# C++ software integration for a high-throughput phase imaging platform

Mikhail E. Kandel[1*], Zelun Luo[1], Kevin Han[1], Gabriel Popescu[1]
[1]Quantitative Light Imaging Laboratory, Department of Electrical and Computer Engineering, Beckman Institute of Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
*Corresponding author: kandel3@illinois.edu

## ABSTRACT

The multi-shot approach in SLIM requires reliable, synchronous, and parallel operation of three independent hardware devices – not meeting these challenges results in degraded phase and slow acquisition speeds, narrowing applications to holistic statements about complex phenomena. The relative youth of quantitative imaging and the lack of ready-made commercial hardware and tools further compounds the problem as Higher level programming languages result in inflexible, experiment specific instruments limited by ill-fitting computational modules, resulting in a palpable chasm between promised and realized hardware performance. Furthermore, general unfamiliarity with intricacies such as background calibration, objective lens attenuation, along with spatial light modular alignment, makes successful measurements difficult for the inattentive or uninitiated. This poses an immediate challenge for moving our techniques beyond the lab to biologically oriented collaborators and clinical practitioners.

To meet these challenges, we present our new Quantitative Phase Imaging pipeline, with improved instrument performance, friendly user interface and robust data processing features, enabling us to acquire and catalog clinical datasets hundreds of gigapixels in size.

## 1. INTRODUCTION

The quantitative phase image represents a per-pixel map of the optical path length across the specimen, providing *intrinsic* contrast from the structure of the object. By decoupling the image from the properties of the microscope, quantitative phase imaging techniques provide reproducible diagnostic markers well suited for clinical applications such as blood screening or cancer diagnosis [1-5].

In Spatial Light Interference Microscopy (SLIM [6]) the relative phase shift between scattered and transmitted light is manipulated by way of a spatial light modulator (SLM). In our implementation, each SLIM image is reconstructed from four 'phase contrast' frames acquired at 90 degree offsets between scattered and transmitted light, with the relative offset between the two components used to reconstruct a per-pixel map of the optical path across the specimen. Among the advantages of phase shifting interferometry[7, 8] is the efficient use of camera bandwidth when compared to techniques where the image is retrieved by Fourier demodulation[9], necessitating significantly more pixels to properly sample the modulated image.

The SLIM hardware sits at the output port of a conventional white light microscope with the SLM conjugate to phase ring of the objective. This design is particular well paired with fluorescence microscopy as the low intensity fluorescent signal moves through the hardware relatively unperturbed when compared to diffraction gratings[10], lenslet arrays [11], and multipath setups[12-14] that split the weak emitted light beyond the sensitivity of the camera.

## 2. INSTRUMENTATION

The acquisition of a SLIM frame begins by writing a phase mask onto the SLM. Like a conventional liquid crystal display, the response time for a phase shifting SLM depends on the values written, and is on the order of 10 ms [15]. Nevertheless, to *achieve* this performance, in the case of a computer display connected SLM device, the pixels must *actually* switch on demand. In our system, we run a dedicated OpenGL render context with a condition

Quantitative Phase Imaging, edited by Gabriel Popescu, YongKeun Park, Proc. of SPIE Vol. 9336, 93360Z · © 2015 SPIE · CCC code: 1605-7422/15/$18 · doi: 10.1117/12.2080212

variable releasing the buffer, effectively implementing a software trigger (Code 1). We note that by creating the context from scratch, we can disable the post-processing and motion blurring that is desirable for video games and word processing applications, but otherwise result in an increase per-frame latency in the phase modulating applications.

