Open Source Optical Coherence Tomography Software

Miroslav Zabic[1, 2], **Ben Matthias**[2], **Alexander Heisterkamp**[1], and **Tammo Ripken**[2]

1 Institute of Quantum Optics, Leibniz University Hannover, Welfengarten 1, 30167 Hannover, Germany **2** Industrial and Biomedical Optics Department, Laser Zentrum Hannover e.V., Hollerithallee 8, 30419 Hannover, Germany

DOI: 10.21105/joss.02580

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @jdavidli
- @brandondube

Submitted: 22 May 2020
Published: 29 October 2020

Summary

Optical coherence tomography (OCT) is a non-invasive imaging technique that is often described as the optical equivalent to ultrasound imaging. The basic building block of OCT acquisitions is an optical interference pattern that can be processed into a depth profile, which is also called A-scan. Several adjacent A-scans can be merged into a cross-sectional image. Most research that incorporates OCT requires a software solution for processing of the acquired raw data.

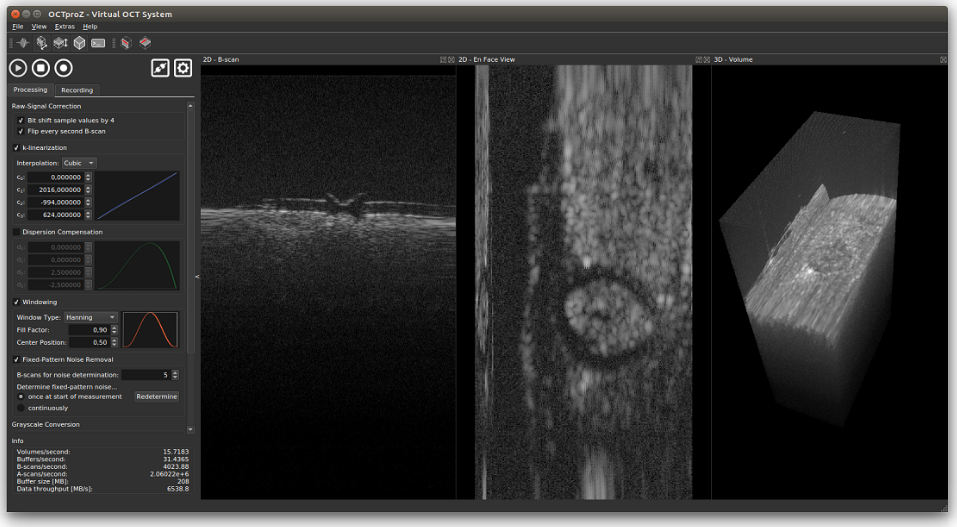
Here we present an open source software package for OCT processing with an easy to use graphical user interface. The implemented OCT processing pipeline enables A-scan processing rates in the MHz range. Custom OCT systems, or any other source of Fourier Domain OCT raw data, can be integrated via a developed plug-in system, which also allows the development of custom post processing modules.

