Research Article | Vol. 14, No. 11 / 1 Nov 2023 / Biomedical Optics Express 6060
Biomedical Optics EXPRESS

# OCTSharp: an open-source and real-time OCT imaging software based on C#

WEIHAO CHEN[1,2] AND HUI WANG[2,3,*]

[1]Department of Chemical, Paper, and Biomedical Engineering, Miami University, Oxford, OH, USA
[2]Department of Biology, Miami University, Oxford, OH, USA
[3]Department of Electrical and Computer Engineering Miami University, Oxford, OH, USA
[*]Hui.wang@miamioh.edu

**Abstract:** Optical coherence tomography (OCT) demands massive data processing and real-time displaying during high-speed imaging. Current OCT imaging software is predominantly based on C++, aiming to maximize performance through low-level hardware management. However, the steep learning curve of C++ hinders agile prototyping, particularly for research purposes. Moreover, manual memory management poses challenges for novice developers and may lead to potential security issues. To address these limitations, *OCTSharp* is developed as an open-source OCT software based on the memory-safe language C#. Within the managed C# environment, *OCTSharp* offers synchronized hardware control, minimal memory management, and GPU-based parallel processing. The software has been thoroughly tested and proven capable of supporting real-time image acquisition, processing, and visualization with spectral-domain OCT systems equipped with the latest advanced hardware. With these enhancements, *OCTSharp* is positioned to serve as an open-source platform tailored for various applications.



## 1. Introduction

Optical Coherence Tomography (OCT) has undergone extensive development over the past three decades. As modern hardware becomes more advanced, high-speed imaging often requires massive computational power to enable high-throughput data acquisition, processing, and visualization. A state-of-the-art Spectral Domain OCT (SD-OCT) can typically reach hundreds of kHz line rate with CMOS cameras, and Swept-Source OCT (SS-OCT) can perform MHz acquisition with GHz photodetectors [1–3]. To be able to achieve real-time image processing and display during high-speed imaging, Field Programmable Gate Arrays (FPGAs) and Graphic Processing units (GPU) are two common approaches for data processing pipelines [4]. However, compared to the FPGA method, GPU implementation is more expandable and flexible for agile development and modifications [5–8]. Although commercially available OCT systems already provide the software along with the devices, their source code is usually not accessible for custom modifications, and hardware customization or upgrade is usually not possible. As a result, it poses a challenge for research groups that aim to customize OCT for various applications, as the software may need to be modified to accommodate specific hardware or scanning protocols.

An open-source OCT imaging software would greatly benefit the OCT research community in exploring applications without limitations, especially for research groups with limited programming capabilities. Usually, C++ is considered the optimal choice for real-time imaging due to its efficiency and capability for low-level management of hardware resources. For example, *OctProz* is an OCT image-processing software that allows users to implement custom hardware binding [9]. *Vortex* is an open-source Application Programming Interface (API) with C++ or Python plug-in of hardware control and data processing for real-time OCT imaging software development [10]. However, C++ lacks built-in automatic memory management, commonly referred to as Garbage Collection (GC). Consequently, the developer is required to engage in manual pointer-based dynamic memory management. This becomes particularly challenging

#505308 https://doi.org/10.1364/BOE.505308
Journal © 2023 Received 6 Sep 2023; revised 13 Oct 2023; accepted 13 Oct 2023; published 31 Oct 2023

Biomedical Optics EXPRESS

in a multi-threaded environment, potentially leading to memory leaks. On the other hand, C# is a memory-safe language with automatic GC mechanism. In 2022, the National Security Agency (NSA) released guidance, suggesting a shift from C/C++ to memory-safe languages, such as C#, Java, or Rust to prevent vulnerabilities in software [11]. The "unsafe code" feature in C# allows pointer reference if necessary and the "fix" statement can guarantee memory safety during the pointer operation. [12]. In addition, C# is a managed programming language that is open-source, robust, easy to learn, and backed by a vast number of libraries for additional functional expansions, including C#'s native graphical user interface (GUI) framework and many other powerful APIs.

To the best of our knowledge, a complete OCT imaging solution in C# that consists of synchronized hardware control, high-speed data acquisition, GPU-based real-time processing, and image visualization has not been investigated, majorly due to the concern about "Stop-the-world" GC events. In this study, *OCTSharp* is developed based on .NET C# as a complete and ready-to-use software that aims to reduce programming complexity and create a user-friendly platform for interactive, intuitive, and high-performance OCT imaging software development [13].

## 2. Method

### 2.1. Hardware configuration

The hardware configuration of the SD-OCT for testing *OCTSharp* is shown in Fig. 1(A). The SD-OCT imaging system consists of a broadband light source (a superluminescent LED at 1310 nm or 850 nm), a fiber-based interferometer, a linear camera, a Frame Grabber (FG), a Digital Acquisition Card (DAC), and a host computer (PC) equipped with Graphics Processing Unit (GPU) card. Various hardware configurations were tested to validate the software's compatibility and performance (Table. 1). Three commonly used linear cameras with line rates at 36kHz, 147kHz, and 250kHz were included to show the compatibility of the software [14–16]. Different DACs that were previously used in other OCT studies were also included to show the hardware expandability [16–19]. Two computers configured with different hardware were built to benchmark the imaging performance.

The triggering and synchronization clock logic is shown in Fig. 1(B). To synchronize frames during imaging acquisition and X-Y scanning, the 'Strobe' signal of the FG (Xtium-CL MX4) is used as the master clock signal, which is, in fact, the Horizontal Synchronized (HSync) signal rerouted from the camera, as depicted in Fig. 1(B). Physically, the master clock is wired from the FG to a digital input port of the DAC (PCI-6221 or PCIe-6361). To control the galvanometer scanner, the software initiates an analog output task to update the voltages of two analog outputs on the DAC based on a pre-generated voltage table. By default, a sawtooth scanning pattern is used with an adjustable fly-back cycle for different duty cycle operations. However, different scanning patterns can be supplied by modifying the voltage table. Additionally, the DAC generates a frame sync signal based on the number of A-scans in each B-Scan; this signal is also physically wired from a digital output port of the DAC to the FG, as shown in Fig. 1(A). During imaging acquisition, OCT images are acquired only during forward scanning. For precise synchronization, the camera is set to external triggering mode, which enables the camera to acquire images entirely based on the frame sync signal. Both analog (Scanner) and digital (Frame Sync) tasks are configured to start when the digital input on the DAC detects the rising edge of the master clock from the FG. This ensures that all hardware is synchronized and starts at the same time. In case of any potential time delay between the frame sync and the actual galvo scanner's position, a tick-based initial time delay can be introduced to the frame sync to accommodate the time difference.