Code 1: Synchronous pixel changing on a DVI SLM provides predictable software triggering.

```
//GL thread
while (!glDone)
{
	{
		std::unique_lock<std::mutex> lk(m);
		cv.wait(lk, [&]{return ping; });
	}
	ping = false;
	render()
	SwapBuffers(g_hDC); // on gdi
	{
		std::lock_guard<std::mutex> lk(m);
		pong = true;
	}
	cv.notify_one();
}

//From  control thread
frame = framenum;
{
	std::lock_guard<std::mutex> lk(m);
	ping = true;
}
cv.notify_one();
{
	std::unique_lock<std::mutex> lk(m);
	cv.wait(lk, [&]{return pong; });
	pong = false;
}
```

By default, there is a few millisecond latency between notifying a condition variable and returning control to the associated thread (Code 2). On a Windows platform, Multimedia Timers provide a workaround to achieve response times on the order of Windows NT thread quanta, achieving near optimal response time, reducing jitter due to multithreading to under a millisecond[16]:

Code 2: Multimedia timers improve condition variable response times.

```
struct TimePriority
{
	TimePriority()
	{// should only be instantiated once per thread
		auto err = timeBeginPeriod(p);
		assert(err == TIMERR_NOERROR);
	}
	~TimePriority()
	{
		timeEndPeriod(p);
	}
	static const int p = 1;
};
```

Although hardware camera triggering can be used for time critical acquisitions, instead we opt to use software based camera triggering as it enables us to implicitly adjust for variable image processing time, in particular for in the live acquisition mode. We perform the software triggering and datacopy operations on separate threads, enabling the camera API to respond as data becomes available. Moreover, by decoupling triggering from datacopy we can reduce camera aperture jitter, as triggering is usually immediate while waiting for the data to arrive is subject to other factors such as the computational load.

Writing data to a hard disk at our acquisition speeds is challenging because, even with the fastest solid state drives, the archived speed can vary as a function of capacity, leading to the so-called "SSD write cliff"[17]. Our implementation addresses this problem by performing file writing asynchronously reserving a memory buffer corresponding to the computer's storage capacity letting us briefly exceed hard disk capacity. Further performance increases come from writing the files as binary data and avoiding the operating systems internal file buffering (Code 3).

Code 3: IO created without buffering achieves near theoretical performance.

```
auto hFile = CreateFileA(
       name, GENERIC_WRITE, 0, NULL, CREATE_NEW,
       FILE_FLAG_WRITE_THROUGH | FILE_FLAG_NO_BUFFERING,
       NULL);
```

As the mechanical motion of the microscope takes a variable amount of time to complete, tasks such as opening a shutter or moving the stage require synchronous operation where control returns to the program upon completion. Nevertheless, it is desirable to do such tasks in parallel, for example modulating the SLM while moving to a new position or switch to a reflection fluorescence filter cube while turning off the transmitted illumination. We group such task blocks by launching them asynchronously waiting for their completion through the `std::future` framework. Further, the destruction of purpose built objects ensures a minimal execution time effectively converting portions of the code from asynchronous to synchronous ensuring predictable computation times(Code 4). With these two techniques, the total time becomes the maximum of each step rather than sum of each step - *i.e.*, max(slm,microscope) rather than slm+microscope - and is enforced implicitly without any a-priori knowledge of the execution time of the constituent elements.

Code 4: the `TimeGuarantee` object ensures a minimum execution time of a particular section of code.

```
struct TimeGuarantee
{
   ...
       static inline void fence()
       {
              std::atomic_signal_fence(std::memory_order_seq_cst);
       }
       TimeGuarantee(unsigned int Miliseconds) : milis(Miliseconds), start(timestamp())
       {
              fence();//don't optomize out
       }
       ~TimeGuarantee()
       {
              fence();
              auto left = milis - (timestamp() - start);
              if (left>0)
              {
                     _sleep(left);// more precise compared to std::
              }
       }
};
```

