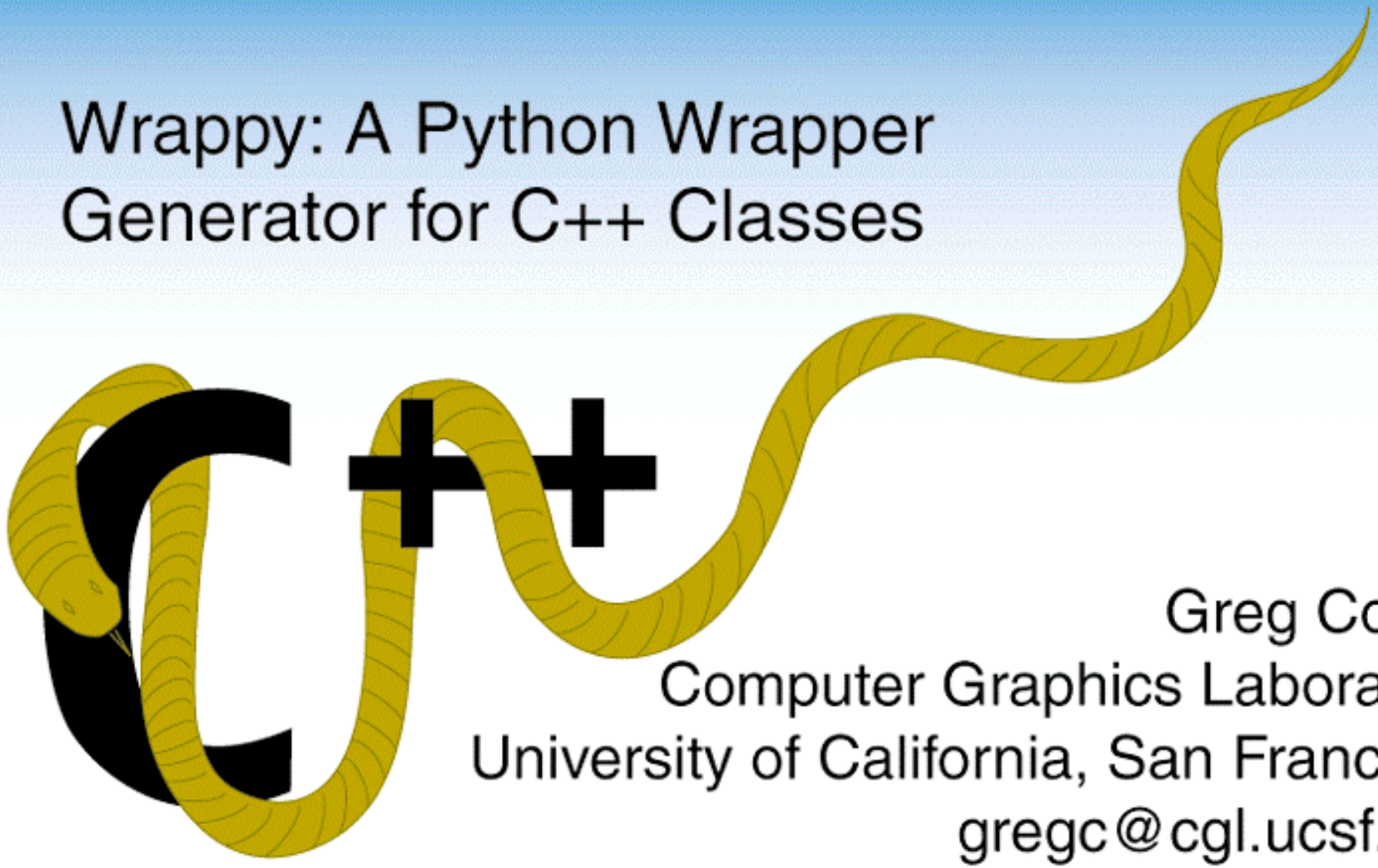
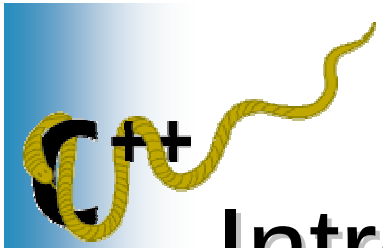
Wrappy: A Python Wrapper Generator for C++ Classes