Biomedical Optics EXPRESS

**Table 1. Configuration and performance of two different hardware platforms. $T_s$: B-Scan Acquisition Cycle Time. $T_p$: B-Scan Processing Time.**

| | PC 1 | | PC 2 |
|---|---|---|---|
| **CPU** | Intel E5-1607 3.1GHz | | AMD 3970X 3.7GHz |
| **GPU** | NVIDIA GeForce RTX 2060 1.37GHz | | NVIDIA GeForce RTX 3080 1.44GHz |
| **RAM (GB)** | 40 | | 128 |
| **Light Source** | SUPERLUM SLD-mCS 850nm | | THORLABS 1310nm SM |
| **Camera** | DALSA OCTOPLUS | AVIIVA SM2 4010 CL | SUL GL 2048R |
| **Camera Line Rate (kHz)** | 250 | 36 | 147 |
| **Pixels of Camera** | 2048 | 2048 | 2048 |
| **A-Scans Per B-Scan** | 1000 | 1000 | 1000 |
| **Frame Grabber** | DALSA Xtium-CL MX4 | DALSA Xtium-CL MX4 | DALSA Xtium-CL MX4 |
| **Digital Acquisition Card** | NI PCI-6221 | NI PCI-6221 | NI PCIe-6361 |
| **$T_S$ w/ 32.2% Duty Cycle (ms)** | 5.9 | 41 | 10 |
| **Minimum $T_p$ (ms)** | 2.09 | | 0.81 |
| **Average $T_p$ (ms)** | 2.38 | | 1.02 |
| **Maximum $T_p$ (ms)** | 12.18 | | 2.38 |

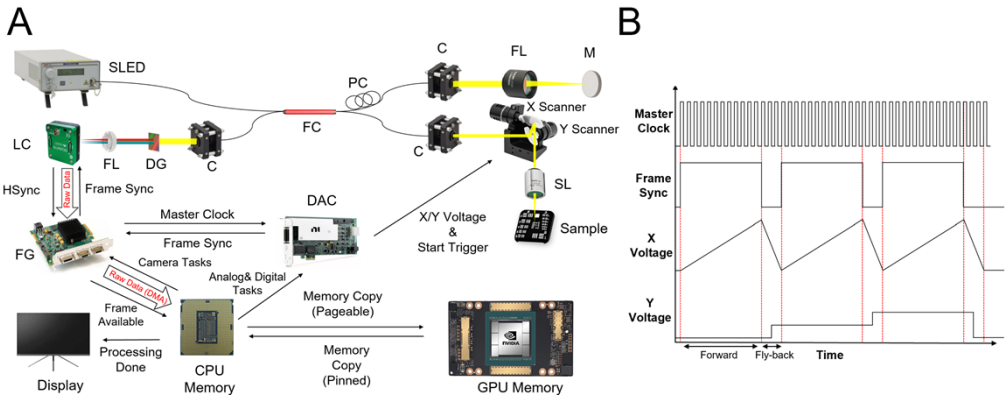


**Fig. 1.** The configuration of the SD-OCT system. A) The hardware connection and triggering mechanism of the SD-OCT system. SLED: Super Luminescent LED; C: Collimator; PC: Polarization Controller; FL: Focusing Lens; M: Mirror; DG: Diffraction Grating; LC: Linear Camera; FG: Frame Grabber; DMA: Direct Memory Access; DAC: Digital Acquisition Card; CPU: Central Processing Unit; GPU: Graphical Processing Unit. B) The timing diagram of the master clock to synchronize the camera using frame sync, and the corresponding timing of the X-Y galvanometer scanner's voltage position.

Biomedical Optics EXPRESS

### 2.2. Software architecture

Software architecture with multi-threading and GPU-based Compute Unified Device Architecture (CUDA) implementation is efficient for real-time OCT imaging in C++ [5–9]. *OCTSharp* shares a similar architecture but is implemented in C#. The hardware control is achieved using Software Development Kits (SDKs), including Sapera LT 8.6 from Teledyne DALSA and DAQmx 20.1 from National Instruments [20,21]. To seamlessly interact with the CUDA library in C#, *OCTSharp* utilizes *ManagedCuda*, an open-source API that wraps the CUDA library in C# [22]. *ManagedCuda* provides a type-safe and object-oriented way to access CUDA resources in a class-based reference to CUDA API, as opposed to the pointer-oriented manner in C++ or C. Notably, all CUDA kernels are written in CUDA C and pre-compiled into a dedicated PTX file using the NVIDIA CUDA Compiler (NVCC) and it is loaded prior to the runtime for real-time imaging processing.

Figure 2 provides a detailed description of the multi-threading logic with the relationship between the hardware and software. The FG has a default ring buffer (or cycling buffer) structure to store raw frames (spectral data) from the camera. A Direct Memory Access (DMA) controller handles data transfer from the FG to the frame buffer in the host memory. If the ring buffer becomes filled, it triggers a frame lost event, which is used as a monitor during imaging to alert potential raw data loss. During imaging acquisition, four dedicated threads are launched on the CPU by *OCTSharp*, including the Graphical User Interface (GUI) thread, the transfer thread, the process thread, and the display thread. The GUI thread primarily manages real-time events, such as user interactions, visualization of benchmark parameters, and charting updates. The transfer thread is initiated by a callback function triggered by the frame available signal whenever a frame is ready in the frame buffer. It then notifies the process thread to commence the OCT imaging processing pipeline. Additionally, the transfer thread handles the transfer of raw frames from the frame buffer to a preallocated volume buffer in the host memory when a user requests to save the raw data before imaging acquisition. After the imaging section, the raw frames in the volume buffer will be saved automatically to a local drive for post-processing.