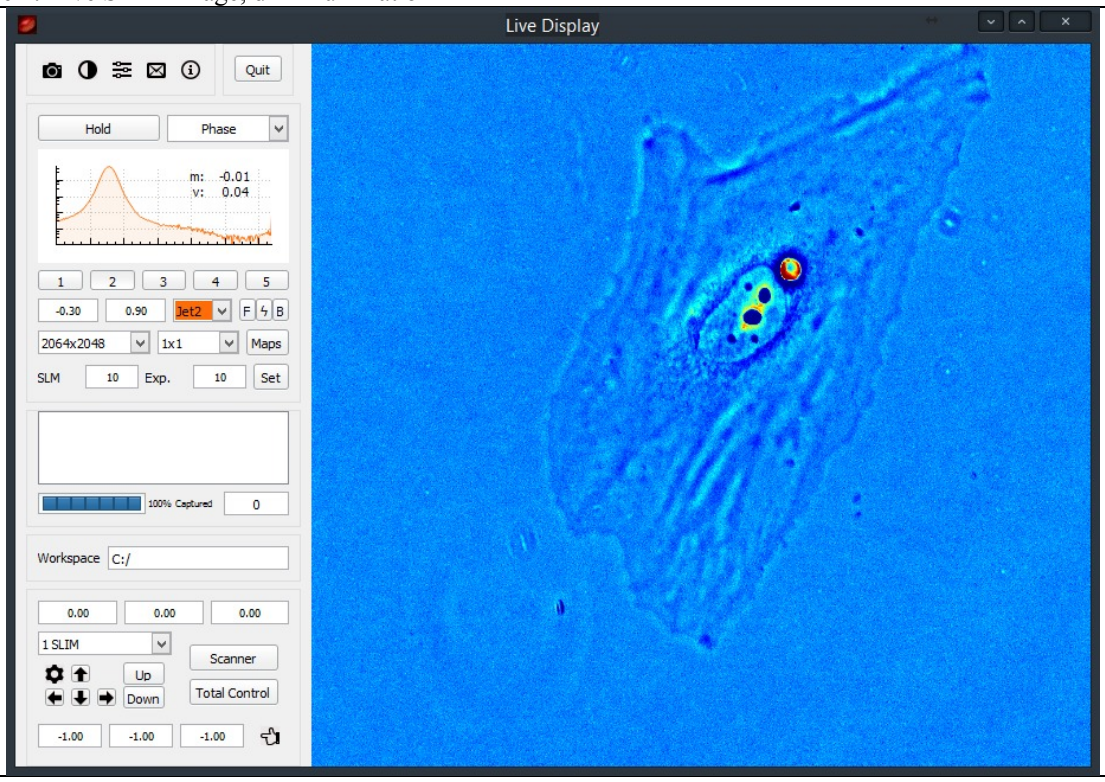
Code 5: Two methods for implicitly overlapping execution of synchrnous (

```
{
        TimeGuarantee t(slm_stability);
        slm->setFrame(slm_pattern);//0ms, asynchronous
        camera->setExposure(expo);//1ms worst case
        auto future1 = camera->applySettings(chan.config);//3 seconds worst case
        auto future2 = scope->moveTo(pos);//90 ms for z, 130 ms xy
        future1.wait();//MSVC futures don't wait on destruction
        future2.wait();
}
```

## 3. USER INTERFACE

To present the SLIM image as a unified contrast setting, in the same way as Bright Field or Phase Contrast, we use CUDA and OpenGL to perform real-time rendering on a separate drawing thread. Passive mechanisms, such as write-combined memory were considered, but not used due conflicts with OEM camera hardware. Instead, we perform the data transfer on a second thread achieving a similar effect, where the next frame loads while the current one is processed. In total, the live mode runs three threads: one for triggering and data upload, one for waiting on new data, and one for rendering the live image (Figure 1).

Figure 1: Live SLIM Image, dim illumination




The acquisition process reads a list of acquisition events, with each event indexing the corresponding channel information. Information stored in each channel includes items such as exposure time, along with QPI specific parameters like the background illumination (Figure 2). To maximize code reusability, OEM SDK are hidden in a particular implementation of generic devices – *i.e.* OspreyCamera implements Camera. The acquisition list enables us to decouple the graphical user interface (Figure 3) from the acquisition process, greatly improving code reusability, as only the list generation function needs to be changed when modifying the user interface or adding features to the acquisition process. The principle downside of such a structure being the need to use synchronization primitives when changing the list during acquisition, although the persistence of well-defined acquisition events facilitates an error correction procedure where a failed event can be easily re-attempted.

