How the Large Synoptic Survey Telescope (LSST) is using Python

Robert Lupton
Princeton University

SciPy08, 22nd August, 2008
Outline
Outline

Introduction
Outline

Introduction
The Large Synoptic Survey Telescope (LSST)
Outline

Introduction
The Large Synoptic Survey Telescope (LSST)
What is LSST?
Outline

Introduction
The Large Synoptic Survey Telescope (LSST)
  What is LSST?
  LSST’s Computing Needs
Outline

Introduction
The Large Synoptic Survey Telescope (LSST)
  What is LSST?
  LSST’s Computing Needs
Where are we using Python?
Introduction

Over the last twenty-five years Astronomy has been revolutionized by the introduction of computers and CCD detectors.
Introduction

Over the last twenty-five years Astronomy has been revolutionized by the introduction of computers and CCD detectors.
Introduction

Over the last twenty-five years Astronomy has been revolutionized by the introduction of computers and CCD detectors.
Introduction

Over the last twenty-five years Astronomy has been revolutionized by the introduction of computers and CCD detectors.
Introduction

Over the last twenty-five years Astronomy has been revolutionized by the introduction of computers and CCD detectors.
The Large Synoptic Survey Telescope (LSST)
The Large Synoptic Survey Telescope (LSST)

Synoptic (n.): Of a table, chart, etc.: Pertaining to or forming a synopsis; furnishing a general view of some subject; spec. depicting or dealing with weather conditions over a large area at the same point in time.
The Large Synoptic Survey Telescope (LSST)

The primary mirror will have a diameter of 8.4m
The Large Synoptic Survey Telescope (LSST)

The primary mirror will have a diameter of 8.4m
on Cerro Pachón in Chile
The optical design is unusual:
The optical design is unusual:
The optical design is unusual:
The optical design is unusual:
To make use of such a large field (9.6 deg$^2$) requires a large camera...
To make use of such a large field (9.6 deg$^2$) requires a large camera...
To make use of such a large field (9.6 deg$^2$) requires a large camera...
To make use of such a large field ($9.6 \text{ deg}^2$) requires a large camera...

That’s a 3.2Gpixel camera. The field of view has a diameter of 3.5 degrees.
To make use of such a large field (9.6 deg$^2$) requires a large camera...

That’s a 3.2Gpixel camera. The field of view has a diameter of 3.5 degrees.

Taking an exposure every 15s, that over 800 Mby/s
/*
 * copy the symmetrised image back to the original data region, where
 * it will become the deblending template (the original pixel values
 * are preserved in the parent's atlas image).
 * We must of course only do this within the master_mask
 */
copy_region_within_mask((REGION *)data, sym, mmask,
                        aimage_drow, aimage_dcol);

/*
 * we next want to run the object finder on that symmetrised image: the image
 * is smoothed, and extra peaks rejected -- see improve_template() for details
 *
 * obj1 = objc->color[c1];
 * if(obj1->Flags & OBJECT1_DEBLENDED_AS_PSF) {
 * no need to check template, as we created it as a multiple of PSF
 */
 3 else {
    float threshold = fiparams->frame[c1].ffo_threshold;
    shAssert(obj1->mask == objc->image->mask[c1]);
    phObjMaskDel(obj1->mask); objc->image->mask[c1] = NULL;
    obj1->mask =
      improve_template(mmask, c, rowc, colc, data, aimage_drow, aimage_dcol,
                       scra, scrb, rsize + filtsize, csze + filtsize,
                       fiparams->frame[c1].smooth_sigma, filtsize,
                       npeak_max, smoothed_ai, threshold, ngrow);
    if(obj1->mask == NULL) {
      objc->flags |= "OBJECT1_DETECTED;"
      obj1->flags |= "OBJECT1_DETECTED;"
    } 3
  }

/*
 * we've found the templates in all colours. They are represented by the
 * pixels in the original data region, within the OBJECT1->mask
 * Now go through them looking for objects which we didn't detect
 * in any band: in this case, the object wouldn't have been found at all
 * if it wasn't part of a blend, so dump it.
 * Actually we cannot just dump it here as we've got an array with all the
 * children in it, and we'd have to move the others down. Instead, mark
 * the entire OBJC as not DETECTED, and we'll dump it when we get a chance.
 */
for(c = 0;c < ncolor;c++) {
  objc->flags |= (objc->color[c]->flags & OBJECT1_DETECTED);
} 3
if(!objc->flags & OBJECT1_DETECTED) { /* not detected in any band */
  phAtlasImageDel(*smoothed_ai, 0); *smoothed_ai = NULL;
  deblend.c 38% L1365 CVS-1.128 (C Abbrev)--2:55PM 0.39------------------
Warning: only of interest to Astronomers
Warning: only of interest to Astronomers

