## OpenDX och ParaView

We end the course with two visualization systems that have more advanced graphics than Matlab. These systems have no support for computations (apart from very simple ones), and the user has to supply the plot-data using files.
In previous versions of the course the focus was on OpenDX, but this year we will use ParaView. See the old PDF-file from the Diary for more about OpenDX.
Let us have a look at OpenDX before we start with ParaView. OpenDX, `www.opendx.org`, is an open version of IBM's "Visualization Data Explorer".

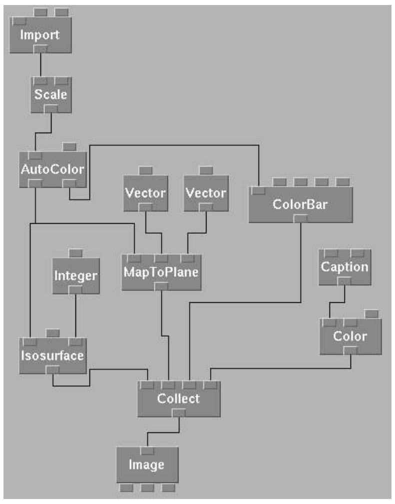
Some, but not all, important points:

- Advanced tools for visualization of 3D-data.
- Takes longer to learn than Matlab, but you can do more. Often faster.
- Modules are connected using a GUI, graphical programming. Visual Program Editor, VPE.
- Input from files (not variables as in Matlab).
- The modules transform the input and sends it to the next module.
- Supports several input formats. Using the "Data Prompter"-program simple inputs can be handled (e.g. uniform, gridded input).
- Lots of documentation. Many demo programs. Few simple examples. Should read a book (or take this course :-)
  David Thompson, Jeff Braun, Ray Ford,
  OpenDX: Paths to Visualization. Consists of a sequence of solved visualization problems.
  `http://www.vizsolutions.com`.

Here is a short example (an extract from the old course) to give you an idea of how one uses OpenDX.

We would like to visualize data of the form $w = f(x, y, z)$.
It is possible to remove part of the data (everything on one side of a plane). We use the module ClipPlane. It takes the data, a point in the plane and a normal defining the clip plane.
Everything on the side of the plane (in the direction of the normal) is removed.

Here is a related construct. The MapToPlane-module creates an arbitrary cutting plane through 3D-space and interpolates data values onto it. The plane is defined by a point a normal, just as the ClipPlane. Using the Vector interactors we can move to plane.
I have combined MapToPlane with Isosurface. I have also added Colorbar which draws a scale (as in Matlab). Finally there is Caption which corresponds to Matlab's title.
Here is the program



and here is a (bad) version of the resulting image.

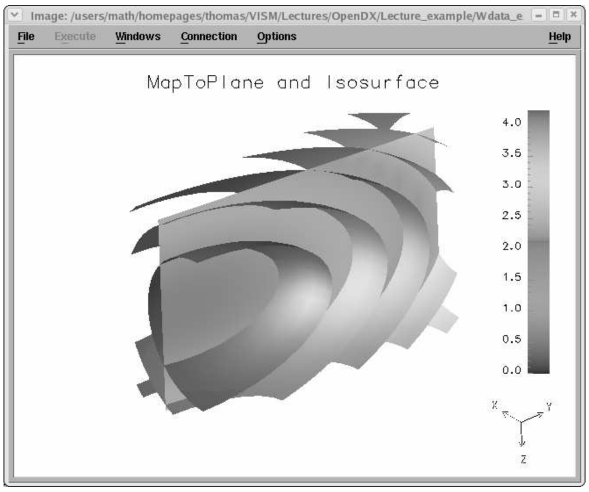


Image: /users/math/homepages/thomas/VISM/Lectures/OpenDX/Lecture_example/Wdata_e
File
Execute
Windows
Connection
Options
Help
MapToPlane and Isosurface
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X
Y
Z

For more details see the old handouts. The rest of the chapter deals with ParaView.

