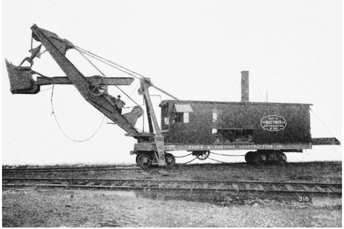
Steamshovel: A New IceCube event viewer

Steve Jackson

12/19/12



Motivation

IceCube has developed at least two event viewers: glshovel and the python event viewer.

- Both semi-abandoned, barely maintained
- Consensus in Madison and Aachen: it's worth starting over

Steve started at IceCube in September; prototyping began in October.

Steamshovel

A scriptable graphical tool for visualizing IceCube data on OS X and Linux.

- Primary user: physicist seeking insight on data, or preparing graphics for presentation
- Secondary users:
    - Student learning about the .i3 data file format and IceCube detector data
    - Professor giving a demonstration

Steamshovel Technology

Steamshovel is a C++ application with an embedded python interpreter.
Required:

- icetray for data interchange
- OpenGL for 3D graphics
- boost::python for application scripting
- Qt (4.6+) for GUI and utilities

Essentially the same requirements as glshovel.
Optional:

- IPython for better scripting interaction
- PyQt4 to use IPython's embedded Qt GUI widget

What About a Scene Graph Library?

Evaluated three possibilities in detail: Panda3D, OpenSceneGraph, and visualizationlibrary.org.

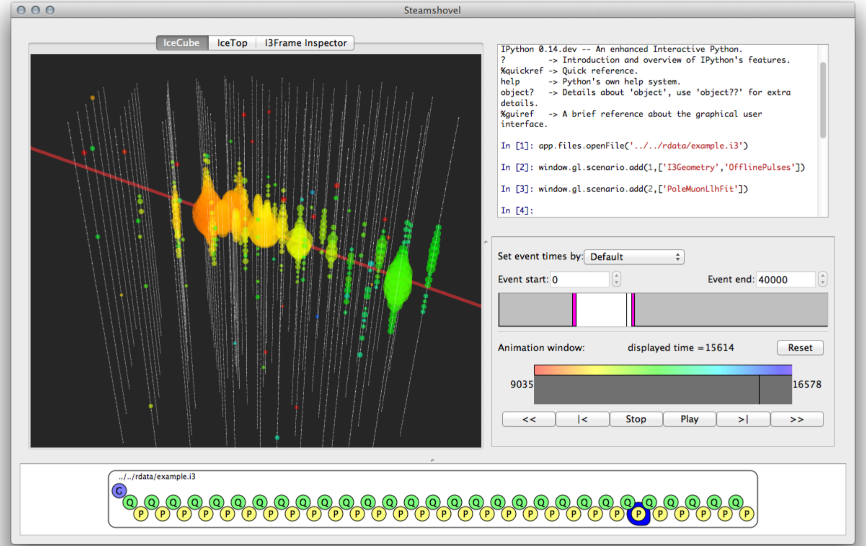
Some disadvantages:

- Heavyweight libraries can create dependency issues
- Complex APIs are difficult to learn (OpenSceneGraph)
- Omni-competent libraries require you to fit their world (Panda3D)
- VL looked promising but appears to have very few actual users
- Steamshovel's developer is already learning icetray and Qt as he goes

So why do we need a scene graph?

- Can draw with advanced/complicated graphics styles
  - But IceCube visualization is not graphically complex
- Provide robust geometry math support (vectors, matrices, quaternions)
  - Qt (and icetray) provide this
- Support 3D text rendering and fonts
  - Qt provides this
- Scene graph gives structure to data on the screen
  - We will create our own, scriptable scene data types
  - Easier for non-experts to understand and maintain

Screenshot



```
IPython 0.14.dev -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
details.
%guiref   -> A brief reference about the graphical user
interface.

In [1]: app.files.openFile('../../rdata/example.i3')

In [2]: window.gl.scenario.add(1,['I3Geometry','OfflinePulses'])

In [3]: window.gl.scenario.add(2,['PoleMuonLlhFit'])

In [4]:
```

Set event times by: Default

Event start: 0

Event end: 40000

Animation window: displayed time =15614

9035

16578

../../rdata/example.i3

Features working today

- glshovel-like rendering of I3Geometry, RecoPulseSeries, and I3Particle (in process)
- Scriptable GUI using Qt introspection
- Loading of multiple, possibly compressed, .i3 files
- dataio-shovel-like frame inspector
- Rendered geometry is "selectable"
  - Selection is performed using OpenGL selection buffer
  - All rendered objects beneath the cursor can be detected, depth sorted
  - Current selection behavior is just printf

Embedded Python

In Steamshovel, C++ and Python communicate across three “bridges”:

- icetray: I3 data interchange between C++ and Python.
- QMeta: GUI scriptability through Qt meta-object system. QObject slots and properties are made automatically accessible to Python.
- Artist bridge: Allow users to create new ways of rendering data in Python.

