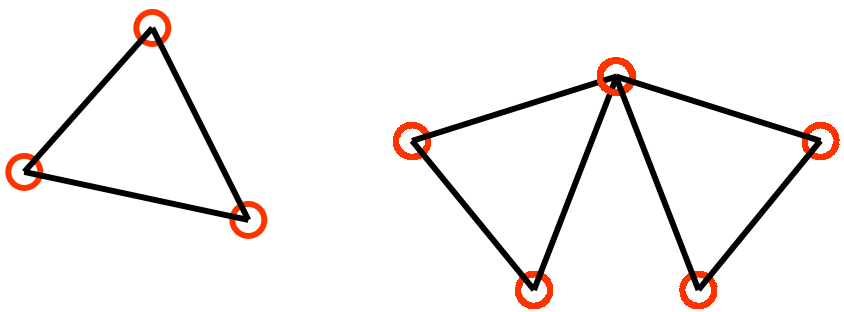


Cells:

1: 1-2-3-4
2: 5-6-8-9-10-12

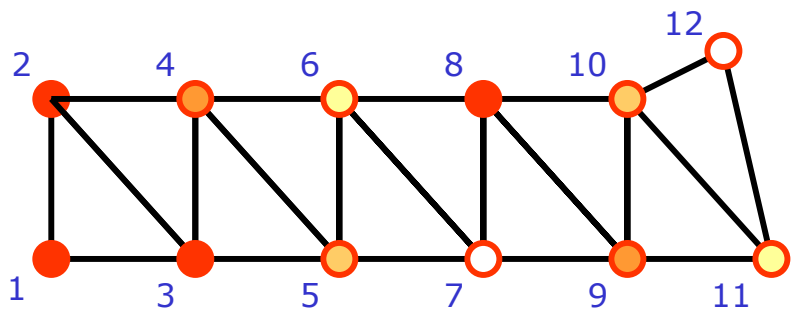
Types of Cells (4)

- Triangle Cells
  - Triangle requires 3 nodes
  - Nodes can be shared and repeated



Types of Cells (5)

- Polytriangles
  - If triangles share an edge (2 nodes)
  - It is possible to store strips of them
    - Start with three nodes for first triangle
    - Add an extra node and use the previous two for each subsequent triangle

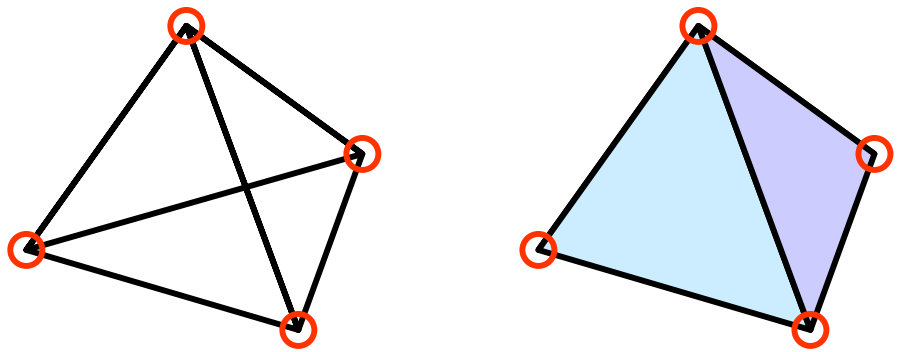


Triangle Connectivity

1) 1-2-3
2) 2-3-4
3) 3-4-5
4) 4-5-6
5) 5-6-7
6) 6-7-8
7) 7-8-9
8) 8-9-10
9) 9-10-11
10) 10-11-12

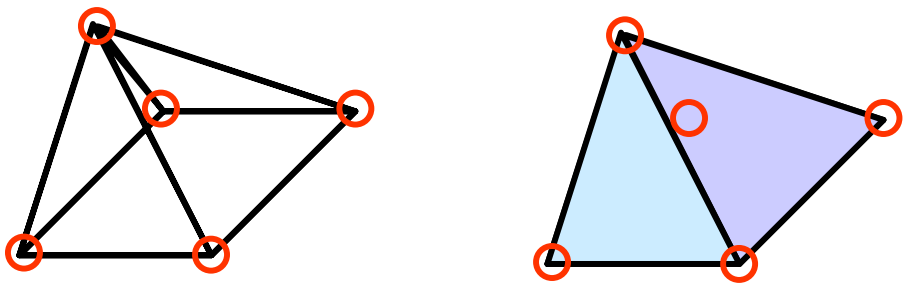
Types of Cells (6)

