VisIt: Visualization and Analysis using Python

Hank Childs  VisIt developers from
Brad Whitlock NNSA/ASC, Office of Science/SciDAC,
Cyrus Harrison Office of Nuclear Energy/AFCI

Presented at: 27B element
SIAM CSE09 Rayleigh-Taylor Instability
Miami, Florida (MIRANDA, BG/L)
March 5, 2009

Work performed under the auspices of the U.S.
Department of Energy by Lawrence Livermore
National Laboratory under contract W-7405-Eng-48.

This is UCRL-PROP-411103
VisIt is a richly featured, turnkey application

- VisIt is an open source, end user visualization and analysis tool for simulated and experimental data
  - Used by: physicists, engineers, code developers, vis experts
  - >20 simulation codes & 300 users at LLNL,
    >50K downloads on web
- R&D 100 award in 2005
- ~50 person years of effort
Terribly Named!! Intended for more than just visualization!

- Data Exploration
- Presentations
- Comparative Analysis
- Visual Debugging
- Quantitative Analysis
VisIt contains a rich feature set.

- **Meshes**: rectilinear, curvilinear, unstructured, point, AMR
- **Data**: scalar, vector, tensor, material, species
- **Dimension**: 1D, 2D, 3D, time varying
- **Rendering (~15)**: pseudocolor, volume rendering, hedgehogs, glyphs, mesh lines, etc…
- **Data manipulation (~40)**: slicing, contouring, clipping, thresholding, restrict to box, reflect, project, revolve, …
- **File formats (~85)**
- **Derived quantities**: >100 interoperable building blocks +,-,*,/, gradient, mesh quality, if-then-else, and, or, not
- **Many general features**: position lights, make movie, etc
- **Queries (~50)**: ways to pull out quantitative information, debugging, comparative analysis
VisIt employs a parallelized client-server architecture.

- **Client-server observations:**
  - Good for remote visualization
  - Leverages available resources
  - Scales well
  - No need to move data

- **Additional design considerations:**
  - Plugins
  - Multiple UIs: GUI (Qt), CLI (Python), more…
VisIt’s primary Python usage is to automate existing “filters”.

• Python language is used very much like the GUI
  – Set up filters, order execution, query results, etc.
• VisIt can be run entirely in batch using Python CLI
• GUI and CLI can be run concurrently
• Learning mode for CLI
  – Perform actions in GUI and equivalent CLI commands are generated.
• Python callbacks can be connected to all VisIt events.

• We are currently implementing an alternate mode for using Python that enables direct access to data.
  – Called “Python Filters” and discussed later in this presentation.
There are many example scripts available at the visitusers.org Wiki

- Animating an isosurface

```python
noise_file = "./noise.silo"
# For you:
# noise_file = "/path/to/visit/data/noise.silo"

OpenDatabase(noise_file)
AddPlot("Pseudocolor", "hardyglobal")
iso_att = IsosurfaceAttributes()
iso_att.contourMethod = iso_att.Value
AddOperator("Isosurface")
DrawPlots()

for i in range(30):
    iso_att.contourValue = (2 + 0.1*i)
SetOperatorOptions(iso_att)
# For moviemaking, you'll need to save off the image
# SaveWindow()"
There are many example scripts available at the visitusers.org Wiki

Saving Min Max and Average values over time

This function saves a plot's minimum, maximum, and average values over time to curve files that can be read in by VisIt.

```python
def SaveMinMaxAvgCurves(minname, maxname, avgnamex):
    minfile = open(minname, "wt")
    maxfile = open(maxname, "wt")
    avgfile = open(avgnamex, "wt")
    w = GetWindowInformation()
    cts = w.timeSliderCurrentStates[w.activeTimeSlider]
    for i in range(TimeSliderGetNStates()):
        SetTimeSliderState(i)
        Query("Time")
        t = GetQueryOutputValue()
        Query("MinMax")
        minfile.write("%g %g\n" % (t, GetQueryOutputValue()[0]))
        maxfile.write("%g %g\n" % (t, GetQueryOutputValue()[1]))
        Query("Variable Sum")
        sum = GetQueryOutputValue()
        Query("NumZones")
        nzones = GetQueryOutputValue()
        avgfile.write("%g %g\n" % (t, sum/nzones))
    minfile.close()
    maxfile.close()
    avgfile.close()
    SetTimeSliderState(cts)
SaveMinMaxAvgCurves("min.curve", "max.curve", "avg.curve")
```
There are many example scripts available at the visitusers.org Wiki

```python
def TakeMassPerSlice():
    DeleteAllPlots()
    AddPlot("Pseudocolor", "chromeVf")
    AddOperator("Slice")
    DrawPlots()
    f = open("mass_per_slice.ultra", "w")
    f.write("# mass_per_slice\n")
    for i in range(50):
        intercept = -10 + 20*(i/49.)
        s = SliceAttributes()
        s.axisType = s.XAxis
        s.originType = s.Intercept
        s.originIntercept = intercept
        SetOperatorOptions(s)
        Query("Weighted Variable Sum")
        t2 = GetQueryOutputValue()
        str = "%.15e %.15e\n" % (intercept, t2)
        f.write(str)
    f.close()

