Steamshovel : an IceCube event viewer

Technical overview

Steve Jackson

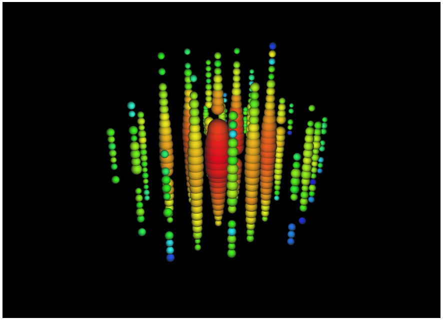
"I devoted several months in privacy to the composition of a treatise on the mysteries of Three Dimensions ... But ... I found myself sadly hampered by the impossibility of drawing such diagrams as were necessary for my purpose ..."
from *Flatland*, by Edwin A. Abbott

IceCube Collaboration Meeting
6 May 2013

Steamshovel

A scriptable graphical tool for visualizing and exploring IceCube data on OSX and Linux.

- In use today by brave early adopters
- Almost beta-ready
- Goal: be useful to normal[1] IceCube people



[1] i.e. non-programmers, non-icetray users

Steamshovel Directory Layout

- `shovelart`
- `shovelio`
- `steamshovel`
- `scripting`

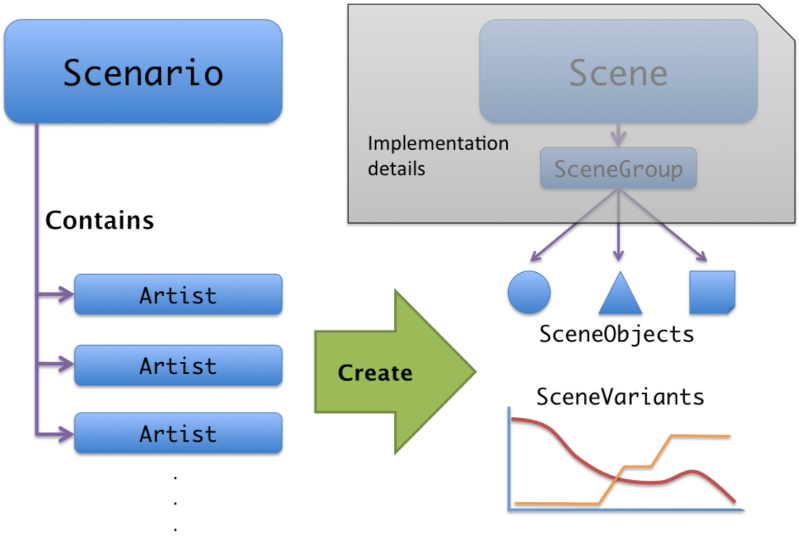
Steamshovel Directory Layout

- `shovelart`
  - Graphics library, with pybindings
- `shovelio`
  - Random access into (compressed) I3 files, like `dataio`
- `steamshovel`
  - Qt-based GUI classes
- `scripting`
  - Utilities for bridging between C++ and Python

Steamshovel Directory Layout

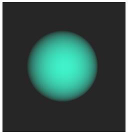
- `shovelart`
  - Graphics library, with pybindings
- `shovelio`
  - Random access into (compressed) I3 files, like `dataio`
- `steamshovel`
  - Qt-based GUI classes
- `scripting`
  - Utilities for bridging between C++ and Python

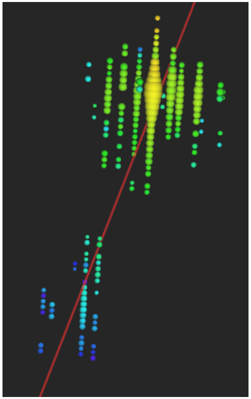
Shovelart API Classes

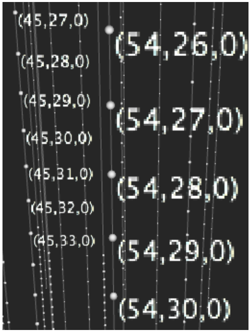


3D SceneObjects

- Spheres
- Lines
- Floating text (ugly)

