File Formats

for vtk Version 2.3

(Taken from The VTK User's Guide
Contact Kitware www.kitware.com to purchase)

vtk File Formats

The *Visualization Toolkit* provides a number of source and writer objects to read and write various data file formats. The *Visualization Toolkit* also provides some of its own file formats. The main reason for creating yet another data file format is to offer a consistent data representation scheme for a variety of dataset types, and to provide a simple method to communicate data between software. Whenever possible, we recommend that you use formats that are more widely used. But if this is not possible, the *Visualization Toolkit* formats described here can be used instead. Note, however, that these formats are not supported by many other tools.

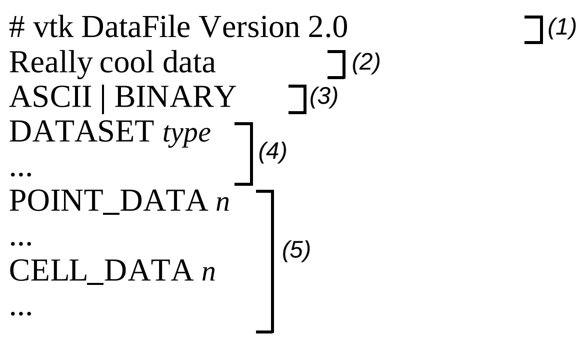
The visualization file formats consist of five basic parts.

1. The first part is the file version and identifier. This part contains the single line: `# vtk DataFile Version x.x`. This line must be exactly as shown with the exception of the version number `x.x`, which will vary with different releases of **vtk**. (Note: the current version number is 2.0. Version 1.0 files are compatible with version 2.0 files.)
2. The second part is the header. The header consists of a character string terminated by end-of-line character `\n`. The header is 256 characters maximum. The header can be used to describe the data and include any other pertinent information.
3. The next part is the file format. The file format describes the type of file, either ASCII or binary. On this line the single word `ASCII` or `BINARY` must appear.
4. The fourth part is the dataset structure. The geometry part describes the geometry and topology of the dataset. This part begins with a line containing the keyword `DATASET` followed by a keyword describing the type of dataset. Then, depending upon the type of dataset, other keyword/data combinations define the actual data.
5. The final part describes the dataset attributes. This part begins with the keywords `POINT_DATA` or `CELL_DATA`, followed by an integer number specifying the number of points or cells, respectively. (It doesn't matter whether `POINT_DATA` or `CELL_DATA` comes first.) Other keyword/data combinations then define the actual dataset attribute values (i.e., scalars, vectors, tensors, normals, texture coordinates, or field data).

An overview of the file format is shown in **Figure 1**. The first three parts are mandatory, but the other two are optional. Thus you have the flexibility of mixing and matching dataset attributes and geometry, either by operating system file manipulation or using **vtk** filters to merge data. Keywords are case insensitive, and may be separated by whitespace.

Before describing the data file formats please note the following.

- *dataType* is one of the types `bit`, `unsigned_char`, `char`, `unsigned_short`, `short`, `unsigned_int`, `int`, `unsigned_long`, `long`, `float`, or `double`. These keywords are used to describe the form of the data, both for reading from file, as well as constructing the



```
# vtk DataFile Version 2.0
Really cool data
ASCII | BINARY
DATASET type
...
POINT_DATA n
...
CELL_DATA n
...
```

Part 1: Header

Part 2: Title (256 characters maximum, terminated with newline `\n` character)

Part 3: Data type, either ASCII or BINARY

Part 4: Geometry/topology. *Type* is one of:

```
STRUCTURED_POINTS
STRUCTURED_GRID
UNSTRUCTURED_GRID
POLYDATA
RECTILINEAR_GRID
```

Part 5: Dataset attributes. The number of data items *n* of each type must match the number of points or cells in the dataset.

Figure 1 Overview of five parts of **vtk** data file format.

appropriate internal objects. Not all data types are supported for all classes.

- All keyword phrases are written in ASCII form whether the file is binary or ASCII. The binary section of the file (if in binary form) is the data proper; i.e., the numbers that define points coordinates, scalars, cell indices, and so forth.
- Indices are 0-offset. Thus the first point is point id 0.
- If both the data attribute and geometry/topology part are present in the file, then the number of data values defined in the data attribute part must exactly match the number of points or cells defined in the geometry/topology part.
- Cell types and indices are of type `int`.
- Binary data must be placed into the file immediately after the “newline” (`\n`) character from the previous ASCII keyword and parameter sequence.
- The geometry/topology description must occur prior to the data attribute description.