1. Introduction

Optical coherence tomography (OCT) is a non-invasive imaging technique used primarily in the medical field, especially in ophthalmology. The core element of any OCT system is an optical interferometer that generates a spectral fringe pattern by combining a reference beam and the backscattered light from a sample. To obtain an interpretable image from this acquired raw OCT signal several processing steps are necessary, whereby the inverse Fourier transform represents an essential step. As the possible acquisition speed for raw OCT data has increased constantly, more sophisticated methods were needed for processing and live visualization of the acquired OCT data. A particularly impressive setup was presented by Choi et al. (Choi, Hiro-Oka, Shimizu, & Ohbayashi, 2012) that utilizes twenty FPGA-modules for real-time OCT signal processing and a graphics processing unit (GPU) for volume rendering. Nowadays, processing is typically done on graphics cards (Jian, Wong, & Sarunic, 2013; Rasakanthan, Sugden, & Tomlins, 2011; Sylwestrzak et al., 2012; Wieser et al., 2014; Zhang & Kang, 2010), not FPGAs, because implementing algorithms on GPUs is more flexible and takes less time (Li, Sarunic, & Shannon, 2011). Most of the publications that describe OCT GPU processing do not provide the actual software implementation. A commendable exemption is the GPU accelerated OCT processing pipeline published by Jian et al. The associated source code, which demonstrates an implementation of OCT data processing and visualization and does not include any advanced features such as a graphical user interface (GUI), already consists of several thousand lines. Thus, the most time consuming task of Fourier Domain OCT (FD-OCT) system development is not the optical setup, but the software development. The software can be separated into hardware control and signal processing, whereby the former being a highly individual, hardware-dependent software module and the latter being a generic software module, which is almost identical for many systems. To drastically reduce OCT system development time, we present OCTproZ, an open source OCT processing software package that can easily be extended, via a plug-in system, for many different hardware setups. In this paper we give a brief overview of the key functionality and structure of the software.

2. Basic overview of OCTproZ

OCTproZ performs live signal processing and visualization of OCT data. It is written in C++, uses the cross-platform application framework Qt (Qt, 2020) for the GUI and utilizes Nvidia's computer unified device architecture (CUDA) (NVIDIA, 2020) for GPU parallel computing. A screenshot of the application can be seen in Figure 1.



Raw-Signal Correction
✓ Bit shift sample values by 4
✓ Flip every second B-scan

✓ k-linearization
Interpolation: Cubic
c_0: 0,000000
c_1: 2016,000000
c_2: -994,000000
c_3: 624,000000

Dispersion Compensation
d_0: 0,000000
d_1: 0,000000
d_2: 2,500000
d_3: -2,500000

✓ Windowing
Window Type: Hanning
Fill Factor: 0,90
Center Position: 0,50

✓ Fixed-Pattern Noise Removal
B-scans for noise determination: 5
Determine fixed-pattern noise...
● once at start of measurement Redetermine
continuously

Grayscale Conversion

Info

```
Volumes/second:             15.7183
Buffers/second:             31.4365
B-scans/second:             4023.88
A-scans/second:          2.06022e+6
Buffer size [MB]:               208
Data throughput [MB/s]:      6538.8
```

Figure 1: Screenshot of OCTproZ v1.0.0 Processing settings visible in the left panel can be changed before processing is started or while processing is in progress. Processed data is live visualized in 2D as cross sectional images (B-scan and en face view) and in 3D as interactive volume rendering. The live view shows a piece of wood with a couple layers of tape and a laser burned hole.

The software can be separated into three parts: main application, development kit (DevKit) and plug-ins. The main application, OCTproZ itself, contains the logic for the GUI, processing and visualization. The DevKit, which is implemented as static library, provides the necessary interface for plug-in development. Plug-ins can be one of two kinds: "Acquisition Systems" or "Extensions". The former represent OCT systems and provide raw data to OCTproZ, the later are software modules that can extend the functionality of an OCT system (e.g., software control of a liquid lens) or provide additional custom defined post processing steps. All plug-ins are dynamic libraries that can be loaded into OCTproZ during runtime.

3. Processing Pipeline

Raw data, i.e. acquired spectral fringe pattern, from the OCT system is transferred to RAM until enough data for a user-defined amount of cross-sectional images, so-called B-scans, is acquired. Via direct memory access (DMA) this raw data batch is then copied asynchronously to GPU memory where OCT signal processing is executed. If the processed data needs to be stored or post processing steps are desired the processed OCT data can be transferred back to RAM with the use of DMA. An overview of the processing steps is depicted in Figure 2.

