Scyllarus™: From Research to Commercial Software

Nariman Habili
NICTA
Canberra Research Laboratory
Tower A, 7 London Circuit
Canberra ACT 2600, Australia
nariman.habili@nicta.com.au

Jeremy Oorloff
NICTA
Canberra Research Laboratory
Tower A, 7 London Circuit
Canberra ACT 2600, Australia
jeremy.oorloff@nicta.com.au

ABSTRACT
In this paper, we describe the development of Scyllarus, a set of computer vision tools used to process, analyze and visualize hyperspectral images. Scyllarus is comprised of a MATLAB® Toolbox, a C++ API and Scyven, an application with a graphical user interface. The merits of the various tools that were used to develop the software are discussed as well as the challenges that were experienced in developing commercial-grade software from research-grade code. A list of our current and future development commitments is also provided.

General Terms
Algorithms, Documentation, Performance, Design.

Keywords
Hyperspectral imaging, research, iterative development, MATLAB®, C++.

1. INTRODUCTION
Scyllarus is a set of computer vision software tools used to process, analyze and visualize hyperspectral images [1]. Unlike traditional color images that consist of three bands (red, green and blue), hyperspectral images consist of 10s or 100s of bands, covering the visible spectrum, as well as infra-red and beyond. Therefore, hyperspectral images provide much more information about a scene than a color image. This leads to an improved ability to classify materials and objects in a scene based on their spectral properties.

Scyllarus is comprised of the following tools:

- A MATLAB® Toolbox;
- A C++ API; and
- Scyven (Scyllarus Visualization Environment).

ASWEC ' 15 Vol. II, September 28-October 01, 2015, Adelaide, SA, Australia Copyright 2015 ACM. ISBN 978-1-4503-3796-0/15/09...$15.00
DOI: http://dx.doi.org/10.1145/2811681.2817752

All three tools are available for download from [2]. Scyllarus uses state-of-the-art image processing algorithms developed by researchers at NICTA.

The Scyllarus software is developed by members of the Scyllarus engineering team. This was achieved by translating, porting, or implementing research-grade code and algorithms, developed by researchers in computer vision over several years, to commercial grade software. The development process involves working collaboratively with researchers in order to understand their existing algorithms, develop new algorithms and then to implement them in Scyllarus. The engineers also work closely with a business team to assist NICTA in finding commercial opportunities for Scyllarus.

As with any software engineering project, there are a set of challenges. For Scyllarus, most of the challenges involve understanding and porting research grade code into commercial software. This is generally a non-trivial task and involves studying the original code or algorithm and also reading research papers to get a good understanding of the algorithm. A challenge during the initial phase of the development was to select which methods and algorithms would be required to build the core functionality of Scyllarus, that is, which components would be required for the initial commercial product.

The paper is organized as follows. In section 2, we provide a brief overview of Scyllarus and Scyven. In section 3, we describe the software development process, the different tools used to develop the software, and our collaboration with the research and business teams. Discussion and future direction is provided in section 4 and the paper is concluded in section 5.

2. SCYLLARUS
The Scyllarus software is based on several years of research at NICTA. The research mainly dealt with hyperspectral imaging for scene analysis and looked at applications in photography, food security, defense, earth sciences and health [3].

As mentioned previously, the Scyllarus software is comprised of a MATLAB® Toolbox, a C++ API and Scyven. The MATLAB® Toolbox is aimed at academic researchers and is available on a trial basis for non-commercial use. The C++ API is a commercial product and is aimed at developers who wish to integrate hyperspectral imaging capability into their existing or new application. Scyven is a Graphical User Interface (GUI) application and is used for processing, analyzing and visualizing hyperspectral images. It uses the C++ API as its image processing engine. Scyven is also a commercial product, however it is currently available free of charge from [2].

Scyven's GUI was designed with the assistance of a User eXperience (UX) designer. To design the GUI, we described a

typical use-case to our UX designer. We assumed a user who is a domain expert in their field (for example in agriculture or forensics), that has some understanding of hyperspectral imaging. This was based on our experience with users at the time. As our user-base grows, we will facilitate direct interaction between the UX designer and users so that the current GUI can be fine-tuned or redesigned for a specific group of users. Some users may prefer a fully automatic system, while others may require direct interaction with the data generated and the ability to set specific settings.