Greg Couch
Computer Graphics Laboratory
University of California, San Francisco
gregc@cgl.ucsf.edu

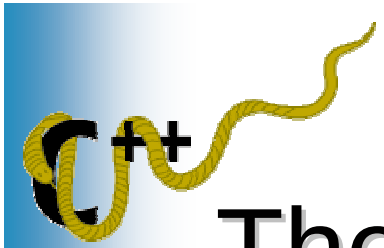
1999 O'Reilly Open Source Software Convention
Python Conference



Introduction

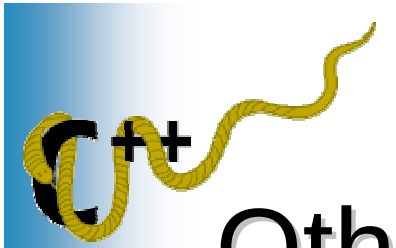
Wrappy is a programming tool for wrapping a C++ library inside a Python extension module. In this presentation we will cover:

- why we developed this tool
- C++ with Python semantics
- examples



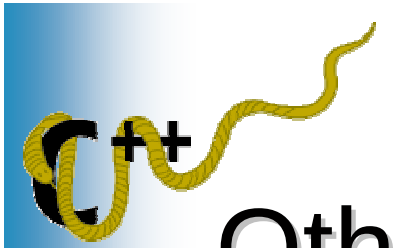
The Need for Speed

- Chimera Molecular Modeling System
  - 3/1997: C++/Motif/OpenInventor/OpenGL
  - 3/1998: Python/Tk/C++/OpenGL
    - 5 C++ classes, 29 member functions
  - 3/1999: Python/Tk/C++/OpenGL
    - 30 C++ classes, 500 member functions



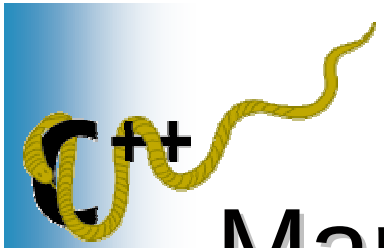
Other Wrapper Generators

- ***SWIG***
  - David Beazley, University of Chicago
  - sets the standard
  - not designed specifically for C++ nor for Python
  - no operator/function overloading
  - no attribute support, no exception support
  - shadow classes



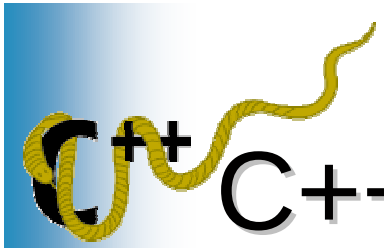
Other Wrapper Generators

- ***PYFFLE***
  - Patrick Miller, LLNL
  - deserves to better known
  - doesn't parse C++ declarations
  - doesn't handle exceptions nor namespaces
  - requires smart pointers for C++ resource management



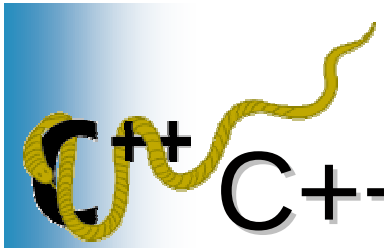
Mapping C++ into Python

- C++ classes become Python types
  - member data/functions are attributes
  - static/enumerated constants map to both module variables and read-only attributes
  - static functions are module functions
  - numeric operators create numeric types
- global functions and constants in module



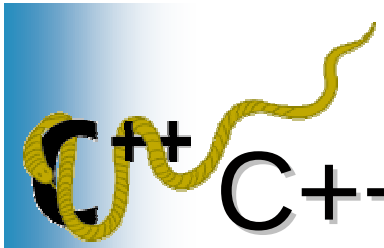
C++ and Python: Issues and Difficulties

- Can C++ classes be subclassed in Python?
- What becomes a Python attribute?
  - member data
  - accessor member functions (get/set)
- How do function parameters return results?