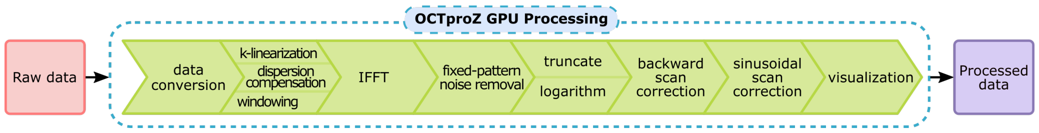


Figure 2: Processing pipeline of OCTproZ v1.2.0. Each box inside "OCTproZ GPU Processing" represents a CUDA kernel. Some processing steps are combinend into a single kernel (e.g. k-linearization, dispersion compensation and windowing) to enhance processing performance.

The processing pipeline consists of data conversion, k-linearization, numerical dispersion compensation, windowing, fast Fourier transform, fixed-pattern noise removal, truncate, logarithm, backward scan correction, sinusoidal scan correction and visualization. A detailed description of each processing step can be found in the user manual. Here we just want to mention that the implementation of the fixed-pattern noise removal is based on a publication by Moon et al. (Moon, Lee, & Chen, 2010) and the volume viewer is based on source code from an open source raycaster (Pilia, 2018). In order to avoid unnecessary data transfer to host memory, CUDA-OpenGL interoperability is used which allows the processed data to remain in GPU memory for visualization.

4. Processing Performance

Processing rate highly depends on the size of the raw data, the used computer hardware and resource usage by background or system processes. With common modern computer systems and typical data dimensions for OCT, OCTproZ achieves A-scan rates in the MHz range. Exemplary, Table 1 shows two computer systems and their respective processing rates for the full processing pipeline. However, since the 3D live view is computationally intensive the processing rate changes noticeably depending on whether the volume viewer is activated or not. The used raw data set consists of samples with a bit depth of 12, 1024 samples per raw A-scan, 512 A-scans per B-scan and 256 B-scans per volume. As the volume is processed in batches, the batch size was set for each system to a reasonable number of B-scans per buffer to avoid GPU memory overflow. It should be noted that this performance evaluation was done with OCTproZ v1.0.0 but is also valid for v1.2.0 if the newly introduced processing step for sinusoidal scan distortion correction is disabled.

Table 1: Comparison of two computer systems and their respective processing rates for raw data sets with 12 bit per sample, 1024 samples per raw A-scan, 512 A-scans per B-scan and 256 B-scans per volume.

	Office Computer	**Lab Computer**
CPU	Intel® Core i5-7500	AMD Ryzen Threadripper 1900X
RAM	16 GB	32 GB
GPU	NVIDIA Quadro K620	NVIDIA GeForce GTX 1080 Ti
Operating system	Windows 10	Ubuntu 16.04
B-scans per buffer	32	256
With 3D live view:		
A-scans per second	$\sim 250 \cdot 10^3$	$\sim 4.0 \cdot 10^6$
Volumes per second	~ 1.9	~ 30
Without 3D live view:		
A-scans per second	$\sim 300 \cdot 10^3$	$\sim 4.8 \cdot 10^6$
Volumes per second	~ 2.2	~ 36

5. Plug-in System

To develop custom plug-ins for OCTproZ and thus extend its functionality, a development kit is provided. It consists of a static library and a collection of C++ header files that specify which classes and methods have to be implemented to create custom plug-ins. Currently two kinds of plug-ins exist: Acquisition Systems and Extensions. For both we made examples including source code publicly available which may be used together with the open source and cross-platform integrated development environment Qt Creator as starting point for custom plug-in development.

The main task of an Acquisition System is to provide raw data to OCTproZ. In most cases, this means that the implementation of an Acquisition System contains the software control of a data acquisition unit.

Extensions have a wide area of use cases. As they are able to receive raw data and processed data via the Qt signals and slots mechanism, they are suitable for custom post-processing routines. The exact implementation of an Extension is mainly up to the developer and can also include hardware control. Therefore, Extensions are ideal for hardware control algorithms that rely on feedback from live OCT images. The best example of this is wavefront sensorless adaptive optics with a wavefront modulator such as a deformable mirror. Particular care must be taken if high speed OCT systems are used, as the acquisition speed of OCT data may exceed the processing speed of the custom Extension. In this case, a routine within the Extension should be implemented that discards incoming OCT data if previous data is still being processed.