- Tetrahedron
  - 3D entity with 4 nodes



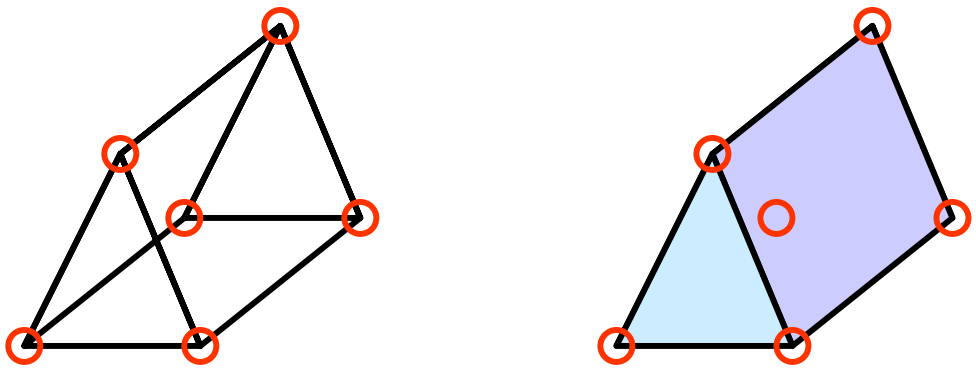
Types of Cell (7)

- Pyramid
  - Quadrilateral Base with extra node
  - 5 nodes in total



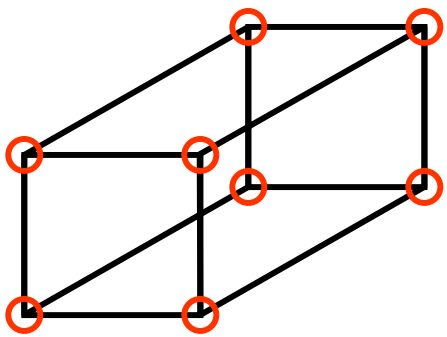
Types of Cell (8)

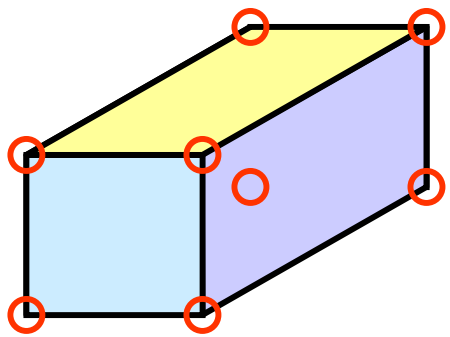
- Prism
  - Extruded triangle cell
  - 6 nodes in total



Types of Cell (9)

- Hexahedral
  - Cubic entity
  - Extruded quadrilateral
  - 8 nodes in total





Summary

- AVS/Express field file is extremely powerful
- Can be used to read an amazing variety of data
- Does not match full definition of a field
  - Only supports subset of internal Field object
  - E.g., does not read Cell Data or Cells
- It is a mechanism for describing your data so that AVS/Express can understand it

Summary (2)

- Sometimes the field file is inadequate to describe the organization of the data and its files
  - May be necessary to change the data, by converting it into an easier format
  - Alternative is to write your own reader which can access the full structure of the internal Field object
- Unfortunately there isn't enough time (in the world) to cover this fully :-)

AVS/Express Field

- Examine AVS/Express support for fields
  - Built-in support for a variety of field types
  - Hierarchical model for building organization
  - Library of pre-built types for common fields
- AVS/Express Field is for spatial data
- Look at structure from low-level first
- Build up definitions to see how:
  - Coordinates are stored
  - Data is stored
  - Relationships are stored

Data Array

- Basic building block of Fields
  - Simply holds an array of data and some information about it
  - May be structured or unstructured
    - Unstructured means an arbitrary list of points
    - Structured means grid data, i.e., 2D+
- It contains
  - The number of nodes “nnodes”
    - If structured, i.e., the data is gridded, then the number of nodes is equal to the product of the dims[] array
  - Ndims, dims[], veclen, data[], min/max etc
  - Created as a specific data type
  - “Data_Array+Float”

Node Data

- Definitions and terms used
  - An item is known as a node
  - Each node has a coordinate
  - Nodes may have data associated with them
    - This is known as Node Data
  - Nodes are organized in arrays
    - A list is just a 1D array
  - Node coordinates are an array
    - AVS/Express stores them in a “Data Array”
    - If the coordinates are in 3D space then the array will be 3D too

Node Data (2)

- Definitions and terms continued:
  - Node Data can have multiple components
    - Multivariate
  - Node Data components are stored as an array
    - Each components is in this array and has a “Data Array” associated with it
  - Each “Data Array” contains an array of data
    - Of one data type, e.g., integer, float
    - Scalar data is a 1D arrays
    - Vector data is 2D or higher