Binary Files

Binary files in **vtk** are portable across different computer systems as long as you observe two conditions. First, make sure that the byte ordering of the data is correct, and second, make sure that the length of each data type is consistent.

Most of the time **vtk** manages the byte ordering of binary files for you. When you write a binary file on one computer and read it in from another computer, the bytes representing the data will be automatically swapped as necessary. For example, binary files written on a Sun are stored in big endian order, while those on a PC are stored in little endian order. As a result, files written on a Sun workstation require byte swapping when read on a PC. (See the class vtkByteSwap for implementation details.) The **vtk** data files described here are written in big endian form.

Some file formats, however, do not explicitly define a byte ordering form. You will find that data read or written by external programs, or the classes vtkVolume16Reader, vtkMCubesReader, and vtkMCubesWriter may have a different byte order depending on the system of origin. In such cases, **vtk** allows you to specify the byte order by using the methods

```
SetDataByteOrderToBigEndian()
SetDataByteOrderToLittleEndian()
```

Another problem with binary files is that systems may use a different number of bytes to represent an integer or other native type. For example, some 64-bit systems will represent an integer with 8-bytes, while others represent an integer with 4-bytes. Currently, the *Visualization Toolkit* cannot handle transporting binary files across systems with incompatible data length. In this case, use ASCII file formats instead.

Dataset Format

The *Visualization Toolkit* supports five different dataset formats: structured points, structured grid, rectilinear grid, unstructured grid, and polygonal data. These formats are as follows.

- Structured Points
  The file format supports 1D, 2D, and 3D structured point datasets. The dimensions n_x, n_y, n_z must be greater than or equal to 1. The data spacing s_x, s_y, s_z must be greater than 0. (Note: in the version 1.0 data file, spacing was referred to as "aspect ratio". `ASPECT_RATIO` can still be used in version 2.0 data files, but is discouraged.)

  DATASET STRUCTURED_POINTS
  DIMENSIONS n_x n_y n_z
  ORIGIN *x y z*
  SPACING s_x s_y s_z

- Structured Grid
  The file format supports 1D, 2D, and 3D structured grid datasets. The dimensions n_x, n_y, n_z must be greater than or equal to 1. The point coordinates are defined by the data in the `POINTS` section. This consists of *x*-*y*-*z* data values for each point.

  DATASET STRUCTURED_GRID
  DIMENSIONS n_x n_y n_z
  POINTS *n dataType*
  p_{0x} p_{0y} p_{0z}
  p_{1x} p_{1y} p_{1z}
  ...
  $p_{(n-1)x}$ $p_{(n-1)y}$ $p_{(n-1)z}$

- Rectilinear Grid
  A rectilinear grid defines a dataset with regular topology, and semiregular geometry aligned along the *x-y-z* coordinate axes. The geometry is defined by three lists of monotonically increasing coordinate values, one list for each of the *x-y-z* coordinate axes. The topology is defined by specifying the grid dimensions, which must be greater than or equal to 1.

  `DATASET RECTILINEAR_GRID`
  `DIMENSIONS` $n_x\ n_y\ n_z$
  `X_COORDINATES` n_x *dataType*
  $x_0\ x_1 \ldots x_{(nx-1)}$
  `Y_COORDINATES` n_y *dataType*
  $y_0\ y_1 \ldots y_{(ny-1)}$
  `Z_COORDINATES` n_z *dataType*
  $z_0\ z_1 \ldots z_{(nz-1)}$

- Polygonal Data
  The polygonal dataset consists of arbitrary combinations of surface graphics primitives vertices (and polyvertices), lines (and polylines), polygons (of various types), and triangle strips. Polygonal data is defined by the `POINTS` `VERTICES`, `LINES`, `POLYGONS`, or `TRIANGLE_STRIPS` sections. The `POINTS` definition is the same as we saw for structured grid datasets. The `VERTICES`, `LINES`, `POLYGONS`, or `TRIANGLE_STRIPS` keywords define the polygonal dataset topology. Each of these keywords requires two parameters: the number of cells *n* and the size of the cell list *size*. The cell list size is the total number of integer values required to represent the list (i.e., sum of *numPoints* and connectivity indices over each cell). None of the keywords `VERTICES`, `LINES`, `POLYGONS`, or `TRIANGLE_STRIPS` is required.