The Scyven GUI was designed to be user-friendly, consistent and intuitive. A screenshot of Scyven in action is shown in Figure 1. The screenshot depicts an image with 'similar' pixels highlighted in the middle pane, analysis plots on the right pane and current images on the bottom pane. More information on Scyven and its user interface is provided in the Scyven User Guide [4].

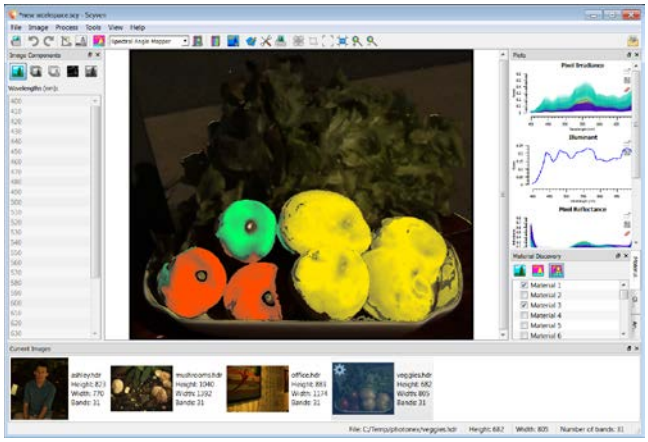

Figure 1: Scyven screenshot

3. SOFTWARE DEVELOPMENT

The software was developed using an iterative development process. Each iteration included requirements gathering, design, coding and testing. We found that an iterative development process allowed users to make suggestions, evaluate performance, and clarify requirements, and thereby enhance the quality and usefulness of the software.

We initially considered adopting a formal software development methodology, such as SCRUM, but found that a prescriptive development process was not suitable for us. This is because, all team collaborators, including researchers, would need to understand and somewhat adhere to such a methodology. However, our sprint or software delivery outputs could not be effectively synchronized to those of researchers. This is because the length of time that a research project takes is usually less predictable than that of an engineering project as it involves algorithm development and experimentation. Also, from time-to-time, researchers have other priorities and are unable to contribute to the Scyllarus project.

3.1 Development Pipeline

The Scyllarus engineering team consists of three research engineers. As mentioned previously, Scyllarus is composed of a MATLAB® toolbox, a C++ API and Scyven. Each engineer is responsible for developing and maintaining a tool, however no engineer has 'ownership' over a tool. Any tool can be modified by any engineer and in practice this happens on a regular basis. As well as making the team more efficient and agile, it also has the added benefit of familiarizing all engineers with the code base of all tools.

The development pipeline of Scyllarus is shown in Figure 2. Firstly, research grade code, developed by PhD students and researchers, is ported to Matlab (if applicable), refactored, optimized and added to the MATLAB® Toolbox. The research code is mostly written in MATLAB®, but sometimes also in C++. The MATLAB® code is then ported to C++ and added to the C++ API. The C++ API is used to develop Scyven. All Scyllarus tools are routinely tested for bug and flaws, which are logged in an issue tracking tool for assessment.

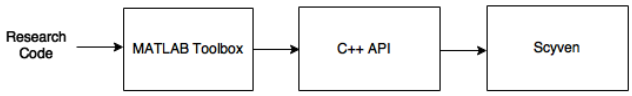

Figure 2: Scyllarus development pipeline

3.2 Tools

The C++ API and Scyven are developed for both Windows and Linux. Therefore, most libraries and tools used are cross-platform. The libraries and tools employed to develop Scyllarus are listed below:

- Git (via GitHub) [5] is used for source control and issue tracking. GitHub Enterprise provides a private GitHub instance that features a fully integrated issue tracker and other tools such as progress reporting and notification systems. The team follows a standard 'Git Flow' methodology, where the master branch of the Git repository for each tool is always considered 'deployable' and features, enhancements and fixes are all worked on and tested in branches before being merged into the master. Team members and internal users are all encouraged to report issues and suggest enhancements using the issue tracker, and assign them to the relevant team member for consideration.
- CMake [6] is a cross-platform build tool. CMake is used to define the C++ and Scyven software dependencies for building and linking. CMake can be used to generate makefiles for several systems, e.g. Visual Studio, or GCC/Make.
- Qt [7] is a cross-platform applications framework and was used to develop Scyven. Qt was chosen because of its use of native controls, well written documentation, large- user-base, its licensing scheme (with the option of moving to a commercial license) and its cross-platform nature.
- The Scyven code base was written in C++ using Microsoft Visual Studio C++ (MSVS). MSVS was chosen based on the engineer's experience at the time and provides a rich set of tools for developing and debugging code. However, MSVC, even the latest version, does not fully comply with the C++11 standard, and as it was our policy to use C++11 constructs where ever possible, there are some limitations to our use of C++11 because of this.

- The C++ API code is built primarily using GCC [8] and then tested in MSVC for compatibility and amendments made. Static code analysis using Red Lizards product Goanna assists in finding potential bugs, memory leaks and other issues.
- Armadillo [9] and BLAS/LAPACK [10] are used for algorithm acceleration and optimization. Armadillo provides a linear algebra library that has access to low level BLAS/LAPACK routines, allowing for considerable speedups.

3.3 Collaborations

3.3.1 Researchers

The engineering team works closely with researchers to understand existing research and algorithms, as well as to discuss new ideas, methods and concepts, and as such, development occurs reciprocally. Working effectively with researchers in computer vision necessitates a background in mathematics and imaging. Although all three engineers possess engineering qualifications and have skills that encompass mathematics and imaging, it is sometimes necessary to engage in self-study to acquire the additional skills and knowledge.

During the early development phase of Scyllarus, the researchers were our initial users, and provided us with numerous suggestions for improvement. Bugs and logic errors were discovered by both researchers and engineers. We found that this early user engagement was very useful, especially since we did not have any external users and could not evaluate the quality of the software in any independent manner.

Some algorithms developed by the researchers were very slow to run when implemented in C++ and were either modified, optimized or redesigned. Some algorithms were also abbreviated into 'fast' versions that produce slightly inferior results but massively reduced execution times.

Depending on the algorithms modified, different metrics were used to determine if the results produced by the faster versions were still correct. In some cases, speedups were gained by omitting certain parts of the result or making the method less thorough (e.g., by subsampling the image). The difference between the results for algorithms with unchanged outputs was measured using Mean Squared Error (MSE), and verifying if the faster version was still producing valid results. If the method was changed or replaced, more subjective measures were used to determine correctness in consultation with the researchers. In most cases, these optimizations are 'toggleable', and users can choose to run the 'full' version if they wish.

3.3.2 Business Team

The engineering team also interacts closely with a business team at NICTA to investigate commercial opportunities for Scyllarus. Through the work of the business developers, new end-users have been acquired who provide feedback. This feedback has been valuable as it has allowed us to eliminate bugs and enhance the software to meet end-user expectations.

4. DISCUSSION AND FUTURE DEVELOPMENT

Developing commercial software out of research grade code is always challenging, and we encountered a few issues while developing our tools. The challenges are discussed below.

4.1.1 Working with Research Code/Algorithms

Taking research code and translating or porting it into a commercial software package is generally a non-trivial process. The task involves studying the original code or algorithm, and potentially also reading theoretical materials (e.g., papers) to get a good understanding of what the algorithm is trying to accomplish before beginning to implement it. It is oftentimes the case that while the researcher who developed the algorithm had the best intentions when designing their code, the implementation is not ideal (especially when being re-written in a language such as C++) and vast improvements and optimizations can be made through effective redesign and reimplementation of the code.

There are many ways of making the process of translating research ideas and code into production code more efficient, although there are pros and cons for each. Researchers do not generally want to spend too much time performing software engineering activities, debugging or solving software meta-problems (library issues, environment setup etc.). This is the primary reason they work in languages such as MATLAB® on a day to day basis. As an initial step, researchers could follow a basic style guide for their programming (even if this were just for MATLAB®) that contained basic principles to make their code more readable and understandable. For example, naming variables explicitly can make a big difference, for instance 'brightness_threshold' rather than 'alpha').

