Writing a File Format Reader for VisIt

Advanced VisIt Tutorial / SC12

Monday, November 12, 2012

Materials by VisIt Team

Lawrence Livermore National Laboratory

LLNL-PRES-XXXXXX

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

VisIt's Data Model

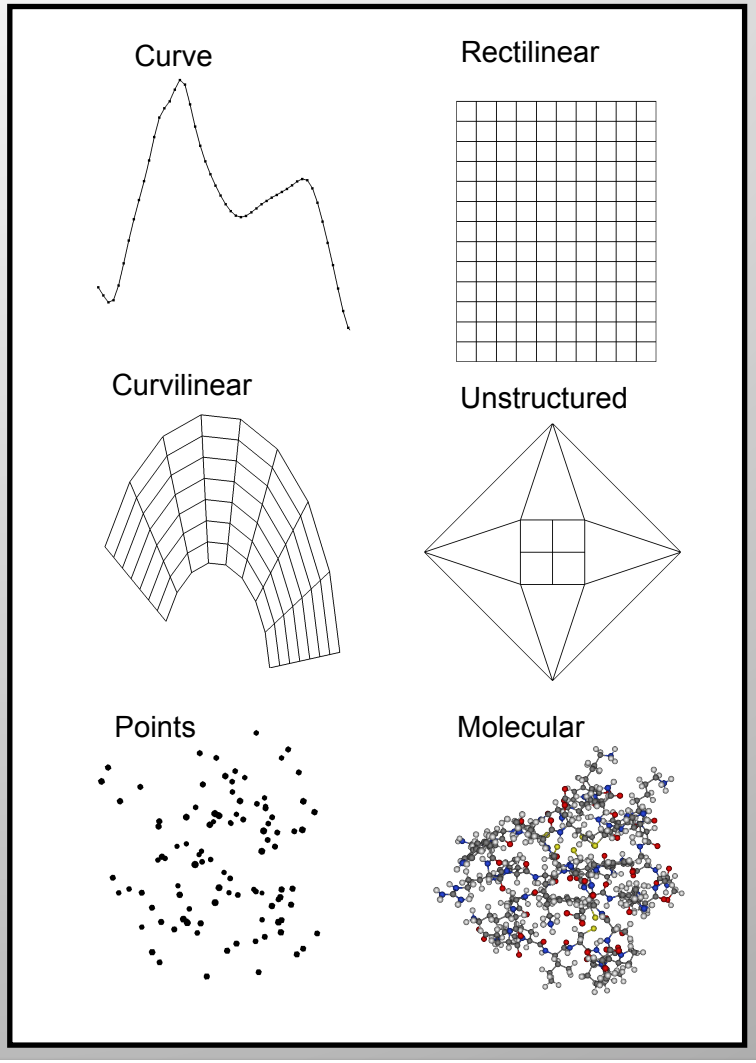
In order to start, we must understand VisIt's Data Model

- A very rich data model
  - Closer to the "computational model"
- Internally implemented with VTK
- Many conventions built on top of VTK

Meshes

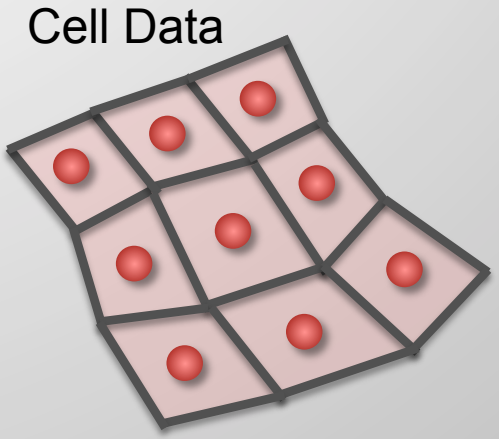
- All data in VisIt lives on a mesh
- Discretizes space into points and cells
  - (1D, 2D, 3D) + time
  - Mesh dimension need not match spatial dimension *(e.g. 2D surface in 3D space)*
- Provides a place for data to be located
- Defines how data is interpolated

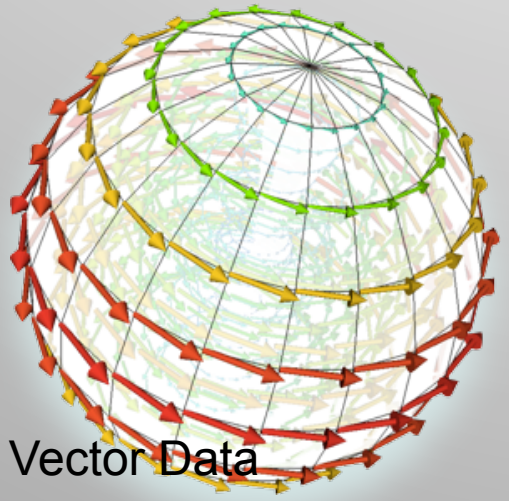
Mesh Types

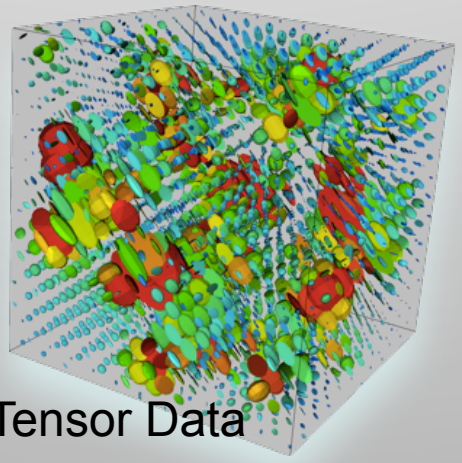


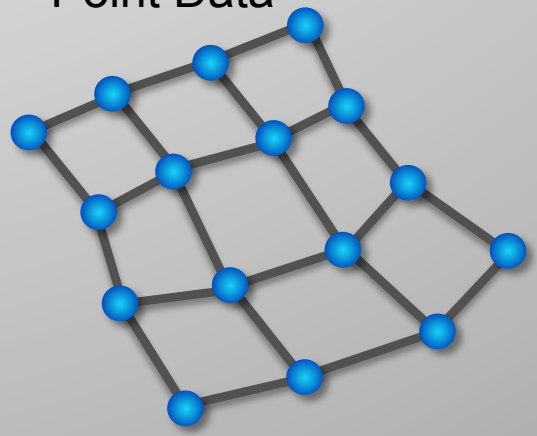
Variables

- Scalars, Vectors, Tensors
- Sits on points or cells of a mesh
  - Points: linear interpolation
  - Cells: piecewise constant
- Can have different dimensionality than the mesh (e.g. 3D vector data on a 2D mesh)