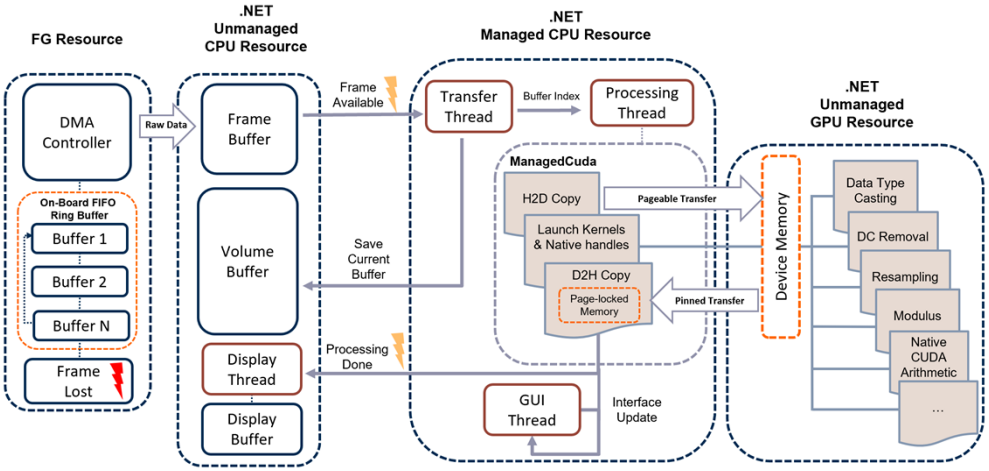


**Fig. 2.** *OCTSharp* Software Architecture. On-board Ring Buffer: Default FIFO memory on the FG; DMA: Direct Memory Access; Frame Lost Event: a software event when the on-board ring buffer overflows; Frame buffer: A buffer for saving a raw B-Scan; Volume Buffer: A pre-allocated buffer for saving raw C-Scan; H2D: data transfer from the host frame buffer to the device memory in a GPU; D2H: data transfer from the device memory to a host page-locked memory; The flash signs represent a software trigger callback event.

Biomedical Optics EXPRESS

Once the process thread is initiated, a raw frame in the frame buffer is transferred to the device memory in the GPU, also known as Host to Device (H2D) memory copy. Then, the process thread launches GPU kernels through *ManagedCuda* and sequentially processes the raw frame to an OCT B-Scan image. When the processing is completed, the process thread transfers the processed data from the device memory to a preallocated and pinned host memory, also known as Device to Host (D2H) memory copy. Then the Processing Done callback event will initiate the display thread to update the latest image on the GUI. The GUI thread ensures that monitoring parameters, such as B-scan processing duration, frame rate, frame loss alerts, etc. are consistently updated. It should be noted that managed resources are in the scope of the Common Language Runtime (CLR), which is part of the .NET framework, while unmanaged resources are out of the scope of CLR. Three major buffers, frame buffer, volume buffer, and display buffer, used for data transfer are not managed by the CLR.

### 2.3. Animal handle and ethics

Our previous work demonstrated that OCT is the ideal *in vivo* imaging modality for lens regeneration study in newts [23]. In this study, Pleurodeles waltl is used as the animal model for the *in vivo* imaging demonstrations of *OCTSharp*. The protocol has been approved by Miami University's Institutional Animal Care and Use Committee (IACUC). Before *in vivo* imaging, the animal is anesthetized and positioned on a specialized imaging stage with 6 degrees of freedom. Water is applied to the cornea during imaging to prevent reflections from dehydration. Imaging takes around 5 minutes per individual. After the experiment, the animal is returned to its habitat in the facility.

## 3. Results and discussion

### 3.1. Validation of hardware compatibility

Two hardware configurations were used to validate the performance of *OCTSharp,* as shown in Table 1. Two CameraLink FGs, Xtium-CL MX4 and Xcelera-CL LX1 (Teledyne DALSA), and two DACs, PCIe-6361 and PCI-6221 (National Instrument), were tested without modifying the source code. Xtium-CL MX4 was used as the main FG to benchmark the performance of two PC configurations. Three linear cameras were also tested including OCTOPLUS (Teledyne DALSA), AVIIVA SM2 4010 CL (Teledyne DALSA), and GL 2048R (Sensor Unlimited) (Table. 1). The buffer of acquired data is stored as a scattered memory in the host memory, thus the image size is only limited by the maximum available system memory in the host computer.

The B-Scan processing time includes memory transfer (H2D & D2H) and the time of all processing kernels for each B-scan. In our cases, each B-scan has a size of 2048 (the number of pixels of the camera) x1000 (the number of A-scan per B-scan). PC2 exhibits significantly shorter single B-scan processing times compared to PC1, thanks to the more efficient CPU and GPU with larger bandwidth. Most LCDs can display images at 60 Hz, corresponding to 16 ms per frame. It is shown that even in the worst-case scenario, with the Maximum B-scan Processing Time shown in Table. 1, *OCTSharp* can still support real-time image displaying at 60 frames per second (FPS) for both configurations, satisfying the requirement of real-time monitoring during *in vivo* imaging. However, to process every incoming B-scan without missing, the B-scan processing time must be less than the B-scan acquisition time, determined by the B-scan acquisition rate. For instance, with the duty cycle set at 32.2%, the B-scan acquisition times for three cameras operating at line rates of 36kHz, 147kHz, and 250kHz are 41 ms, 5.9 ms, and 10 ms, respectively, which correspond to B-scan acquisition rates at 24 Hz, 169 Hz, and 100 Hz.