Figure 2: Acquisition data structure

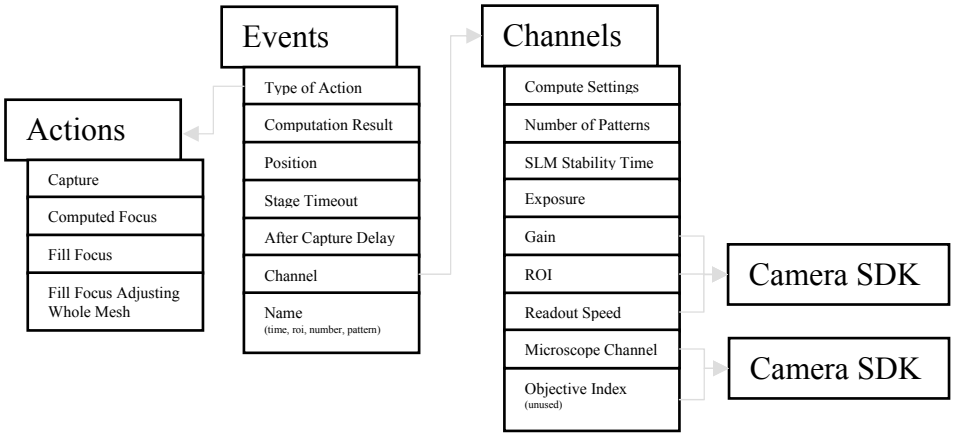
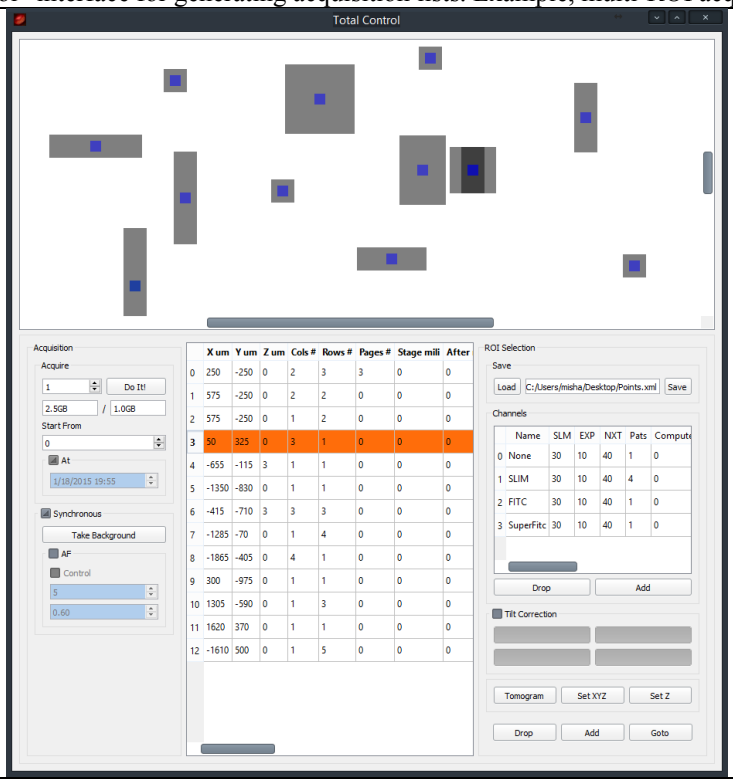
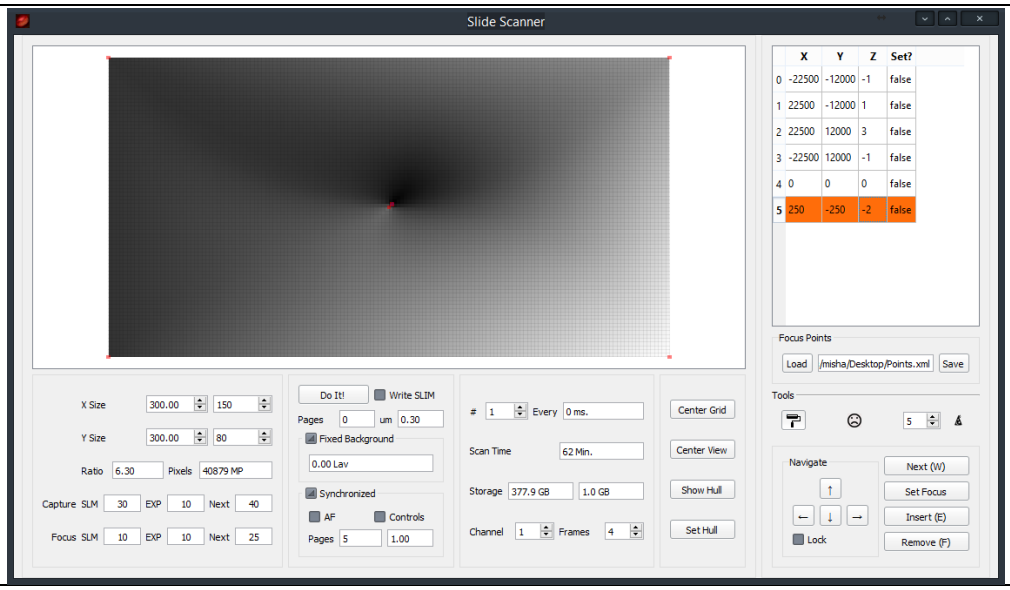