## ParaView

Here are a few sentences from `www.paraview.org`:

Overview:
ParaView is an open-source, multi-platform application designed to visualize data sets of size varying from small to very large.

The goals of the ParaView project include the following:

- Develop an open-source, multi-platform visualization application.
- Support distributed computation models to process large data sets.
- Create an open, flexible, and intuitive user interface.
- Develop an extensible architecture based on open standards.

ParaView runs on distributed and shared memory parallel as well as single processor systems and has been successfully tested on Windows, Mac OS X, Linux and various Unix workstations, clusters and supercomputers. Under the hood, ParaView uses the Visualization Toolkit as the data processing and rendering engine and has a user interface written using Qt®.

The ParaView project started started in 2000 as a collaborative effort between Kitware Inc. and Los Alamos National Laboratory. The initial funding was provided by a three year contract with the US Department of Energy ASCI Views program. Today, ParaView development continues as a collaboration between Kitware, Sandia National Labs, CSimSoft, Los Alamos National Lab, Army Research Lab and others.

---

There is a set of books available from Kitware Inc. providing details about VTK and ParaView. In this course it is sufficient to study the "ParaView 3 tutorial for Supercomputing 07" (used in the labs), and the "VTK file formats documentation" (see the home page for links).

The hardest part with using OpenDX and ParaView is the creation of the input files and this chapter will show you some examples.

VTK supports many styles of file formats. In this course we will use two, the legacy VTK formats and the XML formats.

From the dictionary:

> Legacy: Designating software or hardware which, although outdated or limiting, is an integral part of a computer system and difficult to replace.

Suppose we want implement the following Matlab-program in ParaView:

```
[X, Y] = meshgrid(linspace( 0, 2, 30), ...
                  linspace(-1, 1, 30));
surf(X, Y, X.^2 + sin(3 * Y))
```

Here is a first step, the file `ex1.vtk` (the line numbers are not part of the file). For more details see the formats-manual.

```
1   # vtk DataFile Version 2.0
2   Data for z = f(x, y).
3   ASCII
4   DATASET STRUCTURED_POINTS
5   DIMENSIONS 3 3 1
6   ORIGIN     0 0 0
7   SPACING    1 1 1
8
9   POINT_DATA 9
10  SCALARS name_1 float
11  LOOKUP_TABLE default
12  1 2 3 4 5 6 7 8 9
```

Line 1 is a header, and line 2 a title (comment). Line 3 gives the data format for numbers (coordinates etc), see the documentation for binary formats.

Lines 4-7 describe the dataset structure (also called the geometry or the topology) of the data. In our case we have grid points in the x-y-plane. The points are $(j, k)$, $j$, $k = 0, 1, 2$.
Finally, on lines 9-12, we have the dataset attributes, the values of the function on the grid (the values are 1-9).

We have `POINT_DATA`, i.e. we have defined a scalar value in each grid point. The value is a scalar-float (i.e not a vector for example) and we have named it, `name_1`. Choose meaningful names e.g. pressure, temperature etc.

We can have several quantities, by having several groups like 10-13. Using the name, we can later pick the relevant quantity in ParaView. On line 11 we define a colour lookup table (here the default). One should be able to define ones own, but this seems buggy in the present ParaView-version.

I have not included any images in the handouts, since the PDF-files become so large. Some of the `vtk`-files (and corresponding images) are available on the student computer system, so you can try them yourself (see `~thomas/VIS/Handouts_ex_ParaView`).

The following line denotes a missing image.
[Image]

This is how I made the [Image]. I loaded the file, choose the "Glyph-filter", changed the "Glyph Type" to "Sphere", increased the "Radius" and "Theta Resolution". I pressed the "Toggle Color Legend Visibility"-button. Not to waste printer-toner, I changed the background colour (so the background text is not very visible).