Unless researchers have extensive experience in the target language, it is not necessarily more effective for them to work in it rather than MATLAB®. In our experience, code that came to us already written in C++ still required extensive refactoring to be made useful. Over the course of our project, we have found the most useful step in speeding up the integration process is to spend time discussing with the researchers what they intend their algorithms to do, then with that greater understanding, implement them quickly and effectively. In a few cases, the first 'in code' implementation of an algorithm was done by an engineer after such a discussion with a researcher.

4.1.2 Prioritizing Tasks and Delivery

In the early stages of the project, we were confronted with a vast base of research – algorithms, techniques and code – that could potentially all become a part of Scyllarus. The initial phase of development constituted selecting which of these components would be required to build the core functionality of Scyllarus, that is, which components would be required for the initial commercial product, and from that stage onward, which features and components would be subsequently added as enhancements and additional features. Throughout the development process, many novel and new features were conceived by the engineering team and researchers, and important new features were added as desired. The tools have undergone several releases to date. Some of these releases were prompted by business engagements (and included tasks allocated from the business team), while others were done as a matter of course.

4.1.3 *Libraries and Optimization*

While developing the C++ API and Scyven, care was taken to minimize unnecessary effort duplication where possible. Core libraries including Boost, QT and Armadillo were selected for use from the beginning due to their suitability, with additional library additions being carefully considered before their inclusion into the project. While licensing is a key consideration here, bloat was also a factor considered. In some cases, when a particular routine was needed and only offered by a 'large' or questionably supported library, the algorithm was implemented in house.

4.1.4 *Testing*

Testing of the commercial tools (Scyven and the C++ API) is imperative to the quality of the released software. While it is relatively easy to construct and execute unit testing suites for the C++ API, testing Scyven has proved to be a more difficult task due to the GUI nature of the software. The team continues to investigate a best practice solution for this.

4.1.5 *Future Development*

The current priority is to add capabilities to Scyven for the processing and analysis of earth observation data. These include spatial capability so that location-based information can be collected from satellite and aerial imagery.

Another priority for the team is to modify Scyven's GUI to make it suitable for touchscreen monitors and mobile devices, as well as increasing the ease, usability and intuitiveness. This would allow Scyven to be used out in the field, for example by forensic officers taking photos of a crime scene.

There is also work investigating the possibility of running Scyllarus on lightweight or embedded hardware such that it could be used in drone-based or other portable applications.

5. CONCLUSIONS

In this paper, we described the development of Scyllarus and the challenges that we experienced while developing commercial-grade software from research-grade code. The challenges included:

- Translating or porting research grade code into commercial software;
- Prioritizing tasks and delivery to build the core functionality of Scyllarus;
- Minimizing development effort by using appropriate open-source libraries and re-implementing some algorithms if their source was questionable; and
- Testing of Scyven due to the GUI nature of the software.

The merits of the tools that were used to develop the software are discussed as well as our experience in working with researchers and business development managers at NICTA.

6. ACKNOWLEDGEMENTS

We would like to thank Bill Simpson-Young and Ashley Stacey for reviewing our paper.

7. REFERENCES

[1] Smith, R. 2012. *Introduction to Hyperspectral Imaging*, MicroImages Inc., www.microimages.com.

[2] Scyllarus, www.scyllarus.com

[3] Robles-Kelly, A. and Huynh, C. P. 2003. *Imaging Spectroscopy for Scene Analysis*. Springer-Verlag, London.

[4] Scyven User Guide, http://scyllarus.research.nicta.com.au/resource/scyven/Scyven_User_Guide_1_0_3.pdf

[5] GitHub, github.com

[6] CMAKE, www.cmake.org

[7] Qt, www.qt.io

[8] GCC, gcc.gnu.org

[9] Armadillo, arma.sourceforge.net

[10] LAPACK/BLAS, netlib.org