Biomedical Optics EXPRESS

### 3.2. Evaluation of CUDA processing efficiency

OCT imaging processing can benefit from the massively parallel execution capability of GPUs to significantly reduce data processing time. However, CUDA C is typically limited to run natively in C++ applications and cannot be directly used in C#. APIs like ILGPU allow users to write GPU-accelerated code in C# with high-level abstraction functions [24]. These APIs utilize a Just-In-Time (JIT) compiler to translate C# code into corresponding CUDA code at runtime. In contrast, *ManagedCuda* directly refers to the original CUDA library and compiles code with the native NVCC, allowing more fine-grained control over GPU operations. This approach has its advantages from a software development perspective. While the JIT compiler method eliminates the need to write CUDA C code, *ManagedCuda* provides a more generic CUDA referencing experience yet still preserves the highly efficient performance that is comparable to C++ solutions [9].

Figure 3 compares the execution times of data transfer of H2D, the CUDA kernels, and D2H. Regardless of the configuration, PC1 or PC2, the H2D process consumes the most significant portion of the B-scan processing time, accounting for approximately 32% of the total processing time. When transferring a 16-bit B-Scan buffer ($2048 \times 1000$) with a size of approximately 4.19 megabytes (MB), the average transfer time is approximately 0.79 milliseconds (ms) on PC1 and 0.32 ms on PC2. However, during D2H, when dealing with an 8-bit B-Scan buffer of around 1.04 MB in size, the transfer time is significantly reduced, it takes only 0.08 ms on PC1 and 0.05 ms on PC2.

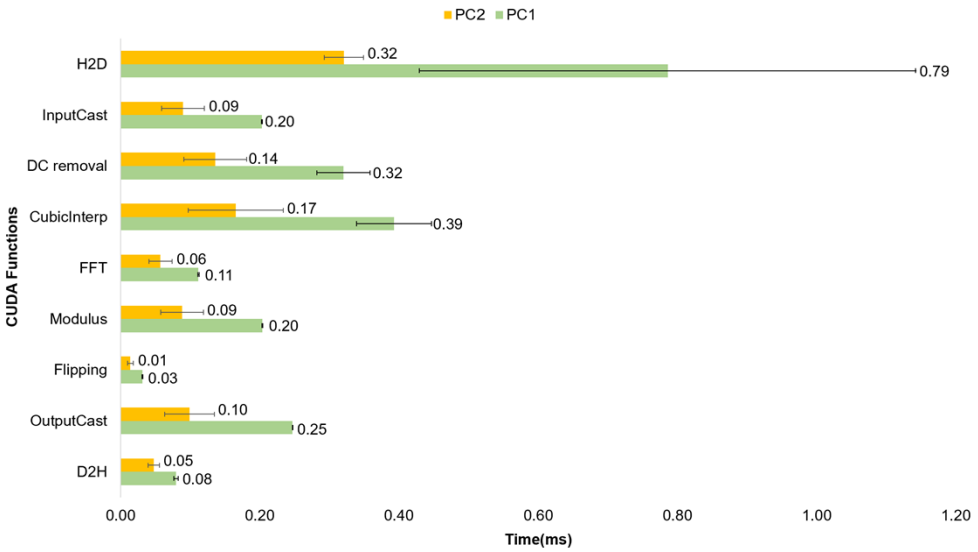


**Fig. 3.** Benchmark of the execution times of all CUDA kernels for a B-Scan with a size of $2048 \times 1000$. H2D: host memory to device memory. D2H: device memory to host memory. InputCast: datacasting from 16-bit unsigned integer data type to 32-bit float point data type. OutputCast: datacasting from 32-bit float point data type to 8-bit unsigned integer data type. DC Removal: the DC background removal of an OCT B-scan. FFT: Fast-Fourier-Transfer using Cufft handle. Modulus: Calculating the magnitude of the FFT results. Flipping: B-Scan transposition using Cublas handle. CubicInterp: cubic interpolation for resampling. The error bar stands for the standard deviation of each kernel.

The different data transfer rates between the H2D and D2H are caused by different memory transfer mechanisms. The data transfer between host memory and GPU devices requires pinned or page-locked memory in the host memory. During H2D, the frame buffer in the host is allocated

Biomedical Optics EXPRESS

by Sapera LT. The frame buffer is recognized by the GPU as a pageable memory. The pageable memory has to be implicitly converted to a temporary pinned memory, and then the data can be transferred to the device memory in the GPU. In contrast, during D2H, the buffer for storing the processed frame data is directly allocated by *ManagedCuda* as a pinned host memory, which allows the data to be transferred via the DMA mechanism. The relatively slow peggable memory transfer mechanism during H2D may become the bottleneck of the processing time, limiting the real-time B-scan processing speed when the camera speed is further improved in the future. In this case, it is worth mentioning that *OCTSharp* is still capable of acquiring high-speed data, because the imaging acquisition itself is not limited by the H2D mechanism. To resolve the potential real-time processing bottleneck, a page-locked agreement on the host memory has to be established between the FG and GPU. One potential solution is adapting FGs that support NVIDIA GPUDirect, which can transfer the camera raw data directly to the GPU device memory via DMA, eliminating any potential overhead associated with memory copy between host memory and GPU memory.

### 3.3. Garbage collection in C#

Different memory management mechanisms are adapted in C++ and C#. In C++ applications, memory safety issues, such as buffer overflows or null pointer dereferences, can lead to unpredictable crashes, or security vulnerabilities. Developers must be diligent in managing memory properly to avoid these issues. In contrast, C# provides automatic memory management known as auto GC. The .NET CLR constantly evaluates the life span of the managed objects and releases unused resources to free the memory when necessary. This automatic memory management mechanism releases the developer from the hassle of dynamic memory management. However, GC can stop the managed threads. This is often referred to as "stop the world" or "pause the world". To minimize the impact of "stop the world", the .NET categorizes GC events into 3 different levels, Gen 0, Gen 1, and Gen 2, where Gen 2 GC is known as a full GC that takes the longest time to collect all unused objects from all three depth, while the other two levels take a much shorter time, but they all suspend the managed threads once occur.