So what is a glyph?

(Glyph from from Greek Glyphe, carved work, from
glyphein to carve.
1: an ornamental vertical groove especially in a Doric frieze
2: a symbolic figure or a character (as in the Mayan system of writing) usually incised or carved in relief
3: a symbol (as a curved arrow on a road sign) that conveys information nonverbally).

Looking at the image we can see that the point data is ordered the following way:

$(x_1, y_1)$ $(x_2, y_1)$ $(x_3, y_1)$ $(x_1, y_2)$ $(x_2, y_2)$ $(x_3, y_2)$ $(x_1, y_3)$ $(x_2, y_3)$ $(x_3, y_3)$

How can we generate data in that order from Matlab? Here is a short example:

```
>> [X, Y] = meshgrid(-1:1, -1:1)

X =  -1     0     1
     -1     0     1
     -1     0     1

Y =  -1    -1    -1
      0     0     0
      1     1     1

>> [X(:), Y(:)]

ans =
    -1    -1  % (x_min,          y_min)
    -1     0  % (x_min,          y_min +   dy)
    -1     1  % (x_min,          y_min + 2 dy)
     0    -1  % (x_min +   dx,   y_min)
     0     0  % (x_min +   dx,   y_min +   dy)
     0     1  % (x_min +   dx,   y_min + 2 dy)
     1    -1  % (x_min + 2 dx,   y_min)
     1     0  % (x_min + 2 dx,   y_min +   dy)
     1     1  % (x_min + 2 dx,   y_min + 2 dy)

>> X = X';   % Not what we want, so transpose
>> Y = Y';
>> [X(:), Y(:)]
ans =
    -1    -1  % (x_min,          y_min)
     0    -1  % (x_min +   dx,   y_min)
     1    -1  % (x_min + 2 dx,   y_min)
    -1     0  % (x_min,          y_min +   dy)
     0     0  % (x_min +   dx,   y_min +   dy)
     1     0  % (x_min + 2 dx,   y_min +   dy)
    -1     1  % (x_min,          y_min + 2 dy)
     0     1  % (x_min +   dx,   y_min + 2 dy)
     1     1  % (x_min + 2 dx,   y_min + 2 dy)
```

Here comes a Matlab-program that produces a suitable datafile for ParaView. In a real application, we may have a Fortran/C/C++-code that produces the data.

```
% Make surface data for ParaView
n = 30;
[X, Y] = meshgrid(linspace( 0, 2, 30), ...
                  linspace(-1, 1, 30));
Z = X.^2 + sin(3 * Y);

% Open output file
fid = fopen('surf_ex.vtk', 'w');

% Write a header and a comment
fprintf(fid, '# vtk DataFile Version 2.0\n');
fprintf(fid, 'z = x^2 + sin(3 y)\n');

% Data type and type of grid
fprintf(fid, 'ASCII\n');
fprintf(fid, 'DATASET STRUCTURED_POINTS\n');

% Here comes the data. First the nodes.
fprintf(fid, 'DIMENSIONS %d %d 1\n', n, n);  % z = 1
fprintf(fid, 'ORIGIN 0 -1 0\n');

% spacing not used for z
spacing = X(1, 2) - X(1, 1); % i.e. 2 / (n - 1)
fprintf(fid, 'SPACING %e %e %e\n', ...
             spacing, spacing, spacing);

fprintf(fid, 'POINT_DATA %d\n', n * n);
fprintf(fid, 'SCALARS z float\n');
fprintf(fid, 'LOOKUP_TABLE default\n');
fprintf(fid, '%e\n', Z'); % Note, transpose

fclose(fid);  % close file
```

In ParaView the data will show up as a flat coloured plane (where the colours correspond to the Z-values). To produce heights from the Z-values we use two filters, “Clean to Grid” followed by “Warp(scalar)”. The first filter (quoting the help):

> It also converts the data set to an unstructured grid. You may wish to do this if you want to apply a filter to your data set that is available for unstructured grids but not for the initial type of your data set (e.g., applying warp vector to volumetric data).

and the second

> The Warp (scalar) filter translates the points of the input data set along a vector by a distance determined by the specified scalars. This filter operates on polygonal, curvilinear, and unstructured grid data sets containing single-component scalar arrays.