  `DATASET POLYDATA`
  `POINTS` *n dataType*
  $p_{0x}\ p_{0y}\ p_{0z}$
  $p_{1x}\ p_{1y}\ p_{1z}$
  ...
  $p_{(n-1)x}\ p_{(n-1)y}\ p_{(n-1)z}$

  `VERTICES` *n size*
  $numPoints_0, i_0, j_0, k_0, \ldots$
  $numPoints_1, i_1, j_1, k_1, \ldots$
  ...
  $numPoints_{n-1}, i_{n-1}, j_{n-1}, k_{n-1}, \ldots$

  `LINES` *n size*
  $numPoints_0, i_0, j_0, k_0, \ldots$
  $numPoints_1, i_1, j_1, k_1, \ldots$
  ...
  $numPoints_{n-1}, i_{n-1}, j_{n-1}, k_{n-1}, \ldots$

  `POLYGONS` *n size*
  $numPoints_0, i_0, j_0, k_0, \ldots$
  $numPoints_1, i_1, j_1, k_1, \ldots$

...
numPoints$_{n-1}$, i_{n-1}, j_{n-1}, k_{n-1}, ...

`TRIANGLE_STRIPS` *n size*
numPoints$_0$, i_0, j_0, k_0, ...
numPoints$_1$, i_1, j_1, k_1, ...
...
numPoints$_{n-1}$, i_{n-1}, j_{n-1}, k_{n-1}, ...

- Unstructured Grid
  The unstructured grid dataset consists of arbitrary combinations of any possible cell type. Unstructured grids are defined by points, cells, and cell types. The `CELLS` keyword requires two parameters: the number of cells *n* and the size of the cell list *size*. The cell list size is the total number of integer values required to represent the list (i.e., sum of *numPoints* and connectivity indices over each cell). The `CELL_TYPES` keyword requires a single parameter: the number of cells *n*. This value should match the value specified by the `CELLS` keyword. The cell types data is a single integer value per cell that specified cell type (see `vtkCell.h` or **Figure 2**).

`DATASET UNSTRUCTURED_GRID`
`POINTS` *n dataType*
$p_{0x}\ p_{0y}\ p_{0z}$
$p_{1x}\ p_{1y}\ p_{1z}$
...
$p_{(n-1)x}\ p_{(n-1)y}\ p_{(n-1)z}$

`CELLS` *n size*
numPoints$_0$, *i, j, k, l*, ...
numPoints$_1$, *i, j, k, l*, ...
numPoints$_2$, *i, j, k, l*, ...
...
numPoints$_{n-1}$, *i, j, k, l*, ...

`CELL_TYPES` *n*
type$_0$
type$_1$
type$_2$
...
type$_{n-1}$

Dataset Attribute Format

The *Visualization Toolkit* supports the following dataset attributes: scalars (one to four components), vectors, normals, texture coordinates (1D, 2D, and 3D), 3×3 tensors, and field data. In addition, a lookup table using the RGBA color specification, associated with the scalar data, can be defined as well. Dataset attributes are supported for both points and cells.

Each type of attribute data has a *dataName* associated with it. This is a character string (without embedded whitespace) used to identify a particular data. The *dataName* is used by the **vtk** readers to extract data. As a result, more than one attribute data of the same type can be included in a file. For example, two different scalar fields defined on the dataset points, pressure and temperature, can be contained in the same file. (If the appropriate *dataName* is not specified in the **vtk** reader, then the first data of that type is extracted from the file.)

- Scalars
  Scalar definition includes specification of a lookup table. The definition of a lookup table is optional. If not specified, the default **vtk** table will be used (and *tableName* should be "`default`"). Also note that the *numComp* variable is optional—by default the number of components is equal to one. (The parameter *numComp* must range between (1,4) inclusive; in versions of **vtk** prior to vtk2.3 this parameter was not supported.)