Evaluating the impact of GC on OCT real-time imaging display and raw data integrity is crucial. During imaging, OCT software should display acquired images, typically B-scan images, in real time to provide feedback to the user. For instance, during *in vivo* retinal imaging, ophthalmologists require real-time displayed images to guide patients in capturing images of specific regions of interest. The image display rate is inherently constrained by the refresh rate of the LCD utilized in the system. Typically, a frame rate of 30 FPS is sufficient enough for real-time monitoring. In advanced OCT systems, B-scan images can be acquired at speeds of up to a few hundred FPS. Not all acquired images are necessary for display, but it is essential to save all raw data for post-processing tasks such as 3D reconstruction or angiography.

During high-speed image acquisition, the occurrence of GC events is inevitable and can pose potential risks of frame loss. Figure 4 illustrates the time diagram of B-scan acquisition and processing cycles with and without GC. To ensure real-time processing of every incoming B-scan, the B-scan processing time without accounting for GC, denoted as $T_P$, must be shorter than the B-scan acquisition cycle time, $T_S$, as depicted in the in Fig. 4. For example, for imaging acquisition with PC2 shown in Table. 1, the maximum B-scan processing time without accounting GC, $T_P$ (2.3 ms), is much less than $T_S$ ($10\,T_{s1}$). Therefore, all acquired images will be timely processed. However, when GC occurs during an acquisition cycle, managed threads like the transfer thread or the processing thread will experience interruptions. Consequently, the total B-scan processing time, $T_{TP}$, including $T_P$ and the interruption time of GC, $T_{GC}$, can be larger than $T_S$, shown $T_{s2}$ in Fig. 4. The incoming frame in that cycle cannot be processed promptly, potentially leading to frame loss. Figure 5 plots the histogram of $T_{TP}$ distribution recorded continuously through three-hour live imaging with PC2 setup. The red vertical line shown in

Biomedical Optics EXPRESS

Fig. 5 represents the location of $T_S$. Most $T_{TP}$ are between 1.19 ms to 2.19 ms, which is far less than $T_S$. However, $T_{TP}$ can be significantly longer, up to around 12.3 ms due to GC events and the variation of data transfer and processing shown in Fig. 3. In about 99.94% of all cycles, $T_{TP}$ is less than $T_S$. On the other hand, in about 0.06% of cycles, $T_{TP}$ is longer than $T_S$, the case depicted as $T_{s2}$ in Fig. 5. In these cycles, due to overtime B-scan processing time, incoming frames cannot be processed before the next frame is ready. The next frame may be lost.

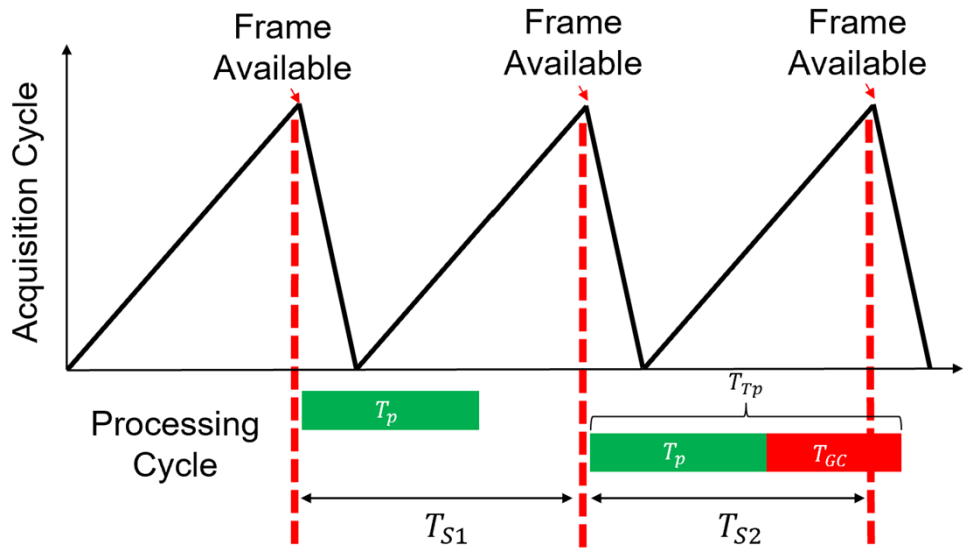


**Fig. 4.** Illustration of the relationship between the acquisition cycle and the processing cycle of each B-Scan. $T_p$: B-scan processing time. $T_{GC}$: Garbage Collection time. $T_{TP}$: Total B-scan processing time when GC events occur. $T_{s1}$, $T_{s2}$ : B-Scan acquisition cycle time.

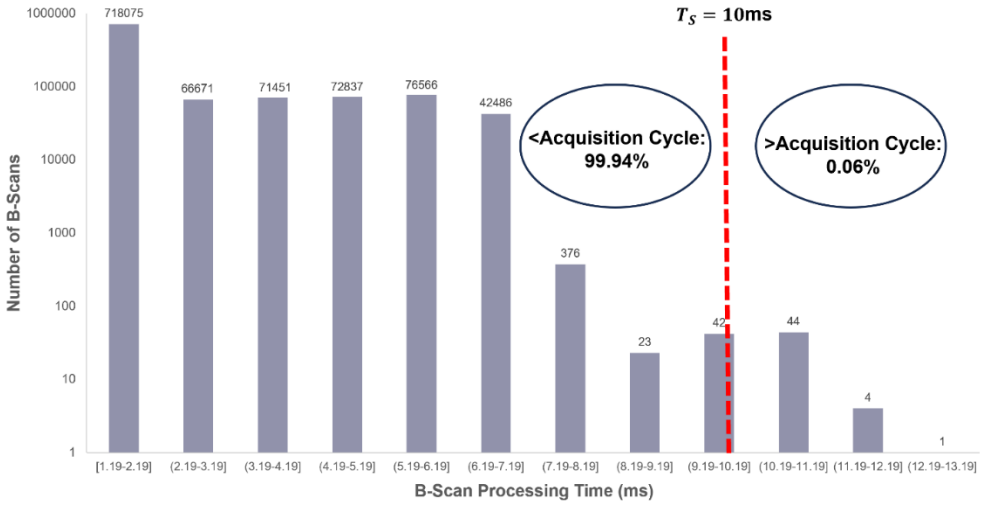


**Fig. 5.** The histogram of the $T_p$ for all acquired frames during a 3-hour imaging section with a B-Scan image size of $2048 \times 1000$ using PC2. $T_s$ is 10 ms as shown by the red-dotted line, which is equivalent to a B-Scan acquisition rate at 100 Hz.