Figure 3: “Total Control” interface for generating acquisition lists. Example, multi-ROI acquisition shown.




Understanding that a clinical slide scanner designed for use by medical professionals must digitize samples whose `surface of best focus` is non-uniform, we incorporate a mechanism to set focus points across the surface of the sample (figure 4). To handle arbitrary locations we generate a Delaunay triangulation and sample it when generating our acquisition list[18]. A slide scan, as would be used to digitize a peripheral blood smear or frozen biopsy section, can use this surface `as is` or take it as a hint for the automated focus system as is addressed in our separate work [19].

Figure 4: Slide Scanner interface displaying focus plain.

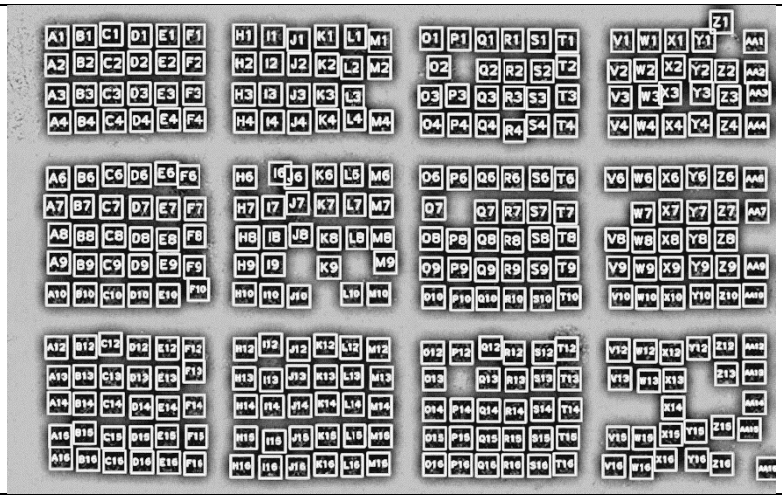



## 4. ARCHIVAL STORAGE

A typical acquisition results in tens of thousands of images, and presents a computational challenge to existing image assembly tools. These tools are often designed for general image alignment tasks such as merging tilings with arbitrary positions or compensating for image warping resulting in producer-consumer schemes with hard to predict threading. For example, some stitching programs are able to align a tile with a variable number of neighbors, leading to parallelism schemes where too few or too many threads are invoked. Instead, we optimize our code for regularly spaced mosaic tiles and rigid transformations[20] facilitating predictable overlap between disk reading and computation, resulting in a code that aligns images comparable to hard disk read speeds. The resulting list is converted into a format compatible with tile based online image viewers, ready for inspection by a medical professional [21, 22].

The intrinsic contrast and quantitative nature of QPI techniques make them well posited for archival storage, because phase, unlike intensity, does not depend on the properties of the instrument but rather only on the actual specimen. To this end, we demonstrate a computational workflow designed to scan a large number slides, processing them in batch and storing them for easy online access. Specifically, a k-means clustering on a down-sampled version identifies the pixels as belonging to a cores and then a second k-means clustering is used to assign the labels (Figure 5) [23].