- Main survey will cover 20,000 deg²
- Over 300 15s exposures in g, r, i, z, y
- $2\sigma$ depths after a pair of 15s exposures are 23.9 (u), 25.0 (g), 24.7 (r), 24.0 (i), 23.3 (z), 22.1 (y)
- At end of the survey, 26.2, 27.4, 27.6, 26.9, 26.1, and 24.8
LSST’s Computing Needs

All of the LSST software is divided into three parts:
LSST’s Computing Needs

All of the LSST software is divided into three parts:

- The applications codes that process the incoming pixels, resulting in catalogues of detected sources’ properties
LSST’s Computing Needs

All of the LSST software is divided into three parts:

- The applications codes that process the incoming pixels, resulting in catalogues of detected sources’ properties
- The middleware is responsible for marshalling the applications layer over a (large) cluster; it’s responsible for tasks such as disk i/o, load balancing, fault tolerance, and provenance.
LSST’s Computing Needs

All of the LSST software is divided into three parts:

• The applications codes that process the incoming pixels, resulting in catalogues of detected sources’ properties

• The middleware is responsible for marshalling the applications layer over a (large) cluster; it’s responsible for tasks such as disk i/o, load balancing, fault tolerance, and provenance.

• The databases to handle the resulting astronomical catalogues, along with all relevant metadata
LSST’s Computing Needs

All of the LSST software is divided into three parts:

• The applications codes that process the incoming pixels, resulting in catalogues of detected sources’ properties
• The middleware is responsible for marshalling the applications layer over a (large) cluster; it’s responsible for tasks such as disk i/o, load balancing, fault tolerance, and provenance.
• The databases to handle the resulting astronomical catalogues, along with all relevant metadata
• A build system
The Application Layer

The application codes are being implemented in C++, exported to python using swig.
The Application Layer

The application codes are being implemented in C++, exported to python using swig.

There are many ways to link python and C/C++:

- ctypes
- boost::python
- swig
- pyrex
- cython

We chose swig because of its non-invasive nature, its level of support, and its high-level semantics — a C++ std::list <...> becomes a python list.
The Application Layer

The application codes are being implemented in C++, exported to python using swig.

```lisp
(defun tcl-return-variable (var)
  "Set a TCL variable from C"

  (interactive (list
    (read-string "0 variable: ")))

  (let {
    (varStr (read-string "Name of TCL variable: " (concat var "Str")))
    (here (point))
  }
    (insert (format "if(\%s != NULL) {
          char buff[100];
          sprintf(buff, "%\%g", \%s);
          if(Tcl_SetVar(interp, \%s, buff, TCL_LEAVE_ERR_MSG) == NULL) {
            Tcl_SetResult(interp, "Failed to return \%s", TCL_VOLATILE);
            return(TCL_ERROR);
          }
        }\"
        varStr var varStr var))
        (save-excursion
          (goto-char here)
          ;(delete-horizontal-space)
        ))
```
The Application Layer

The application codes are being implemented in C++, exported to python using swig.

The classes implementing images are based on JPL’s VisionWorkbench:
The Application Layer

The application codes are being implemented in C++, exported to python using swig.

The classes implementing Images are based on JPL’s VisionWorkbench:
The Application Layer

The application codes are being implemented in C++, exported to python using swig.

The classes implementing Images are based on JPL’s VisionWorkbench:

```cpp
template <class PixelT>
class ImageView : public ImageViewBase<ImageView<PixelT> >
{
  boost::shared_array<PixelT> m_data;
  unsigned m_cols, m_rows, m_planes;
  PixelT *m_origin;
  ptrdiff_t m_cstride, m_rstride, m_pstride;

public:
  // Constructor an empty image with zero size.
  ImageView()
    : m_cols(0), m_rows(0), m_planes(0), m_origin(0), m_cstride(0),
      m_rstride(0), m_pstride(0) {}
...
};
```
This is a different approach from the PyRAF group’s, which defines all of its classes in python, making extensive use of numpy and scipy.
This is a different approach from the PyRAF group’s, which defines all of its classes in python, making extensive use of numpy and scipy.

This is a good match if you e.g.:

• Want to read images from disk, manipulate them and pass them to an external program
• Wish to read a catalogue of objects into a python array,
This is a different approach from the PyRAF group’s, which defines all of its classes in python, making extensive use of numpy and scipy.

This is a good match if you e.g.:

- Want to read images from disk, manipulate them and pass them to an external program
- Wish to read a catalogue of objects into a python array,

We rejected this solution as we felt that the natural place to create many objects was in C++.
This is a different approach from the PyRAF group’s, which defines all of its classes in python, making extensive use of numpy and scipy.