TakeMassPerSlice()
DeleteAllPlots()
OpenDatabase("mass_per_slice.ultra")
AddPlot("Curve", "mass_per_slice")
DrawPlots()
```
VisIt Python Filters: Introduction

- VisIt provides extensive scripting control via a Python command line interface but no direct mechanism for programmatic data manipulation outside of C++.
- To address this we are developing a Python runtime filter infrastructure that provides:
  - Direct dataset access/manipulation via VTK python wrappers.
  - A MPI module enabling data parallel algorithms
  - Integration with VisIt’s pipeline contract
- Python was selected to fill this role because it enables:
  - Runtime prototyping/modification of data filters.
  - Reduced development time for special purpose/one-off filters.
  - A path to advanced custom post-processing
VisIt Python Filters: Filter Roles

- VisIt provides three major categories of data manipulation filters:
  - Expressions – Derivation of new quantities (Gradients, etc)
  - Operators – Mesh manipulation (Revolution, clipping, etc)
  - Queries – Data Summarization (Statistics, integration, etc)

- The new Python filter infrastructure will provide base classes for each of these categories that allow users to quickly extend VisIt’s capabilities.

- The infrastructure may also be useful for data generation via VisIt’s Database interfaces.
from math import sin, pi

class MyExpression(SimplePythonExpression):
    def __init__(self):
        SimplePythonExpression.__init__(self)
        self.point_var = False
        self.output_dim = 1
    def derive_variable(self, ds_in, domain_id):
        ds_bounds = ds_in.GetBounds()
        x_ext = ds_bounds[1] - ds_bounds[0]
        ncells = ds_in.GetNumberOfCells()
        res = vtk.vtkFloatArray()
        res.SetNumberOfComponents(1)
        res.SetNumberOfTuples(ncells)
        for i in xrange(ncells):
            cell = ds_in.GetCell(i)
            bounds = cell.GetBounds()
            xv = bounds[0] + bounds[1] / 2.0
            res.SetTuple1(i, 0.25 * (sin(xv*3*pi/x_ext) + sin(yv * 3*pi / y_ext)))
        return res

py_filter = MyExpression
import mpicom

class CellAvgQuery(PythonQuery):
    def __init__(self):
        PythonQuery.__init__(self)

    def pre_execute(self):
        # init vars used to compute the average
        self.var_name = self.attributes["VariableNames"][0]
        self.total_ncells = 0
        self.total_sum = 0.0

    def execute(self, ds_in, domain_id):
        # sum over cell data array passed to query args
        ncells = ds_in.GetNumberOfCells()
        self.total_ncells += ncells
        cell_data = ds_in.GetCellData().GetArray(self.var_name)
        for i in xrange(ncells):
            self.total_sum += cell_data.GetTuple1(i)

    def post_execute(self):
        # communicate total sum & total # of cells
        # calculate average and set results
        self.total_ncells = mpicom.sum(self.total_cells)
        self.total_sum = mpicom.sum(self.total_sum)
        result = self.total_sum / ((float)self.total_ncells)
        self.set_result_text("The average value of %s = %f" % (self.var_name, result))
        self.set_result_value(result)

py_filter = CellAvgQuery
Outline of the rest of this talk

- VisIt the project:
  - Software engineering in VisIt
  - Getting your data into VisIt
  - VisIt’s developer community
  - Data analysis directions in VisIt
The VisIt team focuses on making a robust, usable product for end users.

- Manuals
  - 300 page user manual
  - 200 page command line interface manual
  - “Getting your data into VisIt” manual
- Wiki for users (and developers)
- Revision control, nightly regression testing, etc
- Executables for all major platforms
- Day long class, complete with exercises

Slides from the VisIt class
Getting Data into VisIt

- Two options:
  - #1, change your I/O routines to write a format friendly with VisIt
    - 85 total readers
    - Silo: established data model, production tools
    - VTK: library with no dependencies to help with this
  - #2, write a custom reader
    - This is done with plugins
    - Well documented
- Manual: “Getting Your Data Into VisIt”
VisIt has a rapidly growing developer community.

- + more partnerships coming soon
- + contributions from people not in the repository
Quantitative analysis means different things to different people.

1) Techniques that span scientific domains (e.g. integration, volumes, surface areas, etc.)
2) Advanced analysis (e.g. connected components, chord length distributions)
3) Physics-based analysis (e.g. hohlraum flux)
Summary

- VisIt is a Python friendly parallel data analysis and visualization tool.
- VisIt is a tool that end users can use right now (… bar data format issues)
  - VisIt contains support for diverse data models
  - VisIt contains requisite feature set for most day-to-day visualization and analysis use.
- The VisIt project is actively seeking co-developers
  - Experimental research
  - Deploy custom functionality to your user community