Biomedical Optics EXPRESS

In *OCTSharp*, the potential frame loss due to GC is accommodated from three aspects. First, we pre-allocate large data buffers, like the frame, volume, and display buffer, as unmanaged resources to avoid GC2 events; Second, we avoid creating unnecessary objects by reusing them to reduce the frequency of GC events. As a result, the total time fraction of GC during imaging acquisition is controlled at less than 0.1% of the total acquisition time. Third, we adopted an FG (Xtium-XL MX4) with a ring buffer mechanism to queue frames if they cannot be processed on time. In our case, the FG has an on-board memory space of 512 MB. For a B-Scan with a size of $2048 \times 1000$, 131 ring buffers are allocated on the FG. When *OCTSharp* cannot process the current frame in the frame buffer on time due to GC, the following frames from the camera will be stored in the ring buffer. Once GC events are completed, all the frames queued in the ring buffer will be processed based on the FIFO manner. Note the FIFO mechanism can work effectively only if the time fraction attributed to overtime B-scan processing is very small. If the time fraction of overtime B-scan processing time is significant, even a sufficiently large ring buffer that can store all queued images, the images will be displayed with a noticeable time delay caused by the cumulative overtime events. In our case, the fraction of overtime B-scan processing time is only 0.06% (Fig. 5). Thus, no frame lost or delayed display occurred during the entire 3-hour live imaging section.

### *3.4. Software functionality, in vivo imaging demo and expandability*

#### 3.4.1. Software functionality

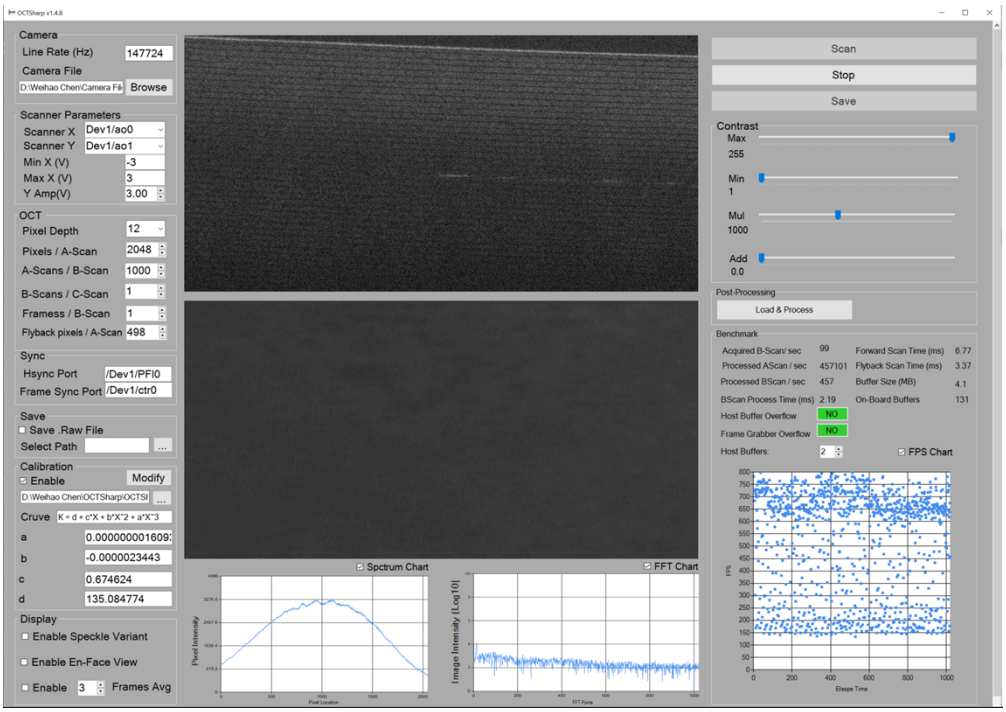


**Fig. 6.** GUI of *OCTSharp* that includes the major display windows as well as adjustable parameters and controls. For more details about the installation and compilation guidelines, refer to https://github.com/OCTSharpImaging/OCTSharp.

A basic GUI for OCTSharp is illustrated in Fig. 6. With this interface, users can manage hardware parameters of the camera and galvanometer scanner, set OCT imaging parameters, and select

Biomedical Optics EXPRESS

display modes, including average display, en-face display, and speckle variance display. Raw data is saved in .raw format for post-processing. The GUI also provides a list of key benchmark parameters to facilitate optical system alignment, such as spectrum monitoring.

#### 3.4.2. *In vivo* imaging demo

Human fingernails and the Newt anterior chamber were utilized to showcase the *in vivo* imaging performance of *OCTSharp* in Fig. 7. Given the high scattering properties of human skin [25], *in vivo* imaging of the fingernail was conducted using PC2 at 1310 nm, and shown in Fig. 7(A). Here, the real-time B-Scan of the fingernail shows various tissue structures, including the nail bed, cuticle, nail fold, nail root, epidermis, and dermis. Subsequently, a C-scan comprising 512 B-scans was acquired and saved. Raw data are post-processed with MATLAB (R2023a) and the volumetric view is visualized with ImarisViewer (9.9.1, Oxford Instrument) as shown in Fig. 7(B).

