Excerpt from the book *Real Programming* (2021) by Corneliusen & Julin.
More information: www.ignorantus.com/real/

Nils Liaaen Corneliusen

Sjur Julin

Real Programming

A book about programming, programmers, programs, and pop culture

Based on the Norwegian book *Ekte Programmering* (ISBN: 978-1-716-41204-2).

Some of the text is based on articles written by Corneliusen in the period 2010-2020.
These were published on the website www.ignorantus.com, owned by Corneliusen.

Please refer to the chapter *Licensing Information* for details about source code and images.

All source code etc. used in the book can be found here:

www.ignorantus.com/real

All enquiries about the book should be made to

real@in-a-vision.com

Publisher:

Ignorantus AS
Munkedamsveien 57
0270 OSLO
NORWAY

ISBN:

978-1-6780-5044-3

Many thanks to Lars Petter and Nils (not that Nils) for useful comments.

"So, I'm taking my work underground. I can't let it fall into the wrong hands again."
- Dr. Lawrence Angelo in the movie *The Lawnmower Man* (1992)

Contents

Introduction 7

A Rant About CPUs, Languages, and Development Methods 9

A Look Back on the Age of Amiga 19

Let's Encrypt! 25

A Quake! Doom is Coming! 31

Let's Encrypt Again! 37

Research is Hard 41

Spaces and Colours and Layout 47

Separate Ways and Separable Filters 57

Parallel Lines and Libraries 75

Snappy Names are Important 83

Let's Encrypt Again! The Last Round 91

A GPU and Some Fractals 95

Julia Quaternions. Err, What? 105

The GPU Raytracer 115

The Binary Fixed-Point Raytracer 125

- The Control Code 135
- A Detour: What About AArch64 Neon? 139

The Rip-Off Artist 143

- The Control Code and a Trip to Denver 159

Bayer, Bayer & Bayer 165

Back for the Quaternion Attack 171

Restoration 177

A Jilted Generation 183

The Big Picture 189

Appendix 191

Licensing Information 193

- Code 193
- Images 196

Introduction

I once read an article called *The Importance of Revisiting Your Old Code*[1] by Kieran Jones that stated

> *As a developer you are constantly learning and improving, or at least you should be (more on this in a future post). One of the hardest things to do is to look back at old projects with your now more experienced eyes and wonder what the heck you were thinking back then.*

Look at some three-year-old code you have written at work. Is the above statement true? Then please quit at once, go home and practice programming before trying again. You are not writing code: You are still learning basic programming on your employer's expense.

Good programmers do not start programming at the university. Programming is a long and laborious process. A good programmer has written code for as long as he can remember. He eschews programming trends and eliminate them ruthlessly whenever they get in the way. A good programmer has a powerful computer at home dedicated to fun code projects, and a collection of smaller ones, like a *Raspberry Pi* or two. Or some microcontrollers with custom hardware, like a laser or a camera. Here is a painful truth: Not everyone will be a great programmer after practicing for years, just like not everyone will be a great guitarist after playing for years. Programming is all about understanding patterns, and few do it well.

A bad programmer, often known as a career programmer, has the following characteristics: They start using computers at the university and master, at best, a subset of the field. They like meetings and plans, but rarely, if ever, write code that is different from the code you find in textbooks. They think the work is done at five o'clock, and do not keep themselves updated in their spare time. They do not own personal computers, so they never write programs for fun, just for money, during daylight hours. And to be politically incorrect: If such a programmer were to be absent from their workplace for a period of months or years, they usually think the world of computing has stood completely still in the meantime. Unfortunately, the world never stands still. At this stage they are usually promoted to a low-end management position or to a support-related department. I have been in this business for decades and seen it happen repeatedly. And it will happen again.

"Best practices" and "programming patterns" (not to be confused with actual patterns) are concepts that were made up to make such bad programmers write average code. These fields present many methods that should be followed to write "good code". I will demonstrate clearly that this is nonsense - formalizing how to write code is as productive as herding cats. Unless, of course, everyone is a sheep. Eventually, the office will be populated by enforcers of the *One True Way* to approach programming, and the number of engineers doing real programming converge towards zero.

Such methods tend to be introduced at some stage, even though it is not required. When this happens the good developers just carry on, writing rock solid code that makes the world a better place. Then the managers step in and enforce these methods, since they are true believers. That is when the brightest abandon ship. All such methods attract bad managers, and bad managers think they are smart and know a lot, since they are, well, managers. It is a downward spiral, and usually connected with the company becoming too large. It always ends up like that: Totally average in every way, using average methods for bad - sorry - average programmers. They go well with average products. And average pay.

I thought, when smartphones started appearing, that it would lead to a new age of computer hobbyism, like in the seventies and eighties, with people hacking away, sharing code and ideas, and having fun. With endless information available, surely people would try to invent their own stuff? Not so fast! *Apple* locked down their phones. If you are going to write code for your new *iPhone*, there is a complicated registering process and there is more money involved. And everything is not fun and games over at Google's web shop either: The *Android* tools and projects are complex enough to keep young and promising programmers far away. Long story short, nobody cared, so now smartphones are mostly used for cheap entertainment and seeing what their friends are doing.

[1] https://medium.com/@Kieran11/the-importance-of-revisiting-your-old-code-58eeb93a0dd7

Personally, I do not own a smartphone, and I do not have any plans to get one, either. I like mechanical watches. Not because they are accurate, which they are not, but because they remind me of the art of programming. Just like in high volume data processing, where every little piece must do its job perfectly and on time, there is no room for a single tooth on a gear to be slightly off. It is the kind of programming that will be covered in this book, and it is the kind of programming eschewed by the sheep: Real programming.