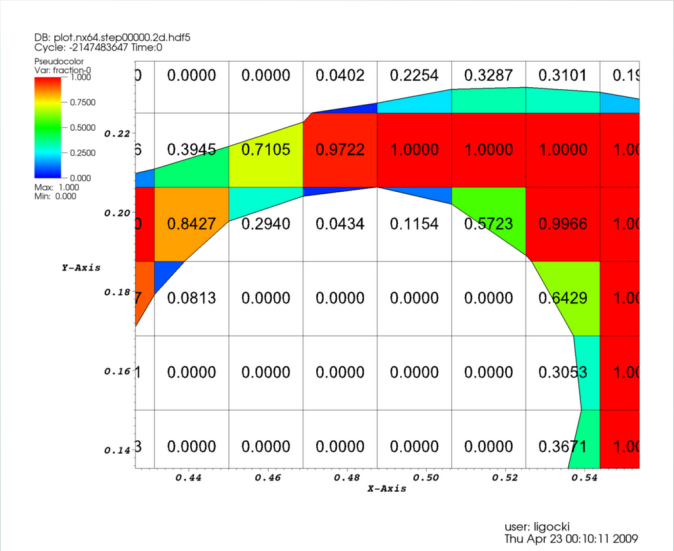

Vector Data

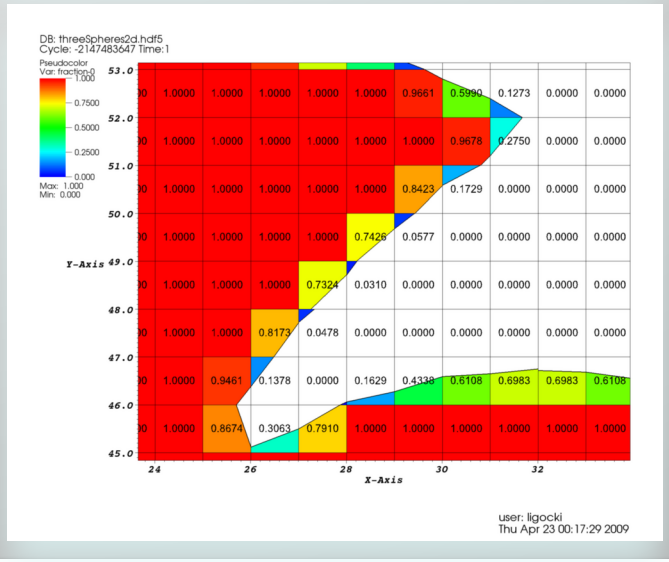

Tensor Data



Materials

- Describes disjoint spatial regions at a sub-grid level
- Volume/area fractions
- VisIt will do high-quality sub-grid material interface reconstruction