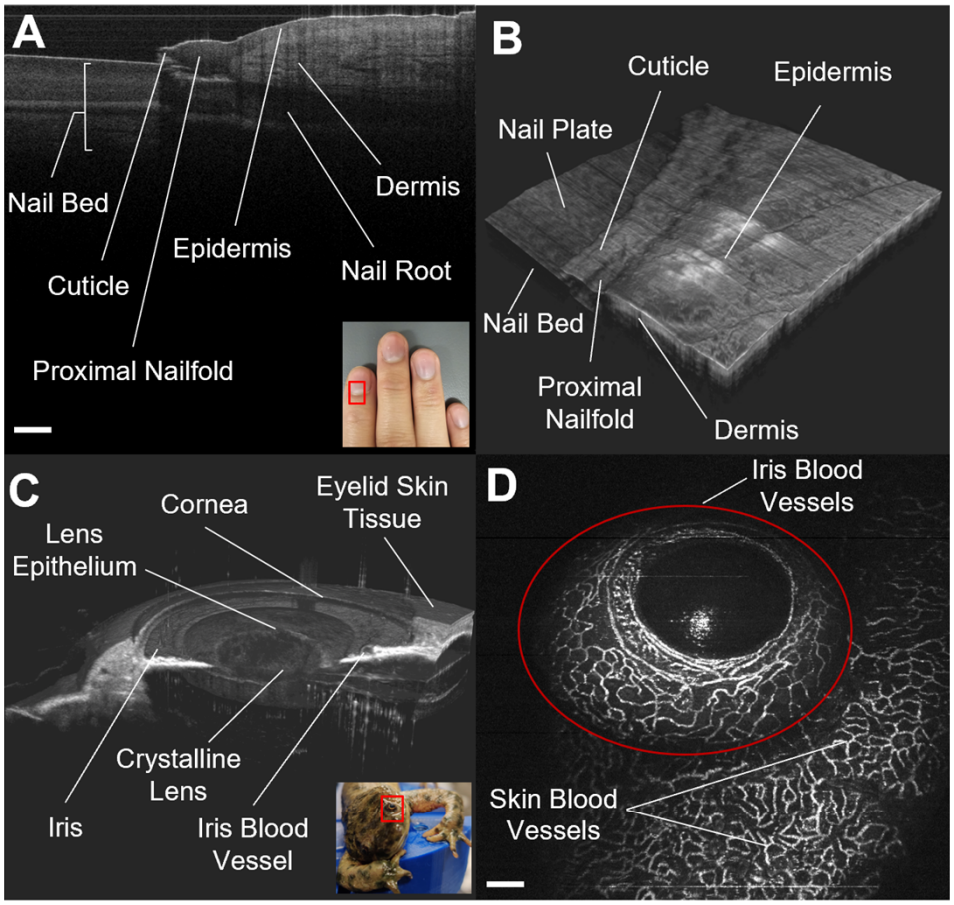


**Fig. 7.** In vivo images acquired with *OCTSharp*. A: B-Scan of the fingernail structure inside the red rectangle area captured with PC2; B: C-Scan perspective view of A; C: Cut-out-view of the Newt's anterior chamber inside the red rectangle area captured with PC2; D: Optical Coherence Tomography Angiography of the Newt's iris and the surrounding skin tissue, captured with PC1. Scale bar: 100 µm.

In our previous work, we demonstrated the unique capability of using OCT to *in vivo* monitor the lens regeneration process of Newts [26]. Therefore, we continue using this animal model to

Biomedical Optics EXPRESS

validate the performance of *OCTSharp*. In particular, we want to test if the raw data captured by *OCTSharp* can be used for reconstructing the microvascular network in the iris. Since the anterior chamber is transparent, we take advantage of the wavelength of PC1 at 850 nm, which can capture images with lateral and axial resolution at ~7 μm and the line rate at 250kHz. Five B-scans are captured at each location, resulting in a total of $5 \times 512$ B-scans, which are saved as raw data. Figure 7(C) shows the 3D reconstructed anterior chamber of a newt's eye, highlighting all morphological tissue structures. Figure 7(D) shows the microvascular network of the iris, extracted using the speckle variance technique [27]. These images confirm that *OCTSharp* can acquire high-quality B-scan images for reconstructing volumetric images of general tissue structure and the microvasculature of the iris.

#### 3.4.3. Software expandability

*OCTSharp* is developed as a foundational software platform that can be modified and expanded based on the application requirements of users. The software itself can be separated into three major components, including the GUI framework built with .NET, hardware control developed with Sapera LT and DAQmx, and image processing implemented with *ManagedCuda*. If a new camera is adopted, as long as the camera supports Cameralink, the user just needs to configure a new camera file. To implement additional custom CUDA processing functions, new functions can be easily added to the same CUDA file along with other kernels in Fig. 3. A potential limitation of *OCTSharp* is its dependency on Sapera LT from Teledyne DALSA and DAQmx from National Instruments for hardware control. However, these SDKs have been well-documented and well-supported. Lastly, the Microsoft .NET community offers abundant 3$^{rd}$ party API resources, which allows further functional expansion for *OCTSharp*. For example, advanced image display features like volume rendering can be achieved using OpenTK [28], and image-level classification or segmentation can be accomplished with OpenCV [29].

## 4. Conclusion

In conclusion, *OCTSharp* is an open-source C#-based platform for real-time OCT software development. Real-time imaging display and raw data integrity are essential for OCT imaging. Compared to the commonly used C++ in OCT software development, C# is more user-friendly and is a memory-safe language. However, doubts have been raised about the suitability of developing high-performance OCT applications with C# due to the 'stop the world' issue induced by the GC mechanism in C# and the challenge of high-throughput data processing. In this paper, we addressed these concerns by verifying that *OCTSharp* can indeed meet the state-of-the-art OCT imaging requirements. *ManagedCuda* enables C# to conveniently access GPUs and leverage their parallel computation capabilities. To overcome the challenges posed by the C# GC mechanism, we implemented strategies to minimize GC pressure and adopted a frame grabber with a ring buffer to prevent frame loss. Our test results show that the impact of GC on real-time imaging can be safely disregarded. These findings reinforce our belief that *OCTSharp* can be customized for various applications, greatly simplifying the development of real-time OCT imaging software.

**Funding.** National Institute of Biomedical Imaging and Bioengineering (R21 EB033993); National Eye Institute (R21EY031865).

**Acknowledgments.** We want to thank Dr.Katia Del Rio-Tsonis from the biology department at Miami University for providing the newt sample for OCT *in vivo* imaging.

