Documenting Code

G.A.

December 10, 2014

Changes: doxygen is no longer used. We use the doc.pl script under PsimagLite/scripts

1 The Problem

I like documentation that reads like a story. Like, you know,

> Once upon a time there was a nice smart brave C++ class called Minimizer who set out on a journey to find....

Although, Minimizer will always be nice, smart and brave, other parts of its documentation might change with time. So the problem becomes that such documentation would be written separately from the code, and (i) will become out-of-date pretty soon, and (ii) will be tedious to maintain in sync with what the code actually does.

So, Knuth (of TEX fame) suggested Literate Programming, that is, writing a `.w` file in human format and then getting from it two files: one for the code and one for the documentation. Apart from having the documentation tied to the code, the advantage here is that one can develop a story for the documentation. The disadvantage with this, though, is that tools (IDE, debugger, valgrind) don't understand `.w` files.

On the other hand, we have things like `doxygen` where one is supposed to write documentation in the code as comments and then one is supposed to extract said documentation from the code. The advantage is that now files like `.h` and `.cpp` are the usual ones and all tools understand them. The problem, though, is that doygen will produce an API-like or library-like documentation, which doesn't fit a storyline documentation.

2 The Solution

One writes one's story in file, say, `manual.ptex`, like in Listing 1.

Listing 1: Story without embedded documentation.

```
Once upon a time there was a nice smart brave C++ class
called Minimizer who set out on a journey to find
the minimum of a function for a given range and
maximum number of iterations. This class takes two
template arguments RealType and FunctionType.

Its member function simplex is the one that actually
```

```
performs the minimization using the simplex method,
by setting its first argument with the vector that
minimizes the function. The other arguments are the
delta and the tolerance.
```

Now to simplify keeping documentation and code in sync, what needs to be said about class `Minimizer` better be said in class `Minimizer`, and, likewise, what needs to be said about function `simplex` better be said near that function too. See Listing 2.

Listing 2: Embedded documentation in code.

```
/* PSIDOC Minimizer
find the minimum of a function for a given range and
maximum number of iterations. This class takes two
template arguments RealType and FunctionType.
*/
template<typename RealType,typename FunctionType>
class Minimizer {

...
        /* PSIDOC MinimizerSimplex
            performs the minimization using the simplex method,
            by setting its first argument with the vector that
            minimizes the function. The other arguments are the
            delta and the tolerance. */
        int simplex(VectorType& minVector,
                    RealType delta=1e-3,
                    RealType tolerance=1e-3)
        {
                ...
        }
};
```

So, now the storyline can use hooks to the embedded code, like in Listing 3.

Listing 3: Story with embedded documentation.

```
Once upon a time there was a nice smart brave C++ class
called Minimizer who set out on a journey to
\ptexPaste{Minimizer}

Its member function simplex actually
\ptexPaste{MinimizerSimplex}
```

Now, putting it all together the recipe for writing and extracting documentation is

1. Write a story line using standard LaTeX markdown into one `manual.ptex` file. Use `\ptexPaste` to bring documentation from the code into the story.
2. Write the embedded documentation as C++ comments with LaTeX markdown in the code near the classes and functions, etc.
3. Run the `doc.pl` script to create a `.tex` file from a `.ptex` one.

```
cd dmrgpp/doc
find ../src -iname "*.h" -or -iname "*.cpp" |
perl ../../PsimagLite/scripts/doc.pl manual.ptex
```

that will produce `manual.tex`.

4. Run `pdflatex manual.tex` to obtain `manual.pdf`.

3 Notes

Here are further things I wanted to say but would have distracted from the main points.

1. Knuth's weaving and tangling (or tangling and weaving) would have worked without modification had the tools implemented the understanding of, say, a `.w` file. So, in no way is this writing a criticism of Knuth's idea—and I am using TEX.
2. Compare library-like documentation like this with story-like documentation like this.
3. An excellent example of story-like documentation produced with `doxygen` is the whole book Physically Based Rendering, Second Edition: From Theory To Implementation