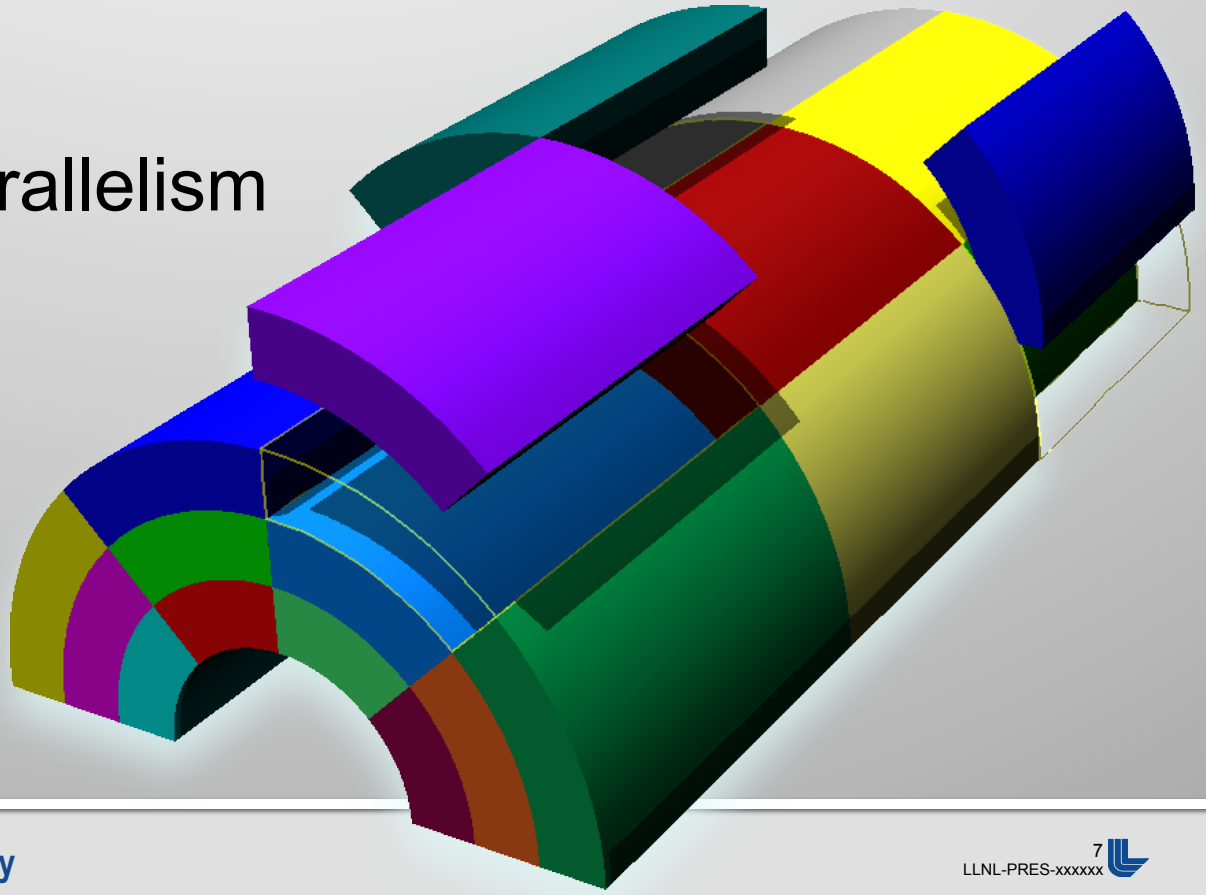


Species

- Similar to materials, describes sub-grid variable composition
  - Example: *Material "Air" is made of species "N_2" ,"O_2", "Ar", "CO_2", etc.*
- Used for mass fractions
- Generally weights other scalars (e.g. partial pressure)

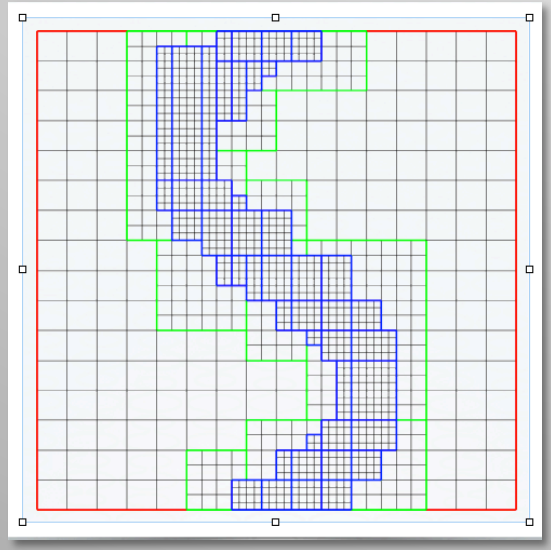
Parallel Meshes

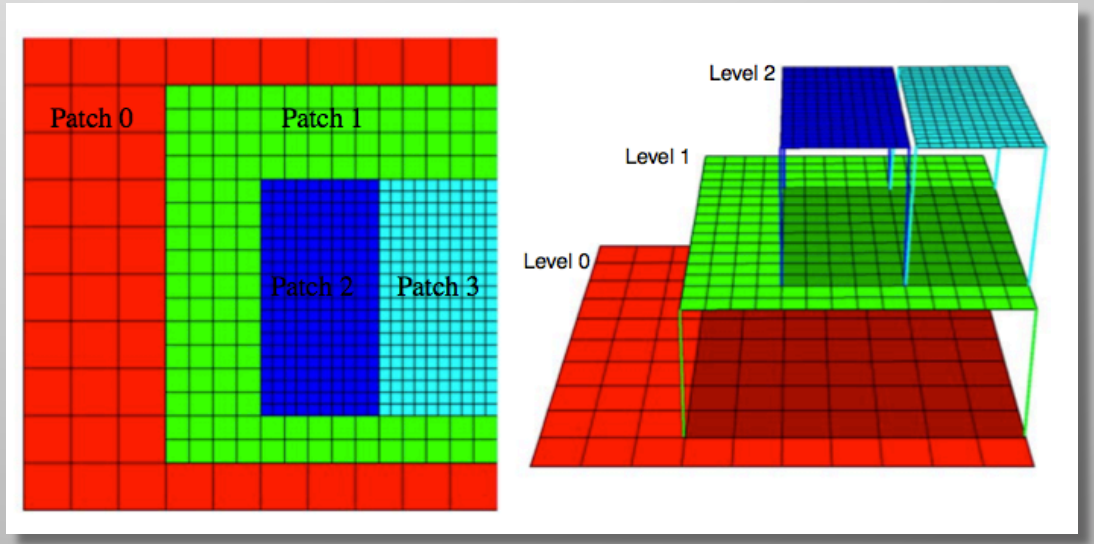
- Provides aggregation for meshes
- A mesh may be composed of large numbers of mesh “blocks”
- Allows data parallelism