**Disclosures.** The authors declare no conflicts of interest.

**Data availability.** Data supporting the findings of this paper are currently unavailable to public but can be obtained from the authors upon request.

Biomedical Optics EXPRESS

## References


1. B. Tan, Z. Hosseinaee, L. Han, *et al.*, "250 kHz, 1.5 µm resolution SD-OCT for in-vivo cellular imaging of the human cornea," Biomed. Opt. Express **9**(12), 6569–6583 (2018).
2. T. S. Kim, J. Joo, I. Shin, *et al.*, "9.4 MHz A-line rate optical coherence tomography at 1300 nm using a wavelength-swept laser based on stretched-pulse active mode-locking," Sci. Rep. **10**(1), 9328 (2020).
3. T. Huo, C. Wang, X. Zhang, *et al.*, "Ultrahigh-speed optical coherence tomography utilizing all-optical 40 MHz swept-source," J. Biomed. Opt **20**(3), 030503 (2015).
4. D. Choi, H. Hiro-Oka, K. Shimizu, *et al.*, "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," Biomed. Opt. Express **3**(12), 3067–3086 (2012).
5. Y. Huang, X. Liu, and J. U. Kang, "Real-time 3D and 4D Fourier domain Doppler optical coherence tomography based on dual graphics processing units," Biomed. Opt. Express **3**(9), 2162–2174 (2012).
6. Y. Jian, K. Wong, and M. V. Sarunic, "Graphics processing unit accelerated optical coherence tomography processing at megahertz axial scan rate and high resolution video rate volumetric rendering," J. Biomed. Opt. **18**(02), 1 (2013).
7. C. Chen, W. Shi, and V. X. D. Yang, "Real-time en-face Gabor optical coherence tomographic angiography on human skin using CUDA GPU," Biomed. Opt. Express **11**(5), 2794–2805 (2020).
8. Y. Ling, X. Yao, and C. P. Hendon, "Highly phase-stable 200 kHz swept-source optical coherence tomography based on KTN electro-optic deflector," Biomed. Opt. Express **8**(8), 3687–3699 (2017).
9. M. Zabic, B. Matthias, A. Heisterkamp, *et al.*, "Open Source Optical Coherence Tomography Software," JOSS **5**(54), 2580 (2020).
10. M. Draelos, "Vortex — High-Performance OCT Library," Image-Guided Medical Robotics Lab, 2023, https://www.vortex-oct.dev/.
11. "NSA Releases Guidance on How to Protect Against Software Memory Safety Issues," https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/nsa-releases-guidance-on-how-to-protect-against-software-memory-safety-issues/.
12. P. A. Laplante and S. J. Ovaska, *Real-Time Systems Design and Analysis: Tools for the Practitioner*, 4th edition (Wiley-IEEE Press, 2011).
13. W. Chen and H. Wang, "OCTSharp: an open-source C# software for OCT," in *Optical Coherence Tomography and Coherence Domain Optical Methods in Biomedicine XXVII*, J. A. Izatt, eds. (SPIE, 2023), p. 101.
14. Y. Shao, A. Tao, H. Jiang, *et al.*, "Long scan depth optical coherence tomography on imaging accommodation: impact of enhanced axial resolution, signal-to-noise ratio and speed," Eye and Vis **5**(1), 16 (2018).
15. L. Bernstein, A. Ramier, J. Wu, *et al.*, "Ultrahigh resolution spectral-domain optical coherence tomography using the 1000–1600 nm spectral band," Biomed. Opt. Express **13**(4), 1939–1947 (2022).
16. D. G. Revin, R. A. Byers, M. Q. Duan, *et al.*, "Visible-light optical coherence tomography platform for the characterization of the skin barrier," Biomed. Opt. Express **14**(8), 3914 (2023).
17. K.-S. Chiu, M. Tanifuji, C.-W. Sun, *et al.*, "Temporal mirror-symmetry in functional signals recorded from rat barrel cortex with optical coherence tomography," Cereb Cortex **33**(8), 4904–4914 (2023).
18. D. J. Wahl, R. Ng, M. J. Ju, *et al.*, "Sensorless adaptive optics multimodal en-face small animal retinal imaging," Biomed. Opt. Express **10**(1), 252–267 (2019).
19. Q. Li, K. Karnowski, G. Untracht, *et al.*, "Vectorial birefringence imaging by optical coherence microscopy for assessing fibrillar microstructures in the cornea and limbus," Biomed. Opt. Express **11**(2), 1122–1138 (2020).
20. "Sapera LT | Teledyne DALSA," https://www.teledynedalsa.com/en/products/imaging/vision-software/sapera-lt/.
21. "NI-DAQTMmx Download," https://www.ni.com/en-us/support/downloads/drivers/download.ni-daq-mx.html.
22. M. Kunz, C. Bovar, and M. Zelinka,"managedCuda," Github, 2015, https://kunzmi.github.io/managedCuda/.
23. W. Chen, G. Tsissios, A. Sallese, *et al.*, "In Vivo Imaging of Newt Lens Regeneration: Novel Insights Into the Regeneration Process," Trans. Vis. Sci. Tech. **10**(10), 4 (2021).
24. M. Koster"ILGPU - A Modern GPU Compiler for .Net Programs," https://ilgpu.net/.
25. S. Rao D. S., M. Jensen, L. Grüner-Nielsen, *et al.*, "Shot-noise limited, supercontinuum-based optical coherence tomography," Light: Sci. Appl. **10**(1), 133 (2021).
26. G. Tsissios, G. Theodoroudis-Rapp, W. Chen, *et al.*, "Characterizing the lens regeneration process in Pleurodeles waltl," Differentiation **132**, 15–23 (2023).
27. A. Mariampillai, B. A. Standish, E. H. Moriyama, *et al.*, "Speckle variance detection of microvasculature using swept-source optical coherence tomography," Opt. Lett. **33**(13), 1530–1532 (2008).
28. "Open Toolkit," OpenTK 2023, https://opentk.net/.
29. "OpenCV," 2023, https://opencv.org/.