SWIG : An Easy to Use Tool for Integrating Scripting Languages with C and C++

David M. Beazley

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

`beazley@cs.utah.edu`

July 12, 1996

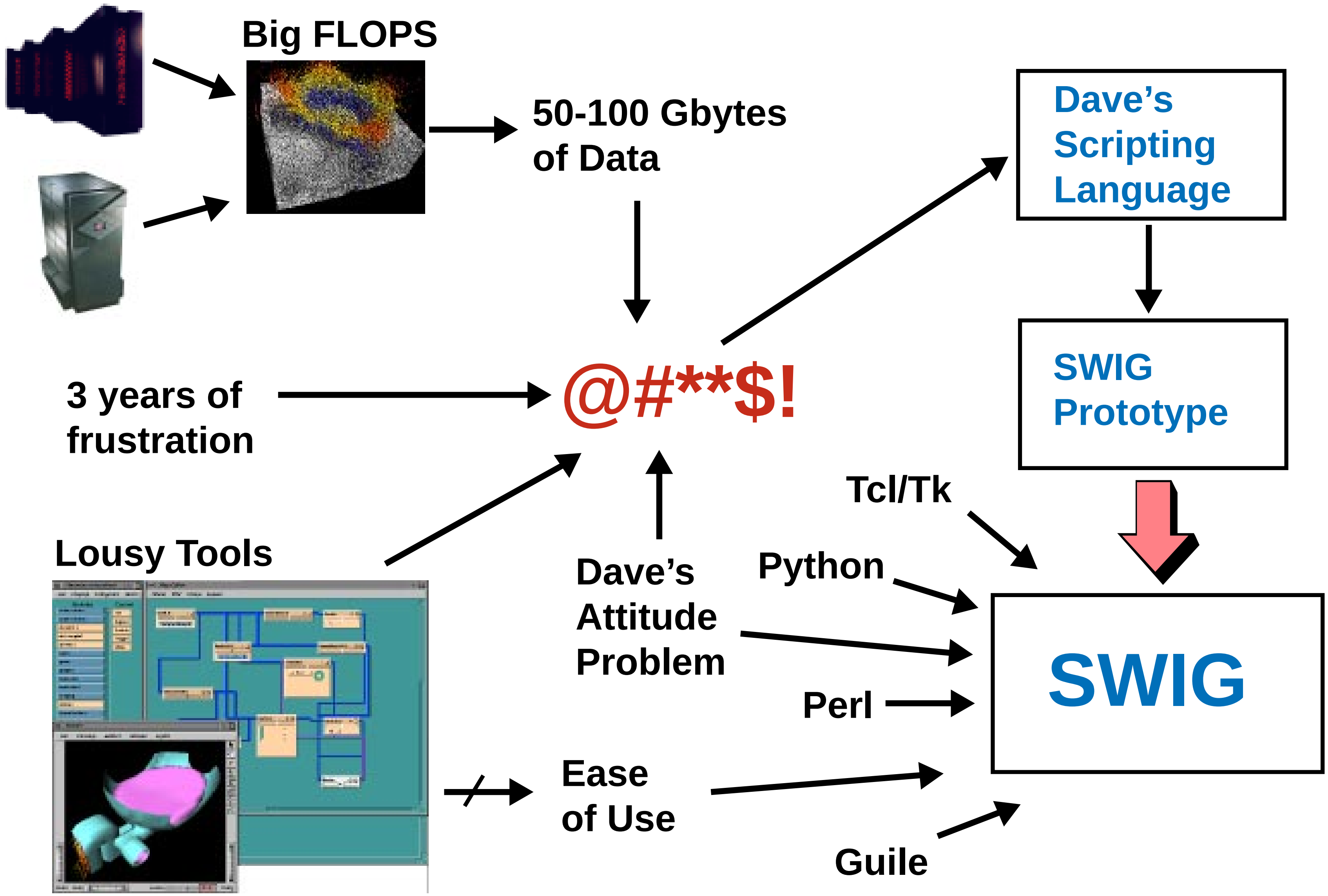
Topics

- **What is SWIG?**
- **Background**
- **A quick tour**
- **Applications**
- **Limitations**
- **Future directions**

SWIG
(Simplified Wrapper and Interface Generator)

- Compiles ANSI C/C++ declarations into bindings to interpreted languages
- Supports most C/C++ datatypes
- Simple C++ classes
- Run-time type checking
- Multiple files and modules
- Automatically generates documentation
- Currently supports Tcl, Python, Perl5, Perl4, Guile3

Where am I coming from?