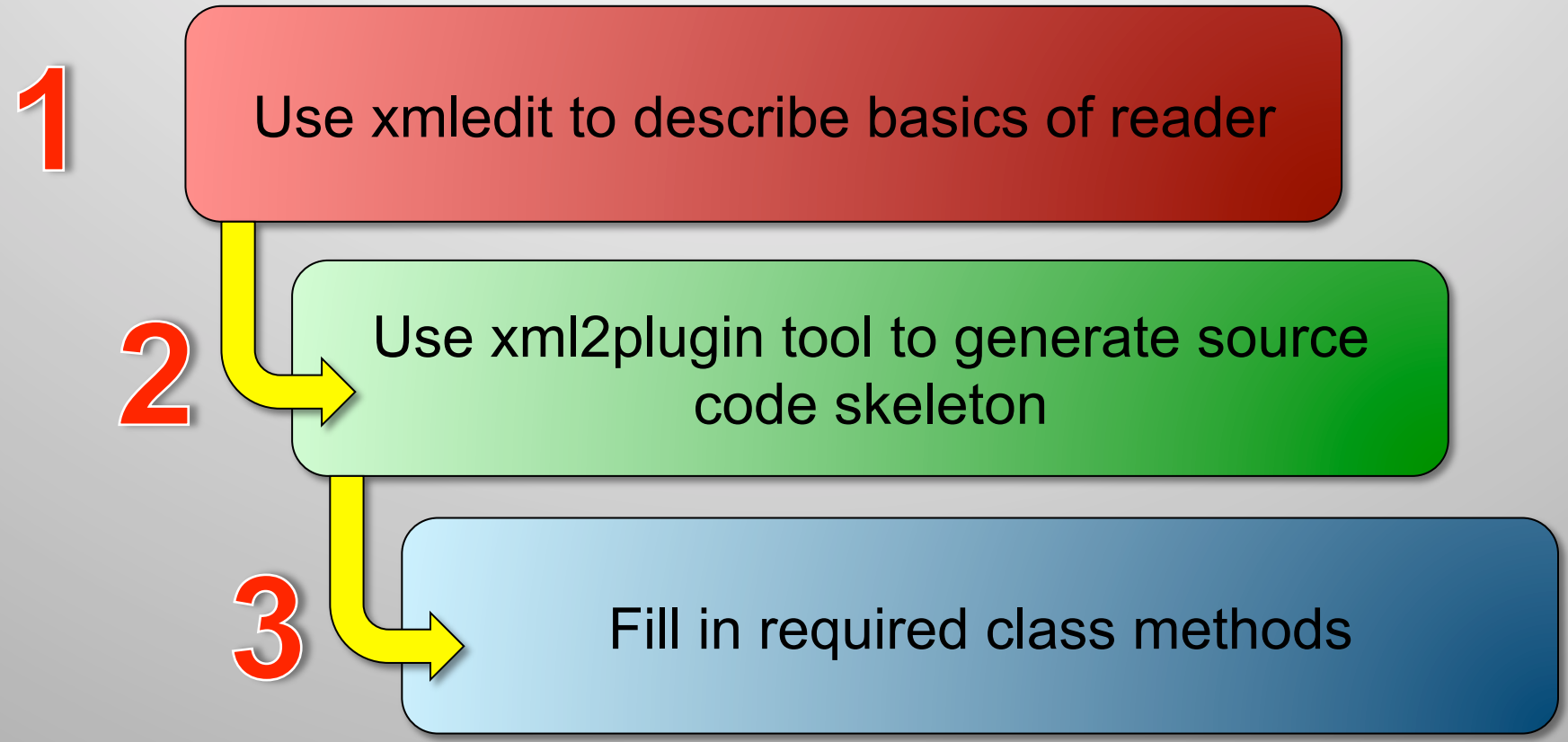
AMR meshes

- Mesh blocks can be associated with patches and levels
- Allows for aggregation of meshes into AMR hierarchy levels