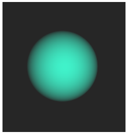


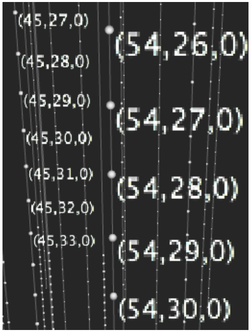


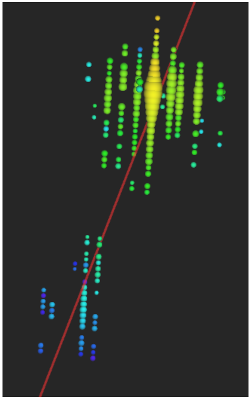


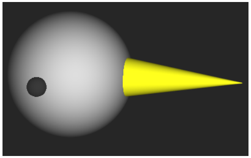
3D SceneObjects

- Spheres
- Lines
- Floating text (ugly)
- Cylinders, cones (uncapped)



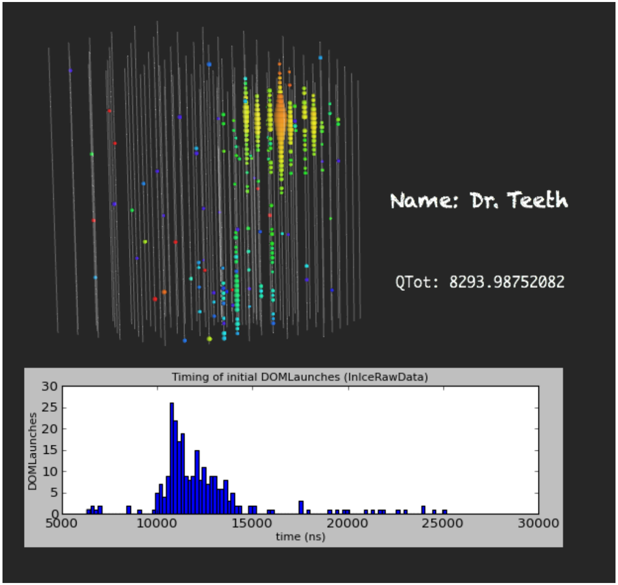






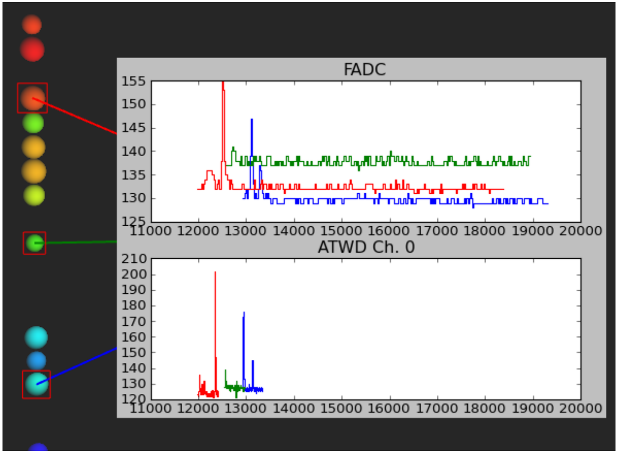
2D (Overlay) SceneObjects

- Text
- Lines
- Images
- Matplotlib plots



2D (Overlay) SceneObjects

- Text
- Lines
- Images
- Matplotlib plots



SceneVariants

The properties of SceneObjects (location, size, color, etc.) are controlled by time-varying functions called SceneVariants.

C++

- `SceneVariant<float>`
- `SceneVariant<vec3d>`
- `SceneVariant<QColor>`

Python

- `VariantFloat`
- `VariantVec3d`
- `VariantQColor`

Built-in variants:

- Constant
- Step function
- Linear interplation

Variants can also be subclassed (in either language) for custom behavior.

Artist Properties

Keys: What data to draw

- String keys for `I3FrameObject`s
- An Artist creates output only when all its keys are valid in the current `I3Frame`

Settings: What style to draw

- Can be any type, but GUI and python bindings exists for:
  - Booleans and integers
  - Floating point ranges (min/max/step/value)
  - `QColor` and shovelart's `ColorMap` type
  - `QFont`

Specifying Key Types: The Simple Way

List required key types, in order:

```
class MyArtist( shovelart.PyArtist ):
  ...
  requiredTypes = [ dataclasses.I3Geometry,
                    dataclasses.I3MCTree ]
  ...
```

Note that Python artists can be used with data types from any project!