Big FLOPS

50-100 Gbytes of Data

3 years of frustration

@#**$!

Lousy Tools

Dave's Attitude Problem

Ease of Use

Dave's Scripting Language

SWIG Prototype

Tcl/Tk

Python

Perl

Guile

SWIG

The Two Language Model

- Two languages better than one
- C/C++ (performance, number crunching, etc...)
- Tcl (control, debugging, modules, user interface)
- Unfortunately, need to write “wrapper” functions

C function

```
int fact(int n) {
    if (n <= 1) return 1;
    else return n*fact(n-1);
}
```

Tcl Wrapper Function

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int arg0, result;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
    return TCL_OK;
}
```

Automatic Wrapper Generation

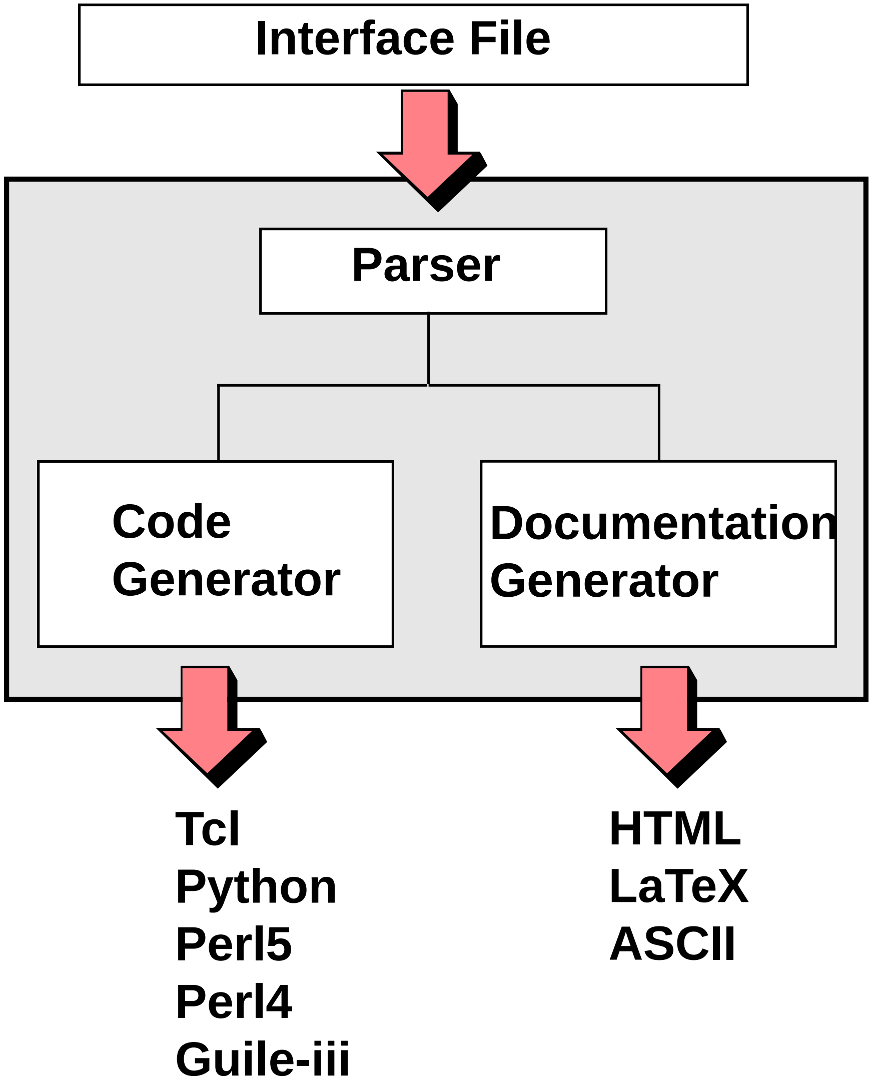
- Most languages have tools to generate wrapper code
- Usually only support a single target language
- Often use non-C syntax
- Special purpose

SWIG Design Goals:

- Use ANSI C/C++ syntax
- Ease of use and flexibility
- Language independence
- Provide a parser and primitives. Allow everything else to be redefined.

SWIG Overview

- Interface file with ANSI C/C++
- Generic parser
- Target languages implemented as C++ classes
- Easy to extend (well mostly)
- Produces C/C++ source file as output.



A Simple Example

fact.c