Figure 5: Stitched and Labeled Microbiopsy Array



## 5. SUMMARY

Our design presents spatial light interference microscopy in a user-friendly manner, unifying a multistep interferometric contrast technique into an image well suited for use by biologists and clinical practitioners. In addition to providing a real time image, our pipeline represents a realization of cloud pathology, with images moving directly from the scanner to a web based image viewer.

## REFERENCES


[1] T. Kim, S. Sridharan, A. Kajdacsy-Balla *et al.*, "Gradient field microscopy for label-free diagnosis of human biopsies," Appl. Opt., 52(1), A92-A96 (2013).

[2] Z. Wang, G. Popescu, K. V. Tangella *et al.*, "Tissue refractive index as marker of disease (Journal Paper)," Journal of Biomedical Optics, 16(11), 116017 (2011).

[3] M. Hunter, V. Backman, G. Popescu *et al.*, "Tissue self-affinity and polarized light scattering in the born approximation: a new model for precancer detection," Physical review letters, 97(13), 138102 (2006).

[4] M. Mir, K. Tangella, and G. Popescu, "Blood testing at the single cell level using quantitative phase and amplitude microscopy," Biomedical Optics Express, 2(12), 3259-3266 (2011).

[5] H. Pham, B. Bhaduri, K. Tangela *et al.*, "Real time blood testing using quantitative phase imaging," PLoS ONE, 8(2), e55676 (2013).

[6] Z. Wang, L. Millet, M. Mir *et al.*, "Spatial light interference microscopy (SLIM)," Optics Express, 19(2), 1016-1026 (2011).

[7] H. Kadono, M. Ogusu, and S. Toyooka, "Phase shifting common path interferometer using a liquid-crystal phase modulator," Optics Communications, 110(3–4), 391-400 (1994).

[8] W. S. Rockward, A. L. Thomas, B. Zhao *et al.*, "Quantitative phase measurements using optical quadrature microscopy," Applied Optics, 47(10), 1684-1696 (2008).

[9] M. Takeda, H. Ina, and S. Kobayashi, "Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry," Journal of the Optical Society of America, 72(1), 156-160 (1982).

[10] G. Popescu, T. Ikeda, R. R. Dasari *et al.*, "Diffraction phase microscopy for quantifying cell structure and dynamics," Optics letters, 31(6), 775-777 (2006).

[11] P. Bon, G. Maucort, B. Wattellier *et al.*, "Quadriwave lateral shearing interferometry for quantitative phase microscopy of living cells," Optics Express, 17(15), 13080-13094 (2009).
[12] P. Kolman, and R. Chmelík, "Coherence-controlled holographic microscope," Optics Express, 18(21), 21990-22003 (2010).
[13] E. Cuche, P. Marquet, and C. Depeursinge, "Simultaneous amplitude-contrast and quantitative phase-contrast microscopy by numerical reconstruction of Fresnel off-axis holograms," Applied Optics, 38(34), 6994-7001 (1999).
[14] G. Popescu, T. Ikeda, C. A. Best *et al.*, "Erythrocyte structure and dynamics quantified by Hilbert phase microscopy," Journal of biomedical optics, 10, 060503 (2005).
[15] G. Thalhammer, R. W. Bowman, G. D. Love *et al.*, "Speeding up liquid crystal SLMs using overdrive with phase change reduction," Optics Express, 21(2), 1779-1797 (2013).
[16] Microsoft, [Multimedia Timers].
[17] C. Mellor, [HP and Violin build Oracle Exadata killer] The Register, (2011).
[18] "Computational Geometry Algorithms Library ".
[19] B. B. Mikhail E. Kandel, Gabriel Popescu, " An efficient autofocusing scheme for quantitative phase imaging " Quantitative Phase Imaging.
[20] E. De Castro, and C. Morandi, "Registration of Translated and Rotated Images Using Finite Fourier Transforms," Pattern Analysis and Machine Intelligence, IEEE Transactions on, PAMI-9(5), 700-703 (1987).
[21] I. Zoomify, [Zoomify].
[22] S. Saalfeld, A. Cardona, V. Hartenstein *et al.*, "CATMAID: collaborative annotation toolkit for massive amounts of image data," Bioinformatics, 25(15), 1984-1986 (2009).
[23] OpenCV, [Clustering].