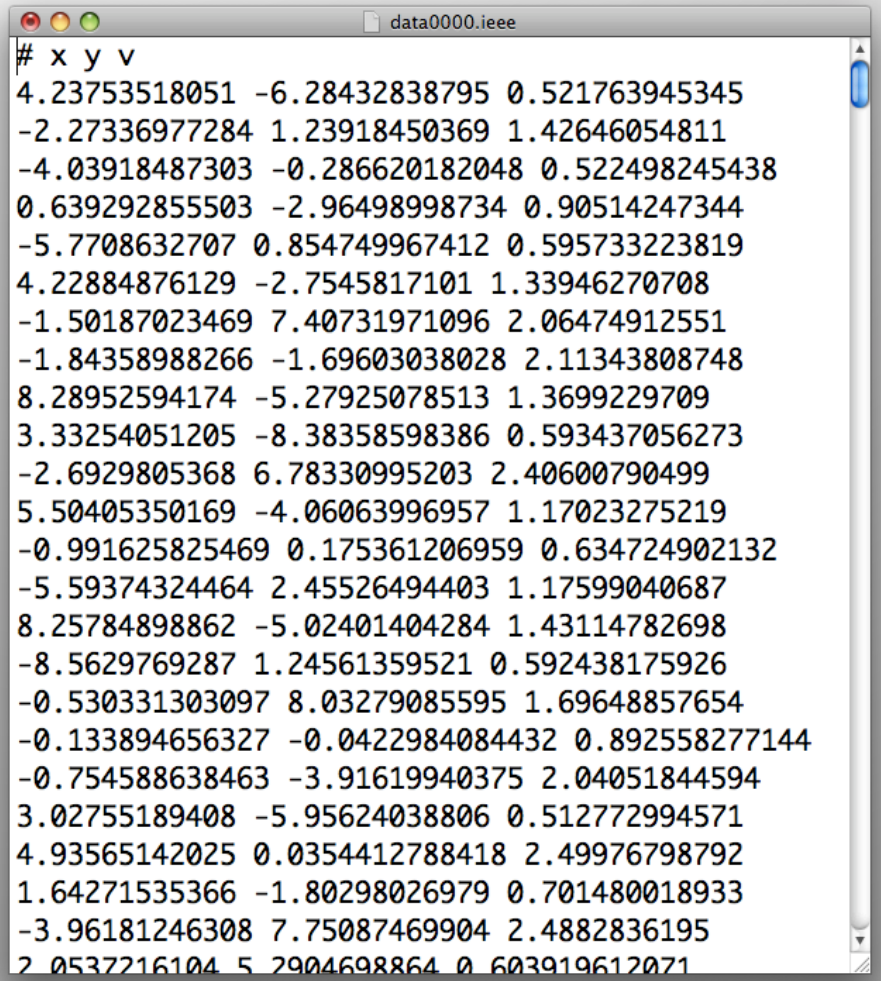


Three Basic Steps to Create Reader



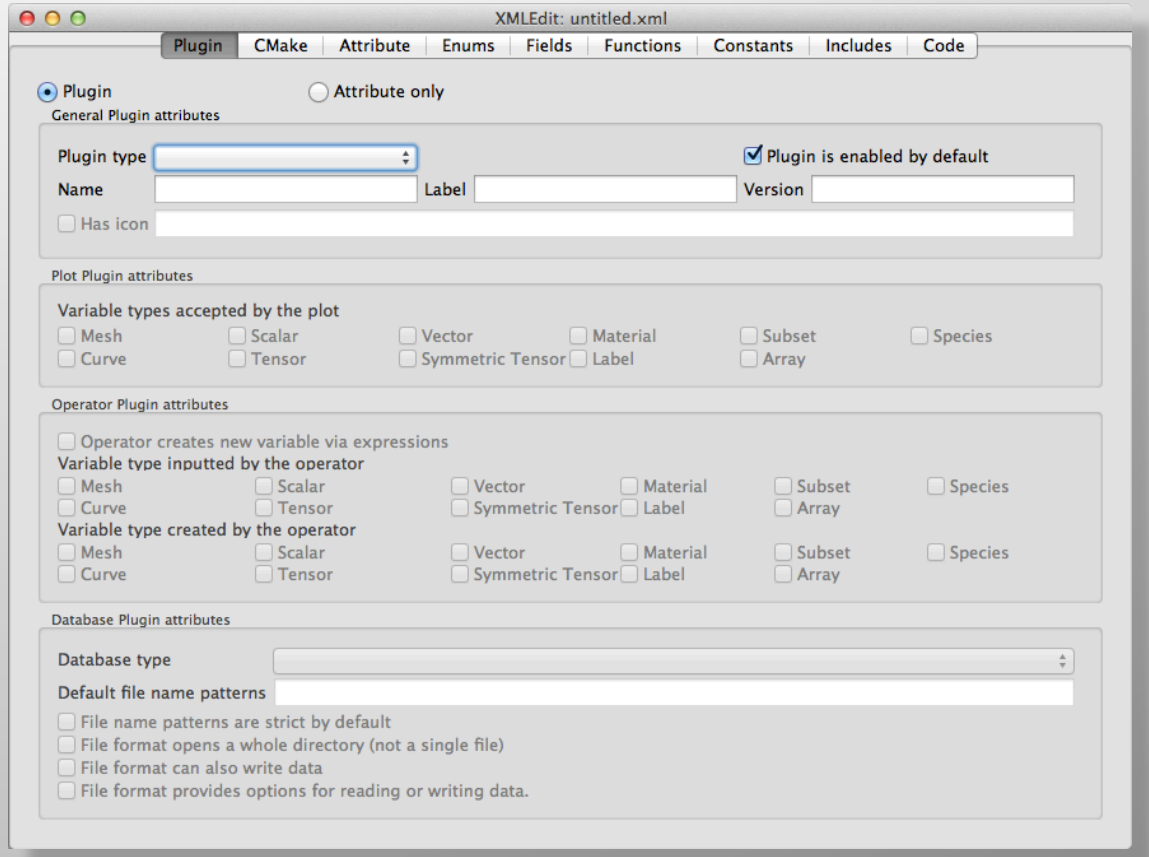
We will make an “XYV” plugin

- Reads single text file of 2D points with data
- Each time step exists in a separate file
- One header line that gives the names of the variables
- All lines after header contain 3 floating point numbers: *x y v*



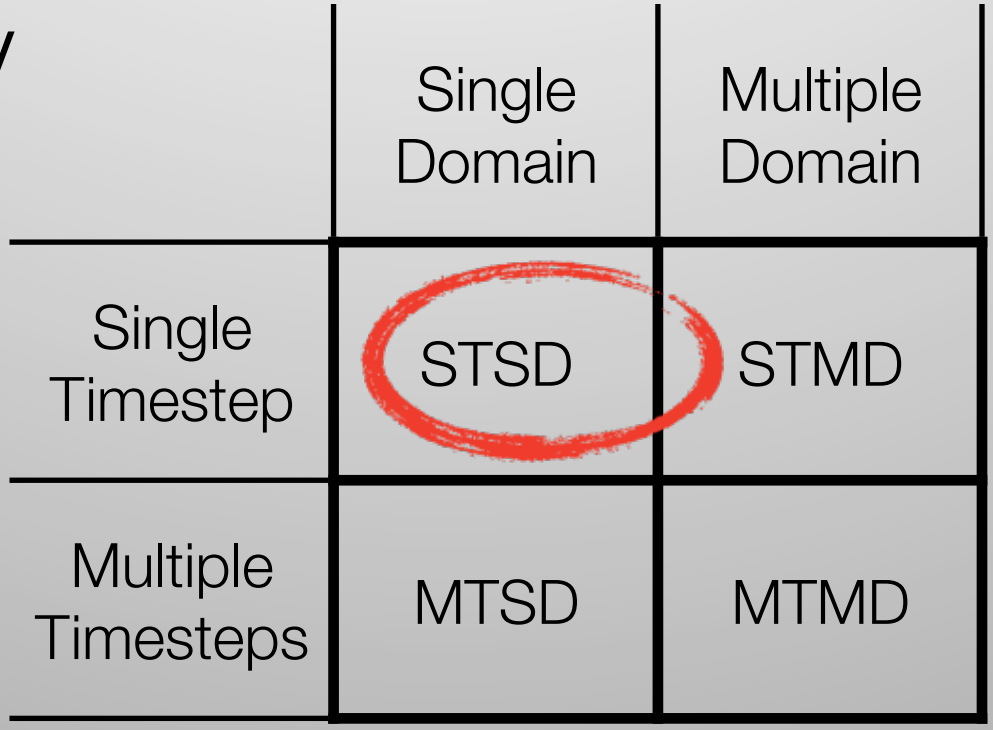
XmlEdit

- GUI tool to edit plugin XML descriptions
- Used to define type of database, filename extensions, etc.
- Creates XML file that describes your reader