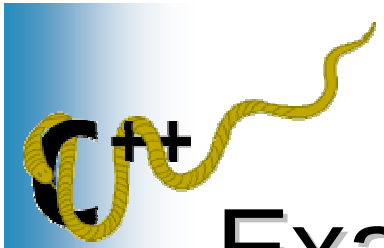
C++ and Python: More Issues and Difficulties

- What about private constructors and/or destructors?
- How are exceptions handled?
- How are containers handled? (vector<>, set<>, map<>, arrays)
- Can callback functions be written Python?



C++ and Python: Yet More Issues and Difficulties

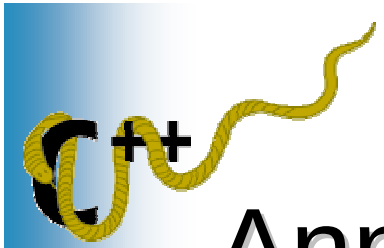
- Do wrapped functions take keyword arguments?
- Are C++ namespaces supported?
- Are documentation strings generated?
- How are object lifetimes controlled?
  - resource (memory) management



Example Code

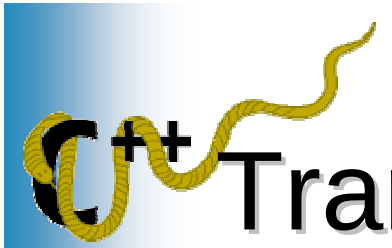
void trackingXY(const char *mode, /*OUT*/ int *x, /*OUT*/ int *y);

```
extern "C"
PyObject *
ToglViewer_trackingXY(PyObject *self, PyObject *args, PyObject *keywds)
{
	ToglViewerObject *wco = static_cast<ToglViewerObject *>(self);
	if (wco->inst == NULL) {
		PyErr_SetString(_chimeraError, "C++ ToglViewer instance gone");
		return NULL;
	}
	try {
		char * ptArg1;
		static char *kwlist[] = { "mode", NULL };
		if (!PyArg_ParseTupleAndKeywords(args, keywds, "z:trackingXY", kwlist, &ptArg1))
			return NULL;
		char* cppArg1 = ptArg1;
		int cppArg2;
		int cppArg3;
		wco->inst->trackingXY(cppArg1, &cppArg2, &cppArg3);
		return Py_BuildValue("ii", cppArg2, cppArg3);
	} catch (std::exception &e) {
		PyErr_SetString(_chimeraError, e.what());
	} catch (...) {
		PyErr_SetString(_chimeraError, "unknown C++ exception");
	}
	return NULL;
}
```



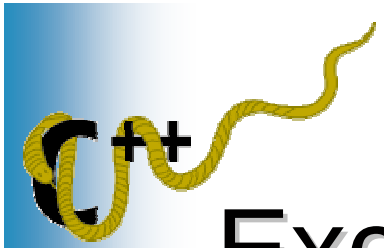
Annotating C++ Header Files

- controlling *wrappy* behavior
  - input/output parameters
  - subclassable
  - abstract base classes
- external methods for controlling *wrappy*
  - use *unifdef* to limit what gets wrapped
  - use subset of header files



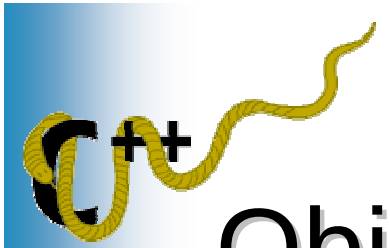
Translating Attribute Names to C++ Primitives

- Python objects use linear list of method names (attribute names) to find appropriate primitive. The list order is based on profiling Python code.
- We have no *a priori* knowledge of which name is more likely to used, so we use a near perfect hash function (courtesy of gperf).