  `SCALARS` *dataName dataType numComp*
  `LOOKUP_TABLE` *tableName*
  s_0
  s_1
  ...
  s_{n-1}

  The definition of color scalars (i.e., `unsigned char` values directly mapped to color) varies depending upon the number of values (*nValues*) per scalar. If the file format is `ASCII`, the color scalars are defined using *nValues* `float` values between (0,1). If the file format is `BINARY`, the stream of data consists of *nValues* `unsigned char` values per scalar value.

  `COLOR_SCALARS` *dataName nValues*
  $c_{00}\ c_{01}\ \cdots\ c_{0(nValues-1)}$
  $c_{10}\ c_{11}\ \cdots\ c_{1(nValues-1)}$
  ...
  $c_{(n-1)0}\ c_{(n-1)1}\ \cdots\ c_{(n-1)(nValues-1)}$

- Lookup Table
  The *tableName* field is a character string (without imbedded white space) used to identify the lookup table. This label is used by the **vtk** reader to extract a specific table.

  Each entry in the lookup table is a `rgba[4]` (*red-green-blue-alpha*) array (*alpha* is opacity where *alpha=0* is transparent). If the file format is `ASCII`, the lookup table values must be `float` values between (0,1). If the file format is `BINARY`, the stream of data must be four unsigned char values per table entry.

  `LOOKUP_TABLE` *tableName size*
  $r_0\ g_0\ b_0\ a_0$
  $r_1\ g_1\ b_1\ a_1$
  ...
  $r_{size-1}\ g_{size-1}\ b_{size-1}\ a_{size-1}$

- Vectors

  `VECTORS` *dataName dataType*
  $v_{0x}\ v_{0y}\ v_{0z}$

v_{1x} v_{1y} v_{1z}
...
$v_{(n-1)x}$ $v_{(n-1)y}$ $v_{(n-1)z}$

- Normals
  Normals are assumed normalized $|n| = 1$.

  `NORMALS` *dataName dataType*
  n_{0x} n_{0y} n_{0z}
  n_{1x} n_{1y} n_{1z}
  ...
  $n_{(n-1)x}$ $n_{(n-1)y}$ $n_{(n-1)z}$

- Texture Coordinates
  Texture coordinates of 1, 2, and 3 dimensions are supported.

  `TEXTURE_COORDINATES` *dataName dim dataType*
  t_{00} t_{01} ... $t_{0(dim-1)}$
  t_{10} t_{11} ... $t_{1(dim-1)}$
  ...
  $t_{(n-1)0}$ $t_{(n-1)1}$ ... $t_{(n-1)(dim-1)}$

- Tensors
  Currently only 3×3 real-valued, symmetric tensors are supported.

  `TENSORS` *dataName dataType*
  t^0_{00} t^0_{01} t^0_{02}
  t^0_{10} t^0_{11} t^0_{12}
  t^0_{20} t^0_{21} t^0_{22}

  t^1_{00} t^1_{01} t^1_{02}
  t^1_{10} t^1_{11} t^1_{12}
  t^1_{20} t^1_{21} t^1_{22}

  ...
  t^{n-1}_{00} t^{n-1}_{01} t^{n-1}_{02}
  t^{n-1}_{10} t^{n-1}_{11} t^{n-1}_{12}
  t^{n-1}_{20} t^{n-1}_{21} t^{n-1}_{22}

- Field Data
  Field data is essentially an array of data arrays. Defining field data means giving a name to the field and specifying the number of arrays it contains. Then, for each array, the name of the array *arrayName(i)*, the number of components of the array, *numComponents*, the number of tuples in the array, *numTuples*, and the data type, *dataType*, are defined.

  `FIELD` *dataName numArrays*
  arrayName0 numComponents numTuples dataType
  f_{00} f_{01} ... $f_{0(numComponents-1)}$
  f_{10} f_{11} ... $f_{1(numComponents-1)}$

...
$f_{(numTuples-1)0}\ f_{(numTuples-1)1} \cdots f_{(numTuples-1)(numComponents-1)}$

arrayName1 numComponents numTuples dataType
$f_{00}\ f_{01} \cdots f_{0(numComponents-1)}$
$f_{10}\ f_{11} \cdots f_{1(numComponents-1)}$
...
$f_{(numTuples-1)0}\ f_{(numTuples-1)1} \cdots f_{(numTuples-1)(numComponents-1)}$