The vector is $(0, 0, 1)$ in this case. The warp-filter has a “Scale Factor” so one can exaggerate (scale) the z-direction.

Another filter, which we can apply directly on the data, is “Contour”.

[Image]

If we make a mistake in the VTK-file, we get an error message in a separate window “Output Message”. As an example, if we give two, instead of three, numbers in the `DIMENSIONS`-statement we get the following error message:

```
ERROR: In /home/berk/Work/ReleaseBuilds/ParaView3/
VTK/IO/vtkStructuredPointsReader.cxx, line 131
vtkStructuredPointsReader (0x8ca9d18):
Error reading dimensions!
```

I have fetched a pre-compiled binary, that is the reason for the absolute path.
It may be instructive to look at the source, to see the origin of the message. Fetching and unpacking `vtk-5.2.0.tar.gz` from `http://www.vtk.org/get-software.php` we look at the C++-file `VTK/IO/vtkStructuredPointsReader.cxx`

```
% wc -l vtkStructuredPointsReader.cxx
 533 vtkStructuredPointsReader.cxx

if ( ! strncmp(this->LowerCase(line), "dimensions",10) )
  {
  int dim[3];
  if (!(this->Read(dim) &&
        this->Read(dim+1) &&
        this->Read(dim+2)))
    {
    vtkErrorMacro(<<"Error reading dimensions!");
    this->CloseVTKFile ();
    this->SetErrorCode( vtkErrorCode::FileFormatError );
    return 1;
    }
```

`vtkErrorMacro` is line 131.

In the following VTK-file we construct a tiny vector field in 3D. You should use more points in a real application. You could have `SCALARS`-data as well.

```
# vtk DataFile Version 2.0
Vector field in 3D.
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 3 3 3
ORIGIN     0 0 0
SPACING    1 1 1

POINT_DATA 27
VECTORS vec float
1 2 3
  etc.
```

One could use the “Stream Tracer” and “Tube” filters to visualize the flow. [Image]
In the previous examples every node (point) has a quantity (scalar or vector) associated with it. In some applications it is more natural to associate a value with an area or volume (a so-called cell). A biologist may count the number of bugs, plants etc. per km$^2$ or number of fish per km$^3$.
Here comes a 2D-example using cell-data with scalar values.

```
# vtk DataFile Version 2.0
A 2D cell example
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 4 4 1
ORIGIN     0 0 0
SPACING    1 1 1

CELL_DATA 9
SCALARS name float
LOOKUP_TABLE default
1 2 3 4 5 6 7 8 9
```

Line 5 defines a $4 \times 4$ point grid so with $3 \times 3$ cells, i.e. nine values on line 9. A plot gives a checkerboard pattern (in color). [Image]
Here is a 3D-example with $3 \times 3 \times 3$ cells, i.e cubes. The central cube is number 14, having the value $-1$ on line 13. The data can be inspected using the Clip filter, for example. [Image]

```
1  # vtk DataFile Version 2.0
2  A 2D cell example
3  ASCII
4  DATASET STRUCTURED_POINTS
5  DIMENSIONS 4 4 4
6  ORIGIN     0 0 0
7  SPACING    1 1 1
8
9  CELL_DATA 27
10 SCALARS name float
11 LOOKUP_TABLE default
12  1  2  3  4  5  6  7  8  9
13 10 11 12 13 -1 15 16 17 18
14 19 20 21 22 23 24 25 26 27
```