QMeta example

In C++, Steamshovel's MainWindow class does the following:

```
QMeta::exportObject( this );
QMeta::addToGlobalNamespace( this, "window" );
```

Python may now control the window, e.g.:

```
window.showFullScreen()
w = window.width()
window.windowTitle = ‘Example’
```

No further pybindings are necessary!
IPython's tab completion makes it easy to explore the scriptable API.

About QMeta

- Completely separate from PyQt.
- Only works on types which inheritent QObject.
- Return values from functions not currently supported. I don't anticipate a need for this.
- Custom argument types must be added to a list of Qt custom metatypes (fairly easy). This makes it is possible to pass, e.g. `I3FramePtr` objects through QMeta.

Rendering

Rendering API provides a set of geometric primitives (spheres, lines, text).

Their properties (location, size, color) can vary according to a time parameter, an integer count of nanoseconds.

The chief API is the Artist, which must implement three methods:

- `description`: e.g. "Accumulated Charge"
- `requiredTypes`: e.g. `I3Geometry` and `I3RecoPulseSeriesMap`
- `create`: given the appropriate I3FrameObjects, return geometry objects.

The Artist API

The active artists may be configured by user via scripting or GUI.

Artists persist between selected events for consistent rendering style.

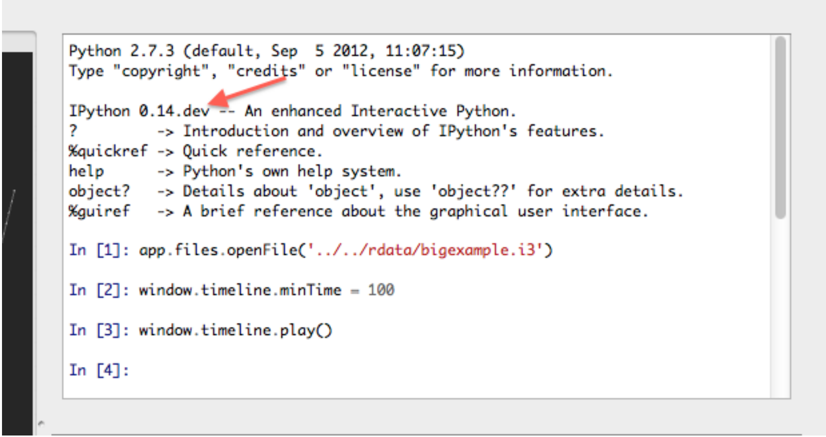
Key feature: Artists may be implemented in Python.

- Visualize icetray data types not known at compile time
- Easy experimentation with visual styles
- Users can create new rendering schemes without modifying project or recompiling

Python never drives OpenGL: more efficient, no need for PyOpenGL

Rendering system exists today; Python bridge is coming soon.

The IPython widget



```
Python 2.7.3 (default, Sep  5 2012, 11:07:15)
Type "copyright", "credits" or "license" for more information.

IPython 0.14.dev -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]: app.files.openFile('../../rdata/bigexample.i3')

In [2]: window.timeline.minTime = 100

In [3]: window.timeline.play()

In [4]:
```

Support for this widget currently requires an unreleased IPython feature with an uncertain future.

We can duplicate the code if needed to work with stable IPython.

Approximate Timeline

- Early January: announce on dataclasses mailing list that early adopters may begin alpha testing.
  - glshovel-like accumulated charge and particle tracks
  - Emphasis on getting early UI feedback
- January-February: finish core technical features
  - GUI for choosing Artists and setting parameters
  - Python bridge for rendering system, Artists written in Python
  - Movement of image data from Python to C++, e.g. to display Matplotlib graphs of DOM waveforms
- Feb-April: implementation v1.0 features
  - Call for beta testers and code review
- May collaboration meeting: presentation to collaboration.
  - Professor-proof .app bundle, user documentation, release of stable v1.0.

Version 1.0 features

- Display waveforms of mouse-selected DOM(s)
- 2D IceTop visualization mode, orthographic camera modes
- Create (high-res) screenshots and movies
- Visualize surface and dust layer; show particle/surface intersection points
- Configurable colors for I3Particles; configurable Artists in general
- Stored user preferences, style templates for various tasks
- Flag and write user-selected "interesting" frames to new file