This is a good match if you e.g.:

- Want to read images from disk, manipulate them and pass them to an external program
- Wish to read a catalogue of objects into a python array,

We rejected this solution as we felt that the natural place to create many objects was in C++.

An example is the task of detecting all of the objects present in an image.
Where we are using Python?

Our fundamental operations are implemented in terms of C++ classes, and in C++. Where does python fit in?
Where we are using Python?

Our fundamental operations are implemented in terms of C++ classes, and in C++. Where does python fit in?

- Writing tests
Where we are using Python?

Our fundamental operations are implemented in terms of C++ classes, and in C++. Where does python fit in?

• Writing tests
  A downside of this is that the developer has to be sure that problems revealed are in the C++ code not the interface layer — but in the long run we need to test both.
Where we are using Python?

Our fundamental operations are implemented in terms of C++ classes, and in C++. Where does python fit in?

- Writing tests
- As a high-level debugger
Where we are using Python?

Our fundamental operations are implemented in terms of C++ classes, and in C++. Where does python fit in?

- Writing tests
- As a high-level debugger
- In quality assurance
Where we are using Python?

Our fundamental operations are implemented in terms of C++ classes, and in C++. Where does python fit in?

- Writing tests
- As a high-level debugger
- In quality assurance
- We've pushed the C++ interfaces down to quite a low level. The modularity desired by the middleware is higher so they are written in python, making calls to a sequence of C++ primitives.
We have the option of writing C++ or python; how should we choose?
We have the option of writing C++ or python; how should we choose?

Answer: it doesn’t really matter unless you write
Integration with numpy

We’d like our array classes to map seamlessly onto numpy’s array classes, but there are at least two issues:
Integration with numpy

We’d like our array classes to map seamlessly onto numpy’s array classes, but there are at least two issues:

- How well our image classes map to (or at least play with) numpy’s.
- The extent to which our C++ function calls and methods return numpy arrays rather than pure python lists.
Let us deal with the former first. LSST’s Image class is similar to numpy’s 2-D array classes, but not identical. In the past (prior to swig 1.3.27) it was possible to create python classes that inherited from both numpy’s ndarray and LSST’s Image but this solution was fragile, and we understood the question of exactly who owned memory and when it could be safely deleted only hazily.
Let us deal with the former first. LSST’s Image class is similar to numpy’s 2-D array classes, but not identical. In the past (prior to swig 1.3.27) it was possible to create python classes that inherited from both numpy’s ndarray and LSST’s Image but this solution was fragile, and we understood the question of exactly who owned memory and when it could be safely deleted only hazily.

Another approach would be to make the LSST’s Image class inherit from ndarray — but there we have problems with the C — C++ barrier.
Let us deal with the former first. LSST’s \texttt{Image} class is similar to numpy’s 2-D array classes, but not identical. In the past (prior to swig 1.3.27) it was possible to create python classes that inherited from both numpy’s \texttt{ndarray} and LSST’s \texttt{Image} but this solution was fragile, and we understood the question of exactly who owned memory and when it could be safely deleted only hazily.

Another approach would be to make the LSST’s \texttt{Image} class inherit from \texttt{ndarray} — but there we have problems with the C — C++ barrier.

It seems likely that a solution can be implemented, but it may not be clean.
The question of numpy record arrays versus python lists, seems to be purely a matter of policy, and writing the proper swig typemaps.
The question of numpy record arrays versus python lists, seems to be purely a matter of policy, and writing the proper swig typemaps.

However, it does raise the question of how much one wants numpy’s arrays to dominate the data structures of what is basically a python programme.
Science with LSST
Science with LSST

- Dark Energy: Weak Lensing, BAO, SNe Ia
Science with LSST

- Dark Energy: Weak Lensing, BAO, SNe Ia
- Evolution of galaxies at high redshift
Science with LSST

- Dark Energy: Weak Lensing, BAO, SNe Ia
- Evolution of galaxies at high redshift
- Milky Way structure out to M31
Science with LSST

- Dark Energy: Weak Lensing, BAO, SNe Ia
- Evolution of galaxies at high redshift
- Milky Way structure out to M31
- New classes of explosive events
Science with LSST

• Dark Energy: Weak Lensing, BAO, SNe Ia
• Evolution of galaxies at high redshift
• Milky Way structure out to M31
• New classes of explosive events
• Populations of small bodies (near and far) in the Solar System.
Science with LSST

- Dark Energy: Weak Lensing, BAO, SNe Ia
- Evolution of galaxies at high redshift
- Milky Way structure out to M31
- New classes of explosive events
- Populations of small bodies (near and far) in the Solar System.
- ...
Variability from SDSS
Variability from SDSS
Variability from SDSS