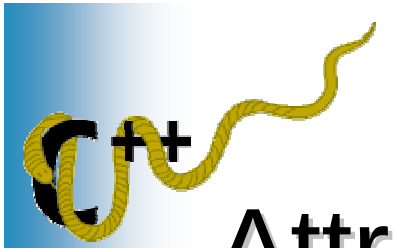
Exception Support

- C++ exceptions are converted into Python exceptions
  - improves error handling, makes Python environment more robust
- use optional function exception specifcations to control exception scaffolding



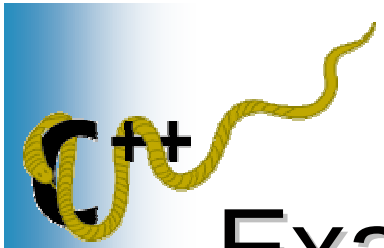
Object Lifetimes

- C++ objects and corresponding Python objects should have the same lifetimes
- need to coordinate Python reference counts with C++ constructors and destructors
- can’t handle every case



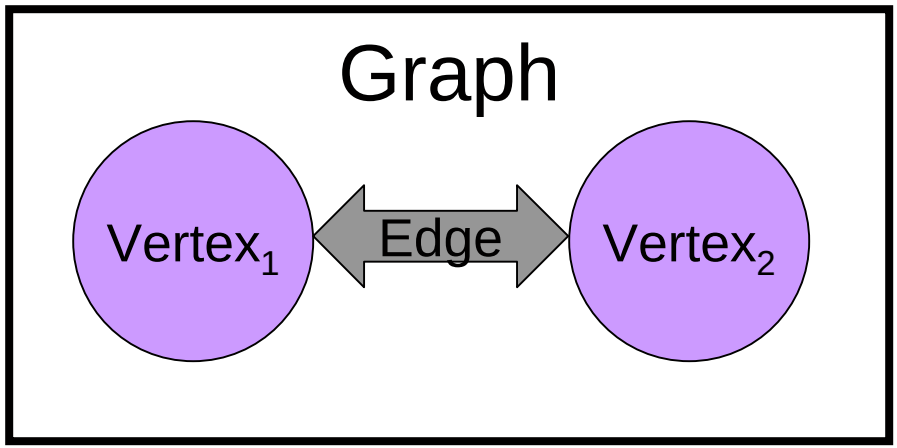
Attribute Caching

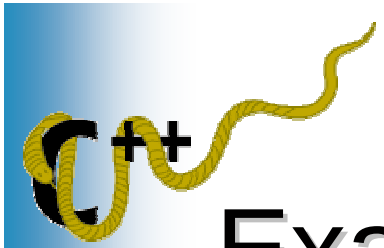
- attributes are non-computed state
- only non-primitive types need to be cached
- Python “sees” C++ object when retrieving attribute
- C++ needs to save Python reference when setting attribute



Example: Graph

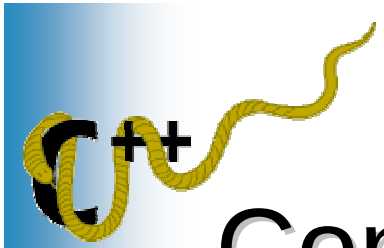
- Vertex and Edge object lifetimes controlled by Graph (*i.e.*, private destructors).





Example: Surfnet

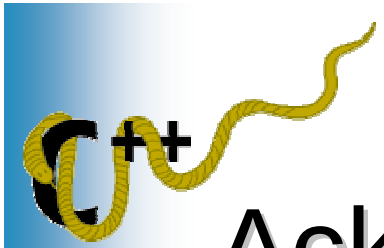
- already did volume decimation in C++
- BSP trees prototyped in Python
- straight translation to C++
  - 109 lines of Python
  - 265 lines of C++ (excluding wrapper)
  - 120 times faster in wrapped C++



Conclusion

- Python is used to write application tools
- C++ can be used for speed
- C++ libraries can be used for additional functionality
- *wrappy* takes care of interfacing C++ code with Python (without extraneous layers)

http://www.cgl.ucsf.edu/Research/otf/wrappy/



Acknowledgements

- the NIH NCRR Resource for Biomolecular Graphics, NIH P41-RR01081
- the UCSF Computer Graphics Laboratory
  - Tom Ferrin, Conrad Huang, Eric Pettersen, Al Conde, Heidi Houtkooper, Tom Goddard