```c
int fact(int n) {
   if (n <= 1) return 1;
   else return(n*fact(n-1));
}
```

fact.i (SWIG Interface File)

```
%module fact
%{
/* put header files here */
%}
extern int fact(int);
```

```
unix> swig -tcl fact.i
unix> gcc -c fact.c fact_wrap.c -I/usr/local/include
unix> ld -shared fact.o fact_wrap.o -o fact.so
unix> tclsh7.5
% load ./fact.so
% fact 6
720
%
```

Datatypes

- All C/C++ built-in datatypes

```
int, short, long, char, float, double, void
```

- C/C++ pointers (used for everything else)

- Pointers represented as character strings

```
Vector *new_Vector(double x, double y, double z);

% set v [new_Vector 1 2 3]
_1008e248_Vector_p
```

- Pointers are type-checked at run-time.

```
% set v [new_Vector 1 2 3]
% set n [new_Node]
% set d [dot_product $v $n]
Type error in argument 2 of dot_product.
Expected _Vector_p
%
```

Functions, Variables, and Constants

- **Wrapping a C/C++ function**

```
double   foo(double a, int b, void *ptr);
extern   Vector *transform(Matrix *m, Vector *v);
int      MyClass::bar(double);
```

- **Linking with a global variable**

```
extern int Status;
```

- **Creating a constants**

```
#define  MY_CONST     5
enum swig {ALE, LAGER, PORTER};
const double PI = 3.1415926;
```

- **Most “typical” C declarations can be handled.**

  (but not functions taking arrays of pointers to functions, etc...)

SWIG and C++

- Simple C++ classes and structs
- Constructors, destructors, and virtual functions
- Single public inheritance
- C++ politely turned into C and wrapped (for now)

```
%module list
%{
#include “list.h”
%}

class List {
public:
    List();
    ~List();
    void insert(char *item);
    int  length;
    static void print(List *);
    ...
};
```