Specifying Key Types: The Flexible Way

Specify number of keys, and a key test function:

```
class MyArtist( shovelart.PyArtist ):
  ...
  numRequiredKeys = 2

  def isValidKey( self, frame, key_idx, key ):
     '''
     Return True if frame[key] is valid
     as this artist's key_idx'th key,
     False otherwise
     '''
  ...
```

Specifying Settings

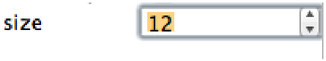
Artist settings are stored as name / value pairs.
Created in constructor:

```
class MyArtist(shovelart.PyArtist):
  ...
  def __init__(self):
    self.defineSettings({'size':12})
  ...
```

Used within create() function:

```
    ...
    size = self.setting('size')
    ...
```

User-controlled via the GUI:



size 12

Creating SceneObjects

```
class PositionBubble( shovelart.PyArtist ):
  requiredTypes = [ dataclasses.I3Position ]
  ...
  def create(self, frame, output):
```

Creating SceneObjects

```
class PositionBubble( shovelart.PyArtist ):
  requiredTypes = [ dataclasses.I3Position ]
  ...
  def create(self, frame, output):
    key = self.keys()[0]
    position = frame[key]
```

Creating SceneObjects

```
class PositionBubble( shovelart.PyArtist ):
  requiredTypes = [ dataclasses.I3Position ]
  ...
  def create(self, frame, output):
    key = self.keys()[0]
    position = frame[key]
    bubblesize = self.setting('size')
    bubblecolor = self.setting('color')
```

Creating SceneObjects

```
class PositionBubble( shovelart.PyArtist ):
  requiredTypes = [ dataclasses.I3Position ]
  ...
  def create(self, frame, output):
    key = self.keys()[0]
    position = frame[key]
    bubblesize = self.setting('size')
    bubblecolor = self.setting('color')
    s = output.addSphere(bubblesize, position)
    s.setColor(bubblecolor)
```

Next Steps in Development

- Finish GUI overhaul
- Steamshovel.app bundle for OSX
- Documentation and (some) tests
- Code review
- Release with metaprojects; retire glshovel

A beta release can happen this month.

Thanks and questions

Many thanks to those who have tested and contributed to this project!

Backup Slides

Scripting Features

Steamshovel is broadly scriptable with python:

- Perform CRUD operations on Artists
- Load files, select frames
- Control OpenGL camera position and drawing styles
- Control application and window objects through their Qt properties and slots

Dependencies

Requirements (like glshovel):

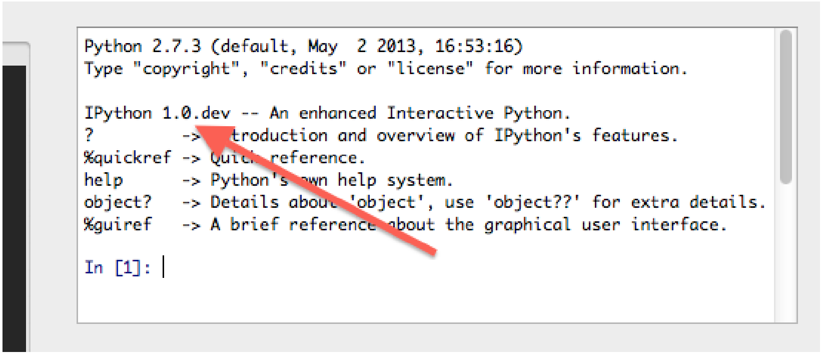
- icetray
- Python
- Qt4 (4.8 recommended)

Optional:

- IPython for a better scripting experience
- matplotlib for plots
- PyQt4 for embedded IPython GUI, separate-window matplotlib plots

About IPython

Embedded IPython GUI widget requires IPython 1.0.dev (or certain versions of the defunct 0.14.dev branch).



```
Python 2.7.3 (default, May  2 2013, 16:53:16)
Type "copyright", "credits" or "license" for more information.

IPython 1.0.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]: 
```

IPython 1.0 to be released July 14, 2013.