...
arrayName(numArrays-1) numComponents numTuples dataType
$f_{00}\ f_{01} \cdots f_{0(numComponents-1)}$
$f_{10}\ f_{11} \cdots f_{1(numComponents-1)}$
...
$f_{(numTuples-1)0}\ f_{(numTuples-1)1} \cdots f_{(numTuples-1)(numComponents-1)}$

Examples

The first example is a cube represented by six polygonal faces. We define a single-component scalar, normals, and field data on the six faces. There are scalar data associated with the eight vertices. A lookup table of eight colors, associated with the point scalars, is also defined.

```
# vtk DataFile Version 2.0
Cube example
ASCII
DATASET POLYDATA
POINTS 8 float
0.0 0.0 0.0
1.0 0.0 0.0
1.0 1.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
1.0 0.0 1.0
1.0 1.0 1.0
0.0 1.0 1.0
POLYGONS 6 30
4 0 1 2 3
4 4 5 6 7
4 0 1 5 4
4 2 3 7 6
4 0 4 7 3
4 1 2 6 5

CELL_DATA 6
SCALARS cell_scalars int 1
LOOKUP_TABLE default
0
1
2
3
```

```
4
5
NORMALS cell_normals float
0 0 -1
0 0 1
0 -1 0
0 1 0
-1 0 0
1 0 0
FIELD FieldData 2
cellIds 1 6 int
0 1 2 3 4 5
faceAttributes 2 6 float
0.0 1.0 1.0 2.0 2.0 3.0 3.0 4.0 4.0 5.0 5.0 6.0

POINT_DATA 8
SCALARS sample_scalars float 1
LOOKUP_TABLE my_table
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
LOOKUP_TABLE my_table 8
0.0 0.0 0.0 1.0
1.0 0.0 0.0 1.0
0.0 1.0 0.0 1.0
1.0 1.0 0.0 1.0
0.0 0.0 1.0 1.0
1.0 0.0 1.0 1.0
0.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0
```

The next example is a volume of dimension $3 \times 4 \times 5$. Since no lookup table is defined, either the user must create one in **vtk**, or the default lookup table will be used.

```
# vtk DataFile Version 2.0
Volume example
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 3 4 6
ASPECT_RATIO 1 1 1
ORIGIN 0 0 0
POINT_DATA 72
SCALARS volume_scalars char 1
LOOKUP_TABLE default
0 0 0 0 0 0 0 0 0 0 0 0
0 5 10 15 20 25 25 20 15 10 5 0
0 10 20 30 40 50 50 40 30 20 10 0
0 10 20 30 40 50 50 40 30 20 10 0
```