Here is a 2D cell example where we associate a vector with each cell. Using the "Cell Centers" and "Glyph"-filters, we get arrows starting in the center of each cell (square). [Image]

```
1  # vtk DataFile Version 2.0
2  A 2D cell, vector, example
3  ASCII
4  DATASET STRUCTURED_POINTS
5  DIMENSIONS 4 4 1
6  ORIGIN     0 0 0
7  SPACING    1 1 1
8
9  CELL_DATA 9
10 VECTORS vec float
11 1 0 0   1 0 0   1 0 0
12 0 1 0   0 1 0   0 1 0
13 0 0 1   0 0 1   0 0 1
```

In the next example we create a more complicated geometry which is not quite so regular. Let us make a simple model of the surface of a house. We use triangles to construct the surface (compare the surface mesh in a finite element computation).
We use point data from now on.
This primitive drawing shows the numbering of the points.

```
8  -------- 9     Top of roof
|\         |\
  6 --------- 7
|/         |/     Roof level
4 --------- 5

  2 --------- 3
 /          /     Ground level
0 --------- 1
```

```
1  # vtk DataFile Version 2.0
2  A house
3  ASCII
4  DATASET POLYDATA
5
6  POINTS 10 float
7  0 0 0    2 0 0    0 1 0    2 1 0
8  0 0 1    2 0 1    0 1 1    2 1 1
9  0 0.5 1.5  2 0.5 1.5
10
11 TRIANGLE_STRIPS 2 20
12 10    0 4 1 5 3 7 2 6 0 4
13  8    4 8 5 9 7 8 6 4
14
15 POINT_DATA 10
16 SCALARS name float
17 LOOKUP_TABLE default
18 11 12 13 14 15 16 17 18 19 20
```

On lines 6-9 we list the 10 coordinates for the points that define the corners of the triangles. The walls are made using one triangle strip (saves space compared to separate triangles), line 12. The points are numbered in a zig-zag-order, the first point having index zero. The roof is defined on line 13. The numbers on line 11 denote number of strips and number of integers in lines 12, 13. The first number on line 12 (13) denotes the numbers of points in the strip.

By using "Surface With Edges", using "Glyph" with "Glyph Type = Sphere", "Scalar Mode=scalar", clicking in "Edit" and setting "Set Scale factor=0.01" we get the following [Image].

---

Here comes an example of an unstructured grid composed by tetrahedrons. We reuse the points from the house example.
I used Matlab to construct the tetrahedrons, here is a code segment. **x**, **y** and **z**, contain the coordinates from the house.

```
...
% Tesselate the volume using tetrahedrons. T is an
% n_tetra x 4 matrix with indices into x, y and z.

T = delaunay3(x, y, z, [])

n_tetra = size(T, 1);
C       = jet(n_tetra);  % some colours

% Explode the view by moving the tetrahedrons
% from the centre
xm = mean(x);
ym = mean(y);
zm = mean(z);
d = 0.1;         % scale factor

% P is used to extract corners in the four faces
% of a tetrahedron
P = [1 2 3; 1 2 4; 1 3 4; 2 3 4];
```

```
for k = 1:n_tetra
  xmT = mean(x(T(k, :)));  % centre of tetrahedron
  ymT = mean(y(T(k, :)));
  zmT = mean(z(T(k, :)));
  vx = d * (xmT - xm);     % translation
  vy = d * (ymT - ym);
  vz = d * (zmT - zm);

  for j = 1:4              % plot all four faces
    t = T(k, P(j, :));
    fill3(x(t) + vx, y(t) + vy, z(t) + vz, C(k, :))
  end
end
...
```

Here is the **T**-matrix

```
T =
     7     3     2     1
     7     2     5     1
     7     3     4     2
     7     4     8     2
     7     6     5     2
     7     8     6     2
    10     8     6     5
    10     7     5     9
    10     7     8     5
```

Here is a sequence of images each with a different **d**-value, showing an "exploded view" [Image].