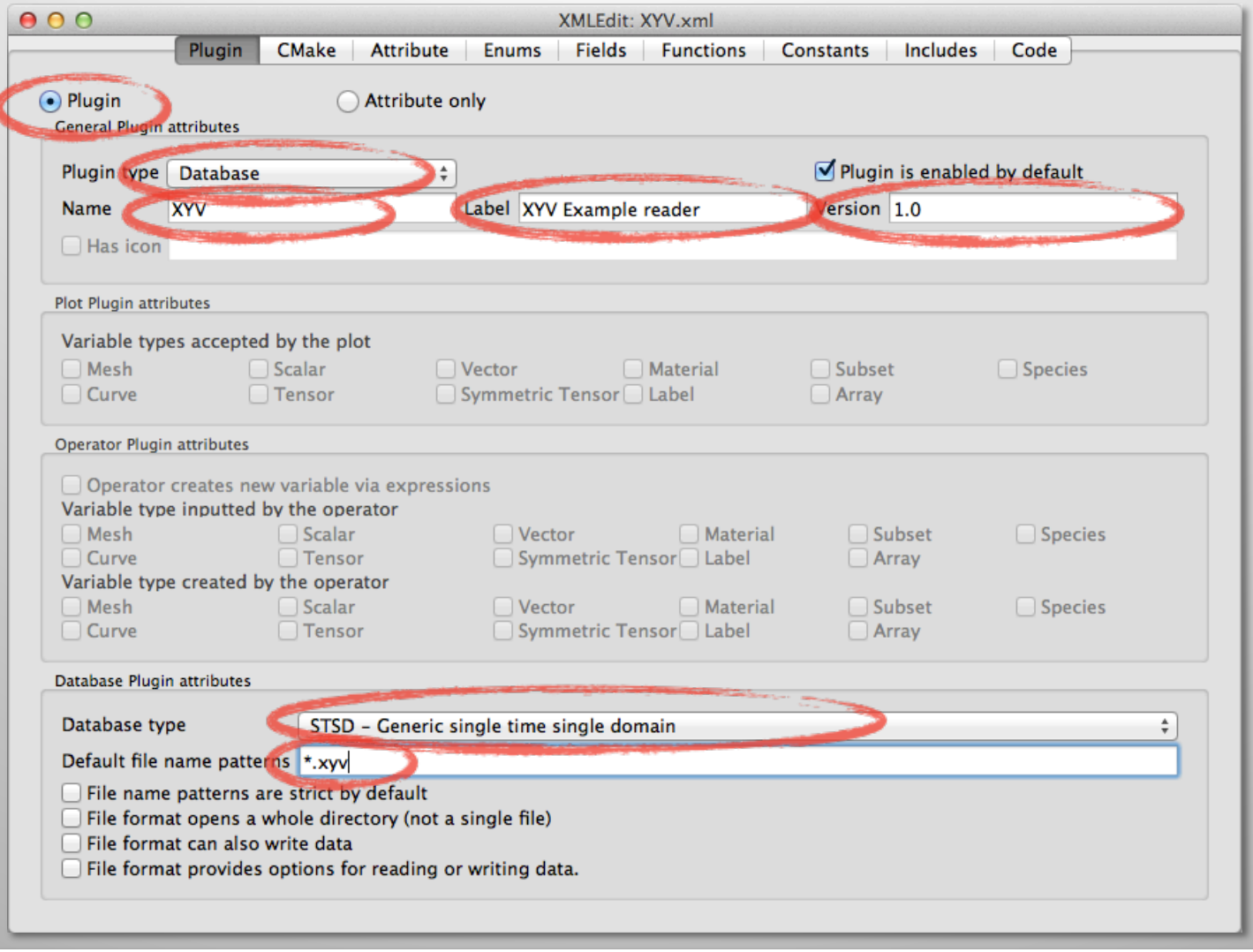


Types of Database Plugins

- Two axes: domains and time
- Moving down and to the right adds complexity



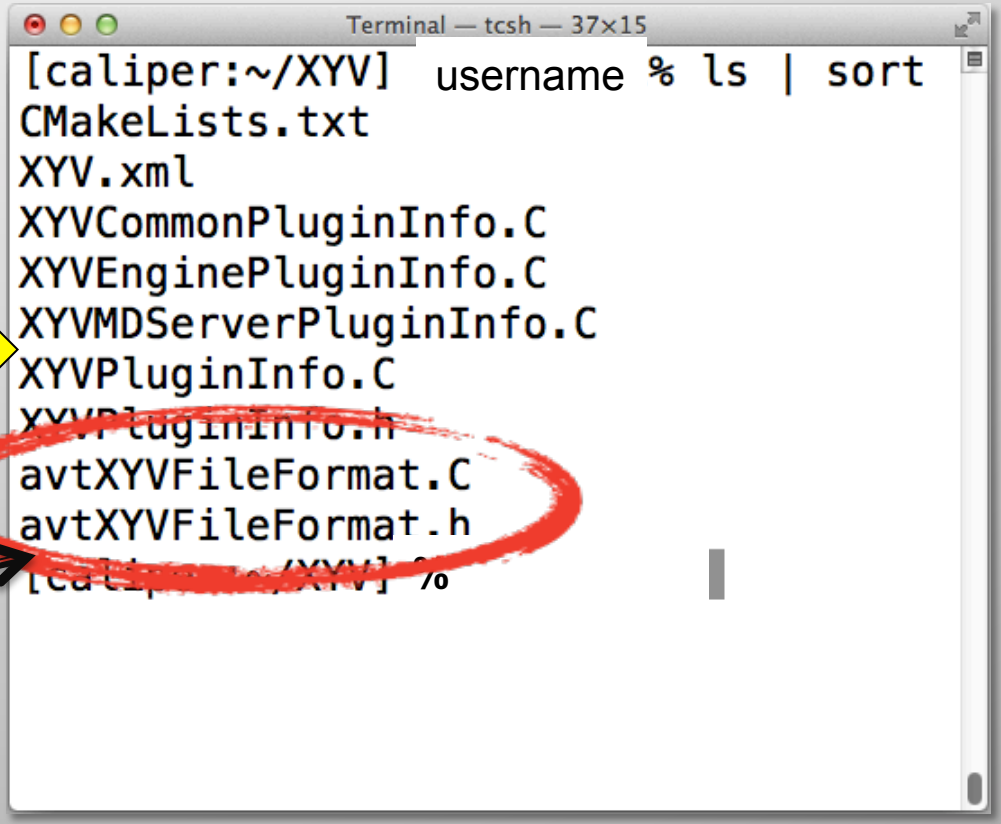
Run XmlEdit and save XYV.xml



Generate Plugin Skeleton

- xml2plugin takes your XML file and generates many files

```
% xml2plugin XYV.xml
```