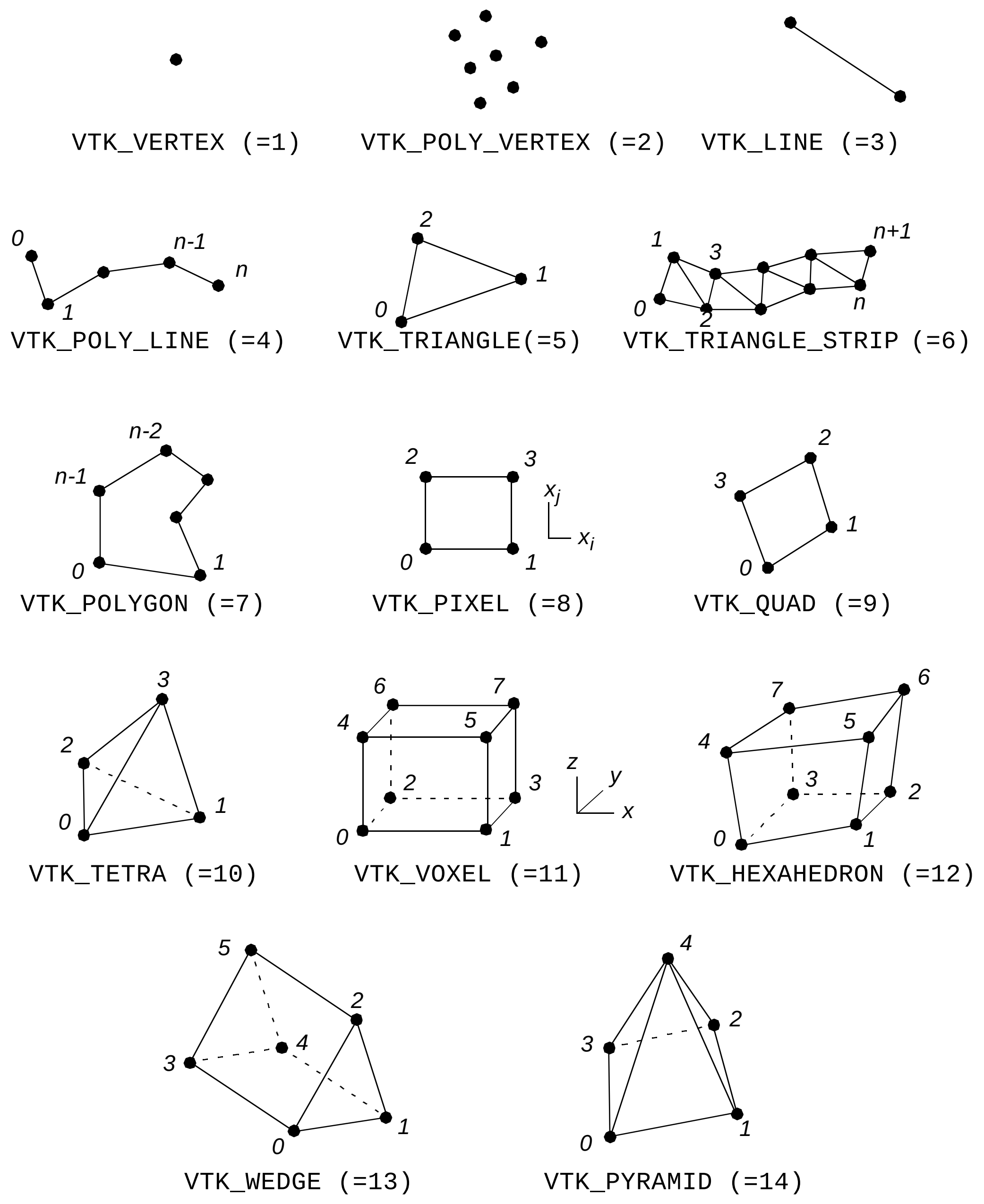


Figure 2 Cell type specification. Use the include file CellType.h to manipulate cell types.

```
0 5 10 15 20 25 25 20 15 10 5 0
0 0 0 0 0 0 0 0 0 0 0 0
```

The third example is an unstructured grid containing all 12 cell types. The file contains scalar and vector data.

```
# vtk DataFile Version 2.0
Unstructured Grid Example
ASCII
```

```
DATASET UNSTRUCTURED_GRID
POINTS 27 float
0 0 0   1 0 0   2 0 0   0 1 0   1 1 0   2 1 0
0 0 1   1 0 1   2 0 1   0 1 1   1 1 1   2 1 1
0 1 2   1 1 2   2 1 2   0 1 3   1 1 3   2 1 3
0 1 4   1 1 4   2 1 4   0 1 5   1 1 5   2 1 5
0 1 6   1 1 6   2 1 6

CELLS 11 60
8 0 1 4 3 6 7 10 9
8 1 2 5 4 7 8 11 10
4 6 10 9 12
4 5 11 10 14
6 15 16 17 14 13 12
6 18 15 19 16 20 17
4 22 23 20 19
3 21 22 18
3 22 19 18
2 26 25
1 24

CELL_TYPES 11
12
12
10
10
7
6
9
5
5
3
1


POINT_DATA 27
SCALARS scalars float 1
LOOKUP_TABLE default
0.0   1.0   2.0   3.0   4.0   5.0
6.0   7.0   8.0   9.0   10.0   11.0
12.0   13.0   14.0   15.0   16.0   17.0
18.0   19.0   20.0   21.0   22.0   23.0
24.0   25.0   26.0
VECTORS vectors float
1 0 0   1 1 0   0 2 0   1 0 0   1 1 0   0 2 0
1 0 0   1 1 0   0 2 0   1 0 0   1 1 0   0 2 0
0 0 1   0 0 1   0 0 1   0 0 1   0 0 1   0 0 1
0 0 1   0 0 1   0 0 1   0 0 1   0 0 1   0 0 1
0 0 1   0 0 1   0 0 1
```

Additional examples are available in the data examples directory.