RiverTools: From Concept to "Commercial Success"

Scott D. Peckham

University of Colorado at Boulder
Scott.Peckham@colorado.edu

Community Sediment Model Workshop
Boulder, February 20, 2002

Maintaining “Big Code”

Project Planning (flowcharts, future wish lists)

Problem ownership

Internal consistency

Debugging: Test cases that test all “extreme cases”

Documentation

Backward compatibility issues

Code Readability

There are a number of simple “tricks” or programming habits that make code much more readable. This makes it MUCH easier to understand, modify, maintain and debug. (Both for others and for yourself in 6 months.)

(1) Blow-by-blow in-code documentation
(2) Logical clustering of lines separated by white space
(3) Consistent indentation
(4) Use of Boolean variables for IF & CASE statements
(5) Logical variable names; underscores between words
(6) Small (sometimes tiny) modules that do one thing well
(7) Complete avoidance of GOTOs
(8) Naming conventions for functions and procedures
(e.g. list = Ellipsoid_List(), d = Distance(x,y),
The “Verb_Adverb_Noun” convention: Close_All_Files

Boolean Variables & Readability

A Boolean variable is one that takes on one of two possible values or states, such as 0 or 1, True or False, Yes or No, On or Off.

Examples:

(1) ERROR = (nlayers lt 0)
(2) DONE = (n eq 1000) OR ERROR
(3) ONSCREEN = (x gt xmin) AND (x lt xmax) AND
(y gt ymin) AND (y lt ymax)
(4) NEGATIVE = (value lt 0)
(5) VALID = NOT(ERROR) AND (value lt 100000)
(6) ALIVE = (base_ID gt 0)
(7) IN_RANGE = (value gt 0) AND (value lt 10)
(8) READABLE = (nlines lt 10) AND (GOOD_NAMES)

A Snippet of IDL Code

```
function Lines_In_File, filename

;---------------------------------
;Open the input text file
;---------------------------------
openr, unit, filename, /get_lun

;-----------------------------------------
;Count number of lines in file
;-----------------------------------------
line = ‘’
n_lines = 0L
while NOT(EOF(unit)) do begin
     readf, unit, line
     n_lines = (n_lines + 1L)
endwhile

;----------------------------
; Close the input file
;----------------------------
free_lun, unit

RETURN, n_lines
END;  Lines_In_File
```

Things to notice in this simple example:

(1) EOF function appears as a Boolean variable in the while loop.
(2) Logical variable names
(3) Logical function name; underscores
(4) Blow-by-blow documentation
(5) Indented while loop

The “Lines_In_File” function could be called on the IDL command line (or from another IDL procedure) as follows:

```
IDL> my_file = ‘Thesis.txt’
IDL> n_lines = Lines_In_File(my_file)
IDL> print, n_lines
```

Data Types

Proper choice of data type is a key consideration in terms of both efficiency and avoidance of “strange bugs”.

Large floating point numbers are “further apart” than small ones.

E.g. Since 32-bit filesystems use long integers to track file pointer position in files, they have a filesize limit of 2.1 GB.

Name	RAM Used	Minimum	Maximum	Notes
byte	1 byte	0	2^8 - 1	2^8 = 256
integer	2 bytes	-(2^15)	2^15 - 1	2^15 = 32,768
long	4 bytes	-(2^31)	2^31 - 1	2^31 = 2,147,483,648
float	4 bytes	a^pmin	c * a^pmax	Machine-dep.
double	8 bytes	b^qmin	d * b^qmax	Machine-dep.

Structures vs. Arrays

Structures are like flexible data types for storing the various"attributes" of some "entity". An array of structures would be a good way to store the "attributes" of all employees in a company or all books in a library.

Structures are available in almost every modern programming language and are the concept upon which object oriented programming is based.

Using arrays of structures when appropriate makes code MUCH easier to maintain and to extend when you think of something new to add.

Examples:

```
employee = {name:'John',  age:28,  height:5.9,  salary:40000.0,
                start_date:'May 5, 1995',  SSN:'561-42-6051'}
print, employee.age

circle = {x0:1.5, y0:2.0, radius:5.0, color:'blue'}

book = {title:'Tale of Two Cities', author:'Charles Dickens',
        n_pages:400, publisher:'John Wiley'}
```

RAM vs. File I/O

Accessing information stored in RAM is typically about 100 times faster than accessing information stored in a file.

However, the amount of hard disk space available for storing files is typically 100 to 200 times more than the amount of RAM.

“Virtual memory” or “paging” is an OS trick for handling more data than will fit into RAM. Not always best to leave this to OS.