```
List *new_List(void) {
    return new List;
}
void delete_List(List *l) {
    delete l;
}
void List_insert(List *this, char *i){
    this->insert(i);
}
int List_length_get(List *this) {
    return this->length;
}
int List_length_set(List *this, int v){
    return (this->length = v);
}
void List_print(List *l) {
    List::print(l);
}
```

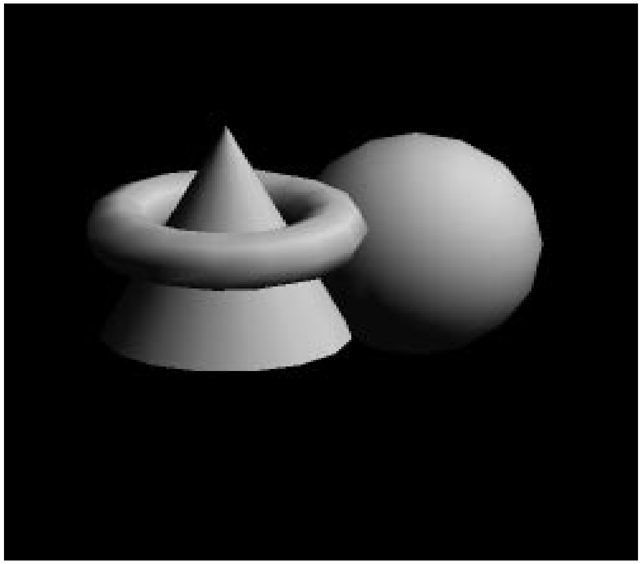
Controlling and Debugging C/C++

- SWIG provides direct access to C/C++ variables, functions, constants, and classes.
- Requires minimal or no code modifications
- Can control existing C/C++ applications at a function level.
- Tcl makes a great debugger.
- Easily add Tcl/Tk to programs without worrying about messy details.
- SWIG works particularly well in research applications

Rapid Prototyping

- **Use SWIG for prototyping and experimentation**
- **Example : OpenGL module**
  - **Developed from OpenGL header files (gl.h, glu.h, aux.h)**
  - **708 constants**
  - **426 functions**
  - **> 8000 lines of wrapper code**
  - **Total development time : < 20 minutes**
- **Sample code :**

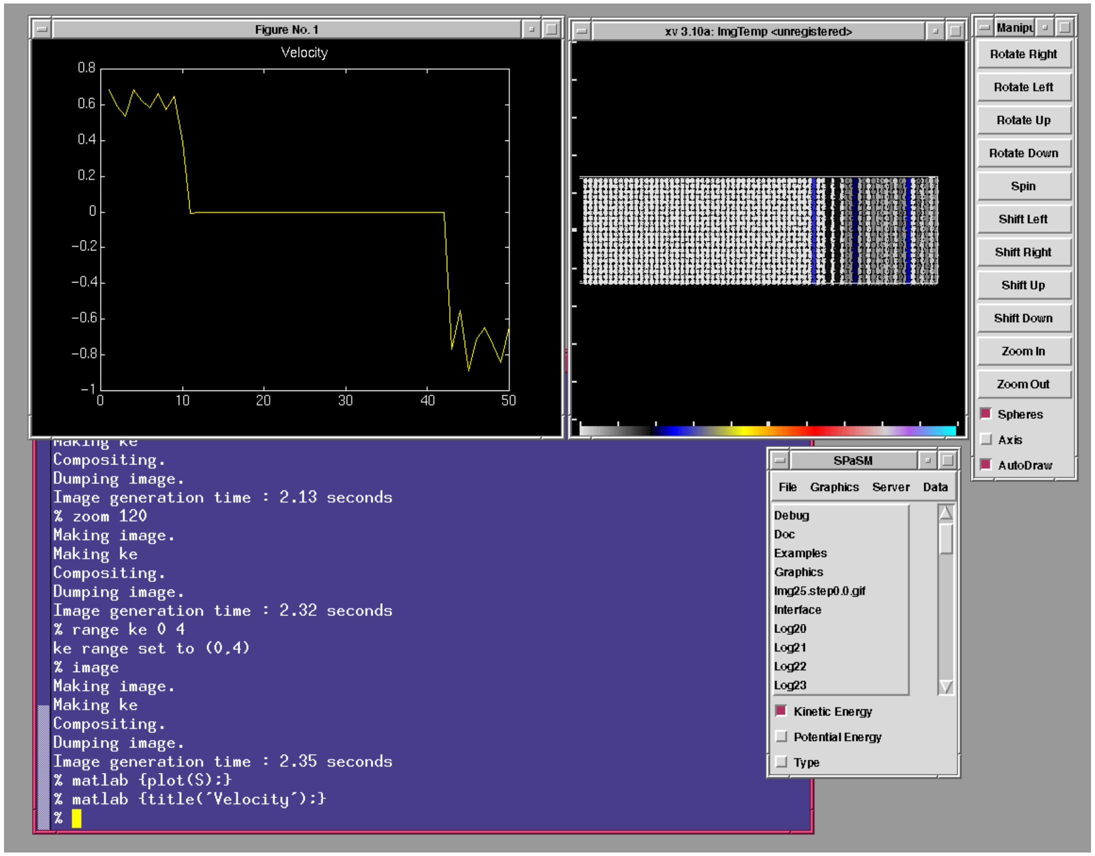
```
set light_ambient [newfv4 0.0 0.0 0.0 1.0]
glLightfv $GL_LIGHT0 $GL_AMBIENT $light_ambient
...
glClear $GL_COLOR_BUFFER_BIT
glPushMatrix
glRotatef 20.0 1.0 0.0 0.0
glPushMatrix
glTranslatef -0.75 0.5 0.0
glRotatef 90.0 1.0 0.0 0.0
auxSolidTorus 0.275 0.85
glPopMatrix
...
```



Building Modular Applications

- SWIG can be used to build highly modular and programmable applications

- SPaSM Molecular Dynamics Code
- MATLAB
- Data Analysis
- Tcl/Tk

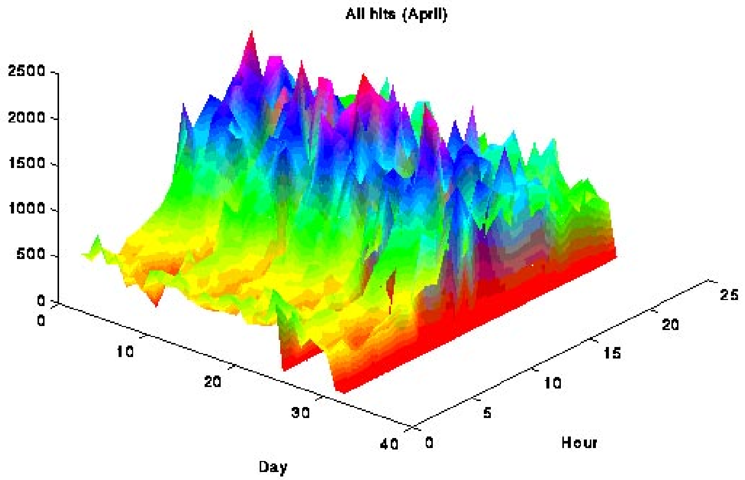