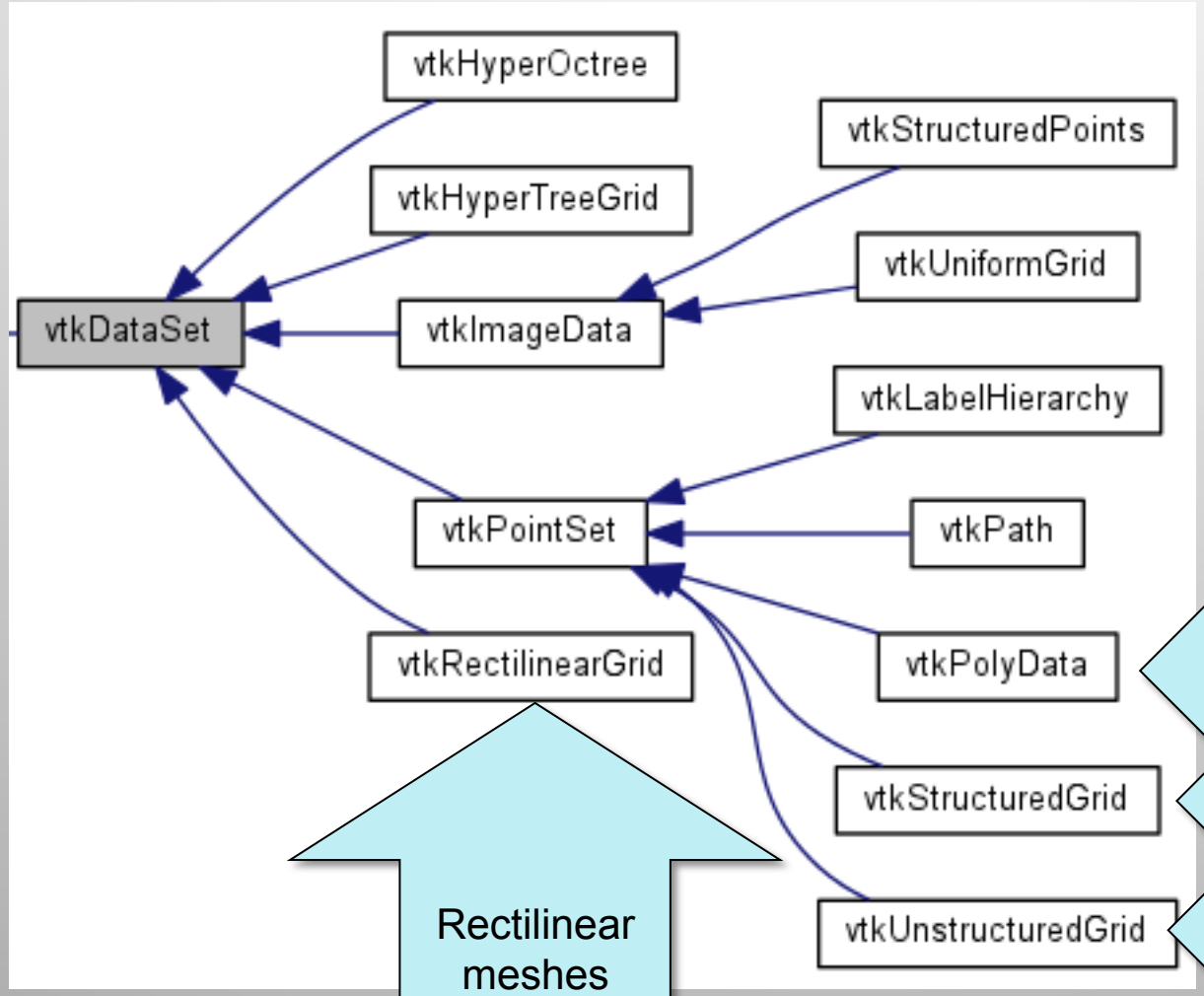
The files we'll customize

Fill in methods for avtXYVFileFormat

- You need to write code to read the file and populate VTK objects
- See ***Getting Data Into VisIt*** manual
- Templates to create VTK objects at:
  - http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/DatabasePlugin/

Method	Description
Constructor	Class constructor
Destructor	Class destructor
PopulateDatabaseMetaData	Fill in metadata that describes the meshes and variables present in the data file
GetMesh	Return vtkDataSet subclass containing the mesh
GetVar	Return vtkDataArray subclass containing variable
GetVectorVar	Just return GetVar()

vtkDataSet is the base class for many meshes in VTK



Create and fill out the appropriate VTK class for the mesh you want to return from your plugin

The Plan

- Add vtkUnstructuredGrid called *grid* to contain the X,Y point mesh
- Add vtkFloatArray called *var* to contain the V value
- Read the file in a ReadData() method
- Call ReadData() from GetMesh() and return reference to *grid*
- Call ReadData() from GetVar() and return reference to *var*

PopulateDatabaseMetaData

```
void
avtXYVFileFormat::PopulateDatabaseMetaData(avtDatabaseMetaData *md)
{
    ReadHeader();

    avtMeshMetaData *mesh = new avtMeshMetaData;
    mesh->name = "mesh";
    mesh->meshType = AVT_POINT_MESH;
    mesh->blockOrigin = 0;
    mesh->spatialDimension = 2;
    mesh->topologicalDimension = 0;
    mesh->hasSpatialExtents = false;
    md->Add(mesh);

    if (varFound)
        AddScalarVarToMetaData(md, varname, "mesh", AVT_NODECENT, NULL);
}
```

Describe the mesh

GetMesh

```
vtkDataSet *
avtXYVFileFormat::GetMesh(const char *meshname)
{
    ReadData();

    // Update the pointer count and return;
    grid->Register(NULL);
    return grid;
}
```

GetVar

```
vtkDataArray *
avtXYVFileFormat::GetVar(const char *varname)
{
    ReadData();

    // Set the data array to our read variable.
    vtkDataArray *rv = var;

    // Adjust the pointer count and return.
    rv->Register(NULL);
    return rv;
}
```