6. Conclusion

In this paper, we introduced OCTproZ, an open source software package for live OCT signal processing. With the presented plug-in system, it is possible to develop software modules to use OCTproZ with custom OCT systems, thus reducing the OCT system development time significantly. OCTproZ is meant to be a collaborative project, where everyone involved in the field of OCT is invited to improve the software and share the changes within the community. We especially hope for more open source publications within the OCT community to reduce the time necessary for the replication of OCT processing algorithms and thereby accelerate scientific progress.

Funding

This work was partially funded by the European Regional Development Fund and the state of Lower Saxony as part of the project OPhonLas (EFRE-SER 2014-2020, 85007492).

EUROPÄISCHE UNION
Europäischer Fonds für
regionale Entwicklung

EUROPA FÜR
NIEDERSACHSEN
EFRE Förderung für die
niedersächsischen Regionen

References

Choi, D.-h., Hiro-Oka, H., Shimizu, K., & Ohbayashi, K. (2012). Spectral domain optical coherence tomography of multi-mhz a-scan rates at 1310 nm range and real-time 4D-display up to 41 volumes/second. *Biomedical optics express*, *3*(12), 3067–3086. doi:10.1364/boe.3.003067

Jian, Y., Wong, K., & Sarunic, M. V. (2013). Graphics processing unit accelerated optical coherence tomography processing at megahertz axial scan rate and high resolution video rate volumetric rendering. *Journal of biomedical optics*, *18*(2), 026002. doi:10.1117/1.JBO.18.2.026002

Li, J., Sarunic, M. V., & Shannon, L. (2011). Scalable, high performance fourier domain optical coherence tomography: Why fpgas and not gpgpus. In *2011 ieee 19th annual international symposium on field-programmable custom computing machines* (pp. 49–56). IEEE. doi:10.1109/fccm.2011.27

Moon, S., Lee, S.-W., & Chen, Z. (2010). Reference spectrum extraction and fixed-pattern noise removal in optical coherence tomography. *Optics express*, *18*(24), 24395–24404. doi:10.1364/OE.18.024395

Qt. (2020). Retrieved from https://www.qt.io

Rasakanthan, J., Sugden, K., & Tomlins, P. H. (2011). Processing and rendering of fourier domain optical coherence tomography images at a line rate over 524 kHz using a graphics processing unit. *Journal of biomedical optics*, *16*(2), 020505. doi:10.1117/1.3548153

Sylwestrzak, M., Szlag, D., Szkulmowski, M., Gorczynska, I. M., Bukowska, D., Wojtkowski, M., & Targowski, P. (2012). Four-dimensional structural and doppler optical coherence tomography imaging on graphics processing units. *Journal of biomedical optics*, *17*(10), 100502. doi:10.1117/1.JBO.17.10.100502

Wieser, W., Draxinger, W., Klein, T., Karpf, S., Pfeiffer, T., & Huber, R. (2014). High definition live 3D-oct in vivo: Design and evaluation of a 4D oct engine with 1 gvoxel/s. *Biomedical optics express*, *5*(9), 2963–2977. doi:10.1364/BOE.5.002963

Zhang, K., & Kang, J. U. (2010). Real-time 4D signal processing and visualization using graphics processing unit on a regular nonlinear-k fourier-domain oct system. *Optics express*, *18*(11), 11772–11784. doi:10.1364/oe.18.011772

NVIDIA. (2020). CUDA. Retrieved from https://developer.nvidia.com/cuda-zone

Pilia, M. (2018). GPU-accelerated single-pass raycaster. *GitHub repository*. GitHub. Retrieved from https://github.com/m-pilia/volume-raycasting