```
Making ke
Compositing.
Dumping image.
Image generation time : 2.13 seconds
% zoom 120
Making image.
Making ke
Compositing.
Dumping image.
Image generation time : 2.32 seconds
% range ke 0 4
ke range set to (0,4)
% image
Making image.
Making ke
Compositing.
Dumping image.
Image generation time : 2.35 seconds
% matlab {plot(S);}
% matlab {title('Velocity');}
%
```

- Don’t build a huge monolithic system---build components and reuse them.

Language Independence

- All languages have strengths and weaknesses
- SWIG interface files are language independent
- Language independent code re-use!
- Example :
  - Perl5 script to generate web statistics
  - Uses MATLAB module developed for Tcl/Tk



Current Limitations

- **C++ support is incomplete (and will probably always be so)**
- **No exception model**
- **Numerical representation problems (particularly unsigned integers and longs)**
- **No variable or optional arguments**
- **Pointer model is extremely flexible, yet incompatible with most other Tcl extensions. (ie. File handles).**
- **SWIG is still too difficult to extend (well, I think so).**

Conclusions

- SWIG works with a wide variety of C code (and a reasonable subset of C++)
- Particularly well-suited for research applications
  - Scientists like the ease of use
  - Works well with existing code
  - Provides a direct mapping onto C/C++
- Language independence is essential!
  - There is no “best” scripting language
  - Different applications have different needs
- SWIG has proven to be remarkably reliable and powerful in a variety of applications
- Much work remains!

Future Work

- **SWIG 2.0**
- **Support for more target languages**
  - **Itcl**
  - **Object Tcl**
  - **ILU**
  - **Java?**
- **An exception model**
- **More complete C/C++ parsing**
- **Simplified extension mechanism**
- **Using SWIG to do cool stuff**

Acknowledgments

- Users who have braved the first few releases and provided feedback
- John Buckman (non-unix platforms)
- Peter Lomdahl, Shujia Zhou, Brad Holian, Tim Germann, Niels Jensen (LANL).
- John Schmidt, Kurtis Bleeker, Patrick Tullmann
- The Scientific Computing and Imaging group
- The Advanced Computing Laboratory (LANL)
- DOE, NSF, NIH

Blatant Advertisement

SWIG is free and fully documented

Source code and user manual are available via anonymous FTP:

`ftp://ftp.cs.utah.edu/pub/beazley/SWIG`

The SWIG homepage:

`http://www.cs.utah.edu/~beazley/SWIG`

The SWIG mailing list:

`swig@cs.utah.edu`