Year	RAM (MB)	Hard Drive (MB)	HD / RAM Ratio
1985	0.1	10	100
1990	1	100	100
1994	10	1,000	100
1998	100	10,000	100
2003	1,000	100,000	100

File Format Issues

Spatial data is often stored in large 2D arrays or “raster grids.” This data may be saved in a file for archiving or for transfer between machines.

People are often inclined to save 2D arrays as text in ASCII files. The main reason is they want to have the option of visually inspecting the data values. This is fine for small grids (say, less than 100 rows and columns) but is not a good idea for large grids.

When raster data is stored in binary files, it takes up much less space on your hard drive and can be read into RAM much more quickly. (ASCII to binary conversion is a very slow process.)

You can still visually inspect data values using “zoom tools” in raster GIS programs such as RiverTools.

Portability of ASCII and Binary files; Byte order and EOL issues.

ASCII header files & binary data files vs. all-in-one files

Efficient Algorithms

The algorithm that is used to solve a particular problem can make an enormous difference in program speed.

The “computational cost” of an algorithm is measured in terms of the number of “units,” **n**, that must be “examined” or handled.

Examples:

FFT algorithm: O(n*log(n)) vs. O(n^2)
If n = 10^6, 30 secs vs. 2 weeks CPU time

Sorting algorithms: Brute force method: O(n^2)
Quicksort usually O(n*log(n)), O(n^2) worst.
Heap sort has worst case of O(n*log(n))

Finding entry in a sorted list: O(log(N)) (e.g. phone book)

Convex hull of n points: O(n^2) vs. O(n*log(n))

Graphical Interface Design

Extendible designs

Droplists

Options menus (e.g. Right Click concept)

Context-specific help

Constrain input to valid values

User friendliness

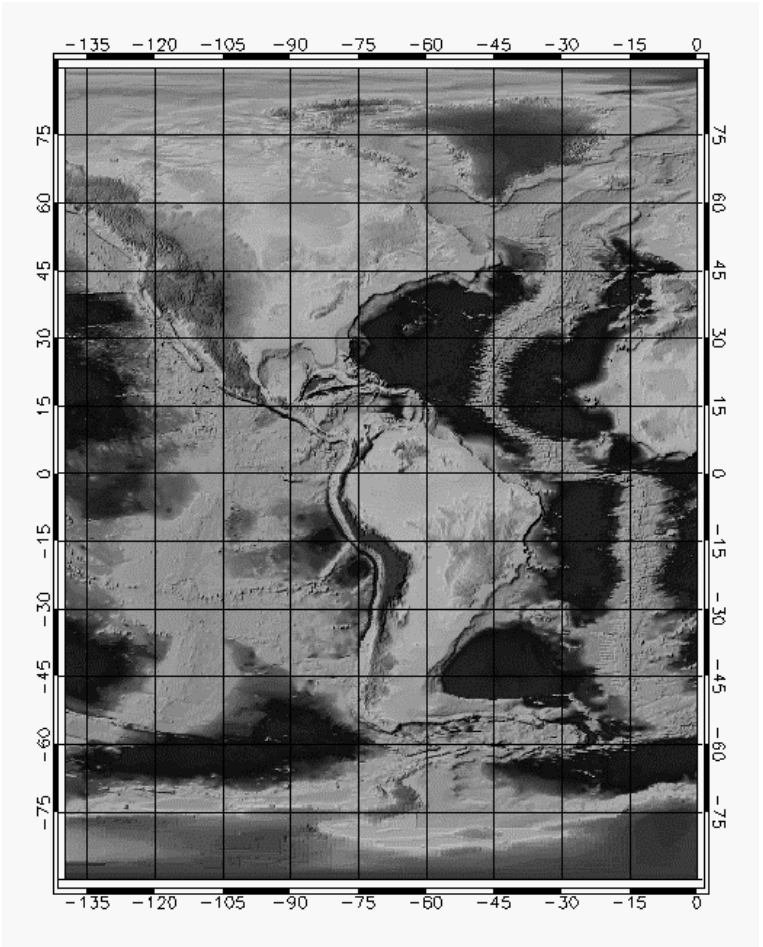
A Few Other Issues

(1) Testing and debugging (it never ends)
(2) Pointers to avoid passing/copying large arrays
(3) Small, reusable modules that aren't hardwired to
a particular application or situation.
(4) Vector vs. Raster issues
(5) Pre- and post-processing utilities
(6) Graphical output utilities (e.g. JPG, BMP, PS)
(7) Inclusion of graphical options in code
(8) High-level languages like IDL, Matlab, Mathematica
(9) Supporting various data formats

Summary

Some important things to think about when writing a big application or “toolkit” are the following:

Project Planning
Modular Design
Extendability
Code Readability
Boolean Variables
Data Types
Structures vs. Arrays
RAM vs. File I/O
File Format Issues
Efficient Algorithms
Graphical Interface Design
Documentation
Testing and Debugging

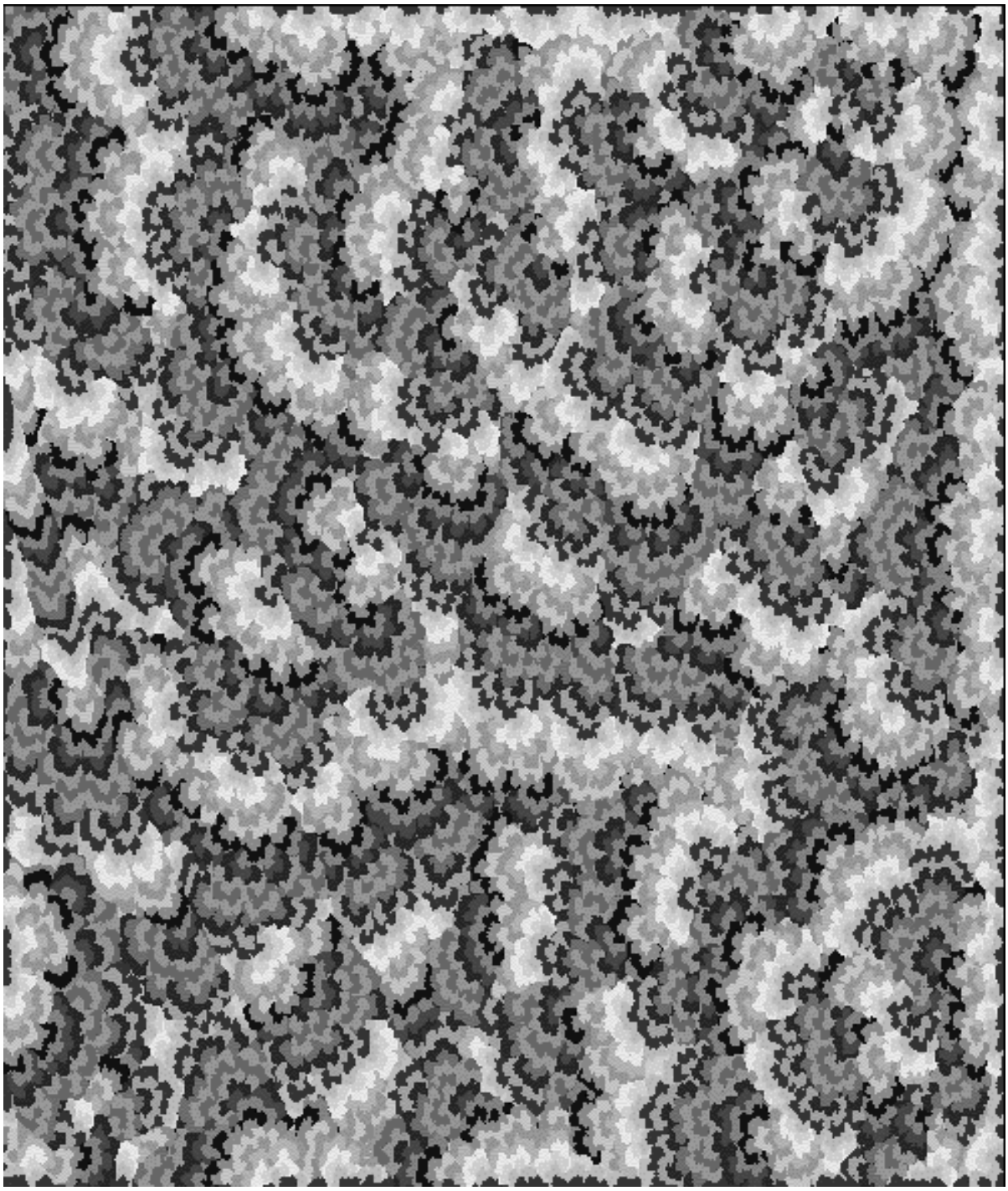


RiverTools: From Concept to "Commercial Success"

Scott D. Peckham

University of Colorado at Boulder
Scott.Peckham@colorado.edu

Community Sediment Model Workshop
Boulder, February 20, 2002



RiverTools: From Concept to "Commercial Success"

Scott D. Peckham

University of Colorado at Boulder
Scott.Peckham@colorado.edu

Community Sediment Model Workshop
Boulder, February 20, 2002

−75
−60
−45
−30
−15
0
15
30
45
60
75
−135
−120
−105
−90
−75
−60
−45
−30
−15
0