Custom method: ReadData

```
void avtXYVFileFormat::ReadData()
{
}
```

Read all lines in file
Store values in x,y,v vectors

Use x,y vectors to make vtkPoints

Use vtkPoints to make vtkUnstructuredGrid

Use v vector to make vtkFloatArray

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
vector<float> x;
vector<float> y;
vector<float> v;
char line[1024];
debug4 << filename << ": Processing all lines" << endl;
while (!ifile.eof())
{
    debug4 << filename << ": Reading line" << endl;
    ifile.getline(line, 1024);
    float xv, yv, vv;
    if (varFound)
        sscanf(line, "%f %f %f", &xv, &yv, &vv);
    else
        sscanf(line, "%f %f", &xv, &yv);
    x.push_back(xv);
    y.push_back(yv);
    if (varFound)
        v.push_back(vv);
}
```

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
    int npts = x.size();

    // Create vtkPoints structure for the points.
    vtkPoints *points = vtkPoints::New();
    points->SetNumberOfPoints(npts);
    for (int i = 0; i < npts; i++)
        points->SetPoint(i, x[i], y[i], 0);
```

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
// Now create a vtkUnstructuredGrid for the real mesh.
grid = vtkUnstructuredGrid::New();
grid->SetPoints(points);
points->Delete();

// Grids have to have cells to define them.
grid->Allocate(npts);
vtkIdType oneVertex[1];
for (int i = 0; i < npts; i++)
{
    oneVertex[0] = i;
    grid->InsertNextCell(VTK_VERTEX, 1, oneVertex);
}
```

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

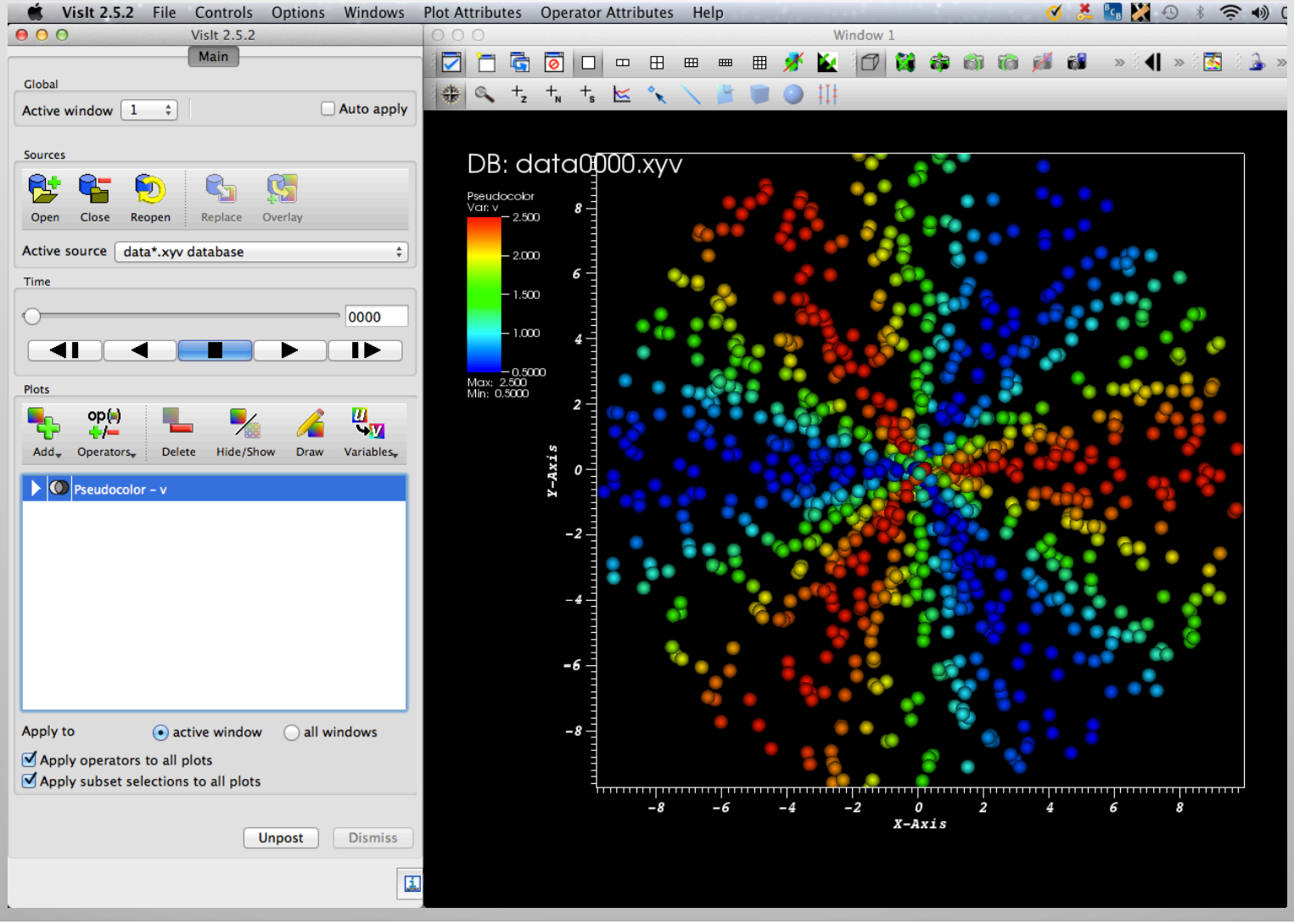
```
    // If we have a variable, get it.
    if (varFound)
    {
        var = vtkFloatArray::New();
        var->SetNumberOfTuples(npts);
        for (int i = 0; i < npts; i++)
            var->SetTuple1(i, v[i]);
    }

    fileRead = true;
}
```

Build and Run

```
Build commands:
% cd XYV
% cmake .
% make
```

- Make will automatically install your plugin to your ~/.visit directory
- VisIt will load your plugin at launch

Run


VisIt 2.5.2
File
Controls
Options
Windows
Plot Attributes
Operator Attributes
Help
VisIt 2.5.2
Main
Global
Active window
1
Auto apply
Sources
Open
Close
Reopen
Replace
Overlay
Active source
data*.xyv database
Time
0000
Plots
Add
Operators
Delete
Hide/Show
Draw
Variables
Pseudocolor – v
Apply to
active window
all windows
Apply operators to all plots
Apply subset selections to all plots
Unpost
Dismiss
Window 1
DB: data0000.xyv
Pseudocolor
Var: v
2.500
2.000
1.500
1.000
0.5000
Max: 2.500
Min: 0.5000
Y-Axis
X-Axis