Boris Nikolaevich Delaunay or Delone, 1890-1980, was one of the first Russian mountain climbers and a Soviet/Russian mathematician (according to Wikipedia).

What is the difference, with respect to visualization, between the two houses (the first and the second)?

| Filter | First house | Second house |
|---|---|---|
| none | surface | volume |
| contour | lines | surfaces |
| clip | surface | volume |
| slice | line | surface |

Here comes the vtk-file:

```
1   # vtk DataFile Version 2.0
2   A tesseleted house
3   ASCII
4   DATASET UNSTRUCTURED_GRID
5
6   POINTS 10 float
7   0 0 0   2 0 0   0 1 0    2 1 0
8   0 0 1   2 0 1   0 1 1    2 1 1
9   0 0.5 1.5  2 0.5 1.5
10
11  CELLS 9 45
12        4    6    2    1    0
13        4    6    1    4    0
14        4    6    2    3    1
15        4    6    3    7    1
16        4    6    5    4    1
17        4    6    7    5    1
18        4    9    7    5    4
19        4    9    6    4    8
20        4    9    6    7    4
21
22  CELL_TYPES 9
23  10 10 10 10 10 10 10 10 10
24
25  POINT_DATA 10
26  SCALARS name float
27  LOOKUP_TABLE default
28  11 12 13 14 15 16 17 18 19 20
```

Line 4 has been changed from the first version. Lines 6-9 are unchanged. I have replaced the 2D triangle strips with 3D tetrahedrons, lines 11-23. The rest of the file is unchanged.

Line 11 starts the description of the corners of the tetrahedrons, there are nine tetrahedrons and 45 $(9 \cdot 5)$ numbers are required to describe them. Line 12, `4 6 2 1 0`, says that the coordinates of the four (the first 4) corners are given by 6:th, 2:d, 1:th and 0:th point (indices start at zero). To get the correct indices I had to subtract one from the `T`-matrix produced by `delaunay3`.

Lines 22-23 describe the type of cells. We have nine tetrahedrons, which are identified by number ten (see the formats-manual for the numbers). [Image]

---

Suppose you want to visualize data produced by $w = f(x, y, z)$, and where you are using Matlab to produce the $w$-values. It is better to use `ndgrid` instead of `meshgrid` as will be explained below.

```
>> [X, Y, Z] = ndgrid(0:0.1:1, 10:2:40, -1:0.1:1);
>> W = X.^2 + (0.05 * (Y - 10)).^2 + Z.^2;
>> w = W(:);
>> save -ascii wdata w  % for example
```

To understand how the values are stored in the file we look at a much smaller example.

```
>> [X, Y, Z] = ndgrid(0.1:0.1:0.3, -1:1, 20:10:40)

X(:,:,1) =
    0.1000    0.1000    0.1000
    0.2000    0.2000    0.2000
    0.3000    0.3000    0.3000
X(:,:,2) =
    0.1000    0.1000    0.1000
    0.2000    0.2000    0.2000
    0.3000    0.3000    0.3000
```

```
X(:,:,3) =
    0.1000    0.1000    0.1000
    0.2000    0.2000    0.2000
    0.3000    0.3000    0.3000

Y(:,:,1) =
    -1     0     1
    -1     0     1
    -1     0     1
Y(:,:,2) =
    -1     0     1
    -1     0     1
    -1     0     1
Y(:,:,3) =
    -1     0     1
    -1     0     1
    -1     0     1

Z(:,:,1) =
    20    20    20
    20    20    20
    20    20    20
Z(:,:,2) =
    30    30    30
    30    30    30
    30    30    30
Z(:,:,3) =
    40    40    40
    40    40    40
    40    40    40
```

So we get 3D-matrices and from the next page we see that when W is computed, x varies faster than y which changes faster than z. Had I used `meshgrid` the order would have been y, x, z, which is less regular.

```
>> [X(:), Y(:), Z(:)] % I have added blank lines
ans =
    0.1000   -1.0000   20.0000  % (x1, y1, z1)
    0.2000   -1.0000   20.0000  % (x2, y1, z1)
    0.3000   -1.0000   20.0000  % (x3, y1, z1)

    0.1000         0   20.0000  % (x1, y2, z1)
    0.2000         0   20.0000  % (x2, y2, z1)
    0.3000         0   20.0000  % (x3, y2, z1)

    0.1000    1.0000   20.0000  % (x1, y3, z1)
    0.2000    1.0000   20.0000  % (x2, y3, z1)
    0.3000    1.0000   20.0000  % (x3, y3, z1)

    0.1000   -1.0000   30.0000  % (x1, y1, z2)
    0.2000   -1.0000   30.0000  % etc.
    0.3000   -1.0000   30.0000

    0.1000         0   30.0000
    0.2000         0   30.0000
    0.3000         0   30.0000

    0.1000    1.0000   30.0000
    0.2000    1.0000   30.0000
    0.3000    1.0000   30.0000

    0.1000   -1.0000   40.0000
    0.2000   -1.0000   40.0000
    0.3000   -1.0000   40.0000

    0.1000         0   40.0000
    0.2000         0   40.0000
    0.3000         0   40.0000

    0.1000    1.0000   40.0000
    0.2000    1.0000   40.0000
    0.3000    1.0000   40.0000
```

## A more general format, using XML

XML, “Extensible Markup Language”, is a language which can be used to transport and store data. It can be used to create markup languages, such as HTML (a language defining how text and images should be displayed). Note that HTML was designed to display data defining the size and position of text for example. XML does not know about layout.

In this XML-example we define our own tags to structure some data.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- This is a comment. -->
<!-- The first line is an XML declaration, defining
     version and encoding -->
<course>
<student>
Thomas Ericsson
</student>
<student>
Karin Andersson
</student>
</course>
```

XML is case sensitive, `<student>Thomas Ericsson</Student>` is illegal.
In the following example we use our own attributes as well:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<course>
<student sex="male">   <!-- " " or ' ' -->
Thomas Ericsson
</student>
<student sex="female">
Karin Andersson
</student>
</course>
```

This is all we need to know about XML (but one can learn more). Here comes a simple data file, **xml_ex.vts** (note **vts**), in XML-format. The line numbers are not part of the file.

```
1   <?xml version="1.0" encoding="iso-8859-1" ?>
2   <VTKFile type="StructuredGrid" version="0.1">
3
4    <StructuredGrid WholeExtent="0 1 0 1 0 2">
5     <Piece Extent="0 1 0 1 0 2">
6
7      <PointData>
8       <DataArray type="Float32" Name="temp"
9                  format="ascii">
10       1 2 3 4 5 6 7 8 9 10 11 12
11      </DataArray>
12     </PointData>
13
14     <CellData>
15     </CellData>
16
17     <Points>
18      <DataArray type="Float32"
19                 NumberOfComponents="3" format="ascii">
20   0.4 0.1 0.0  1.1 0.0 0.0   0.0 0.3 0.0   0.9 1.0 0.0
21   0.3 0.2 1.0  1.0 0.0 1.0   0.0 1.1 0.7   0.9 1.0 1.0
22   0.2 0.3 1.9  1.0 0.0 1.7   0.1 1.0 1.6   1.0 1.0 2.0
23      </DataArray>
24     </Points>
25
26    </Piece>
27   </StructuredGrid>
28  </VTKFile>
```

The file describes a structured grid. Think of producing a grid using Matlab’s `ndgrid`, and then perturbing the points, but not so much that cells overlap or intersect. [Image].

See the ParaView-tutorial, page 2-3, for other types of grids.

Lines 17-24 define the vertices in the structured grid.
Line 4 defined min/max in each coordinate direction.
Lines 7-12 